

# Prediction Competition 6: Text Analysis, Economic News and ML Research

Anonymized name: Yuta Okkotsu

$R^2 = 0.34$

$MSE = 1.32$

Model used: LSTM & NN

## Q2

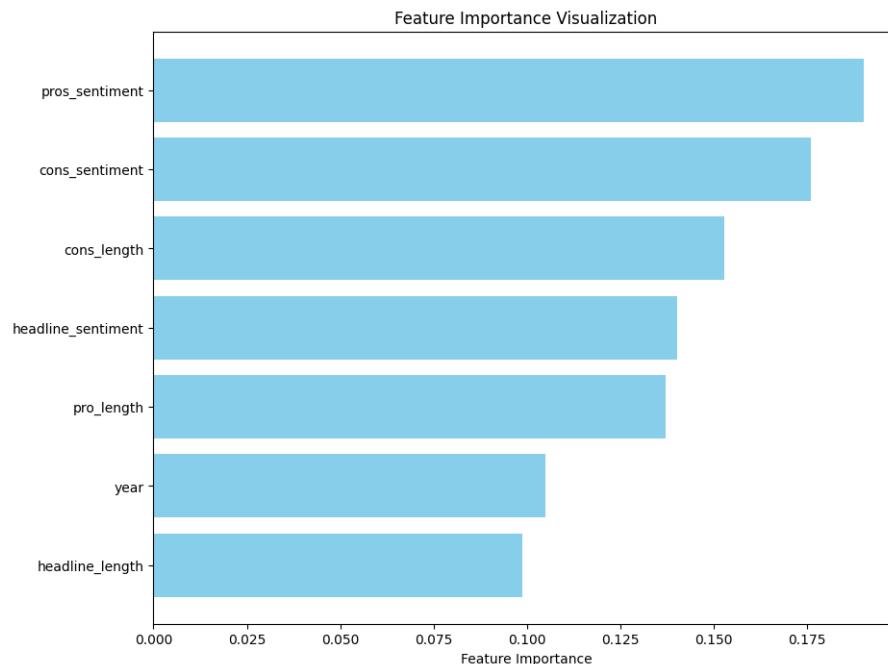


Figure 1: A horizontal bar plot of the importance of the variables in my model

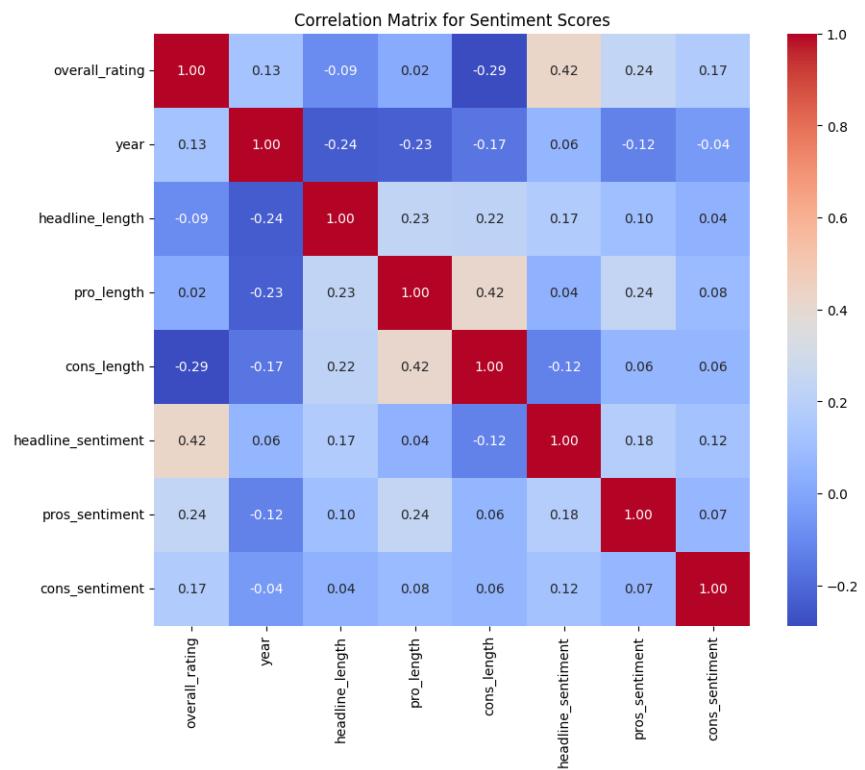


Figure 2: A correlation matrix of the variables in my model

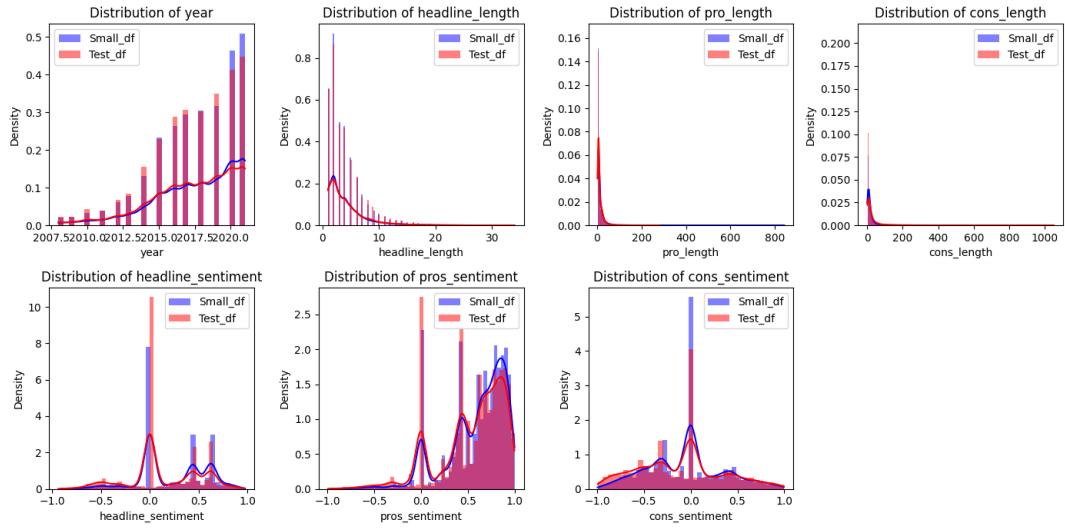


Figure 3: A multiplot graph of the distributions of the variables in the training data vs the test data

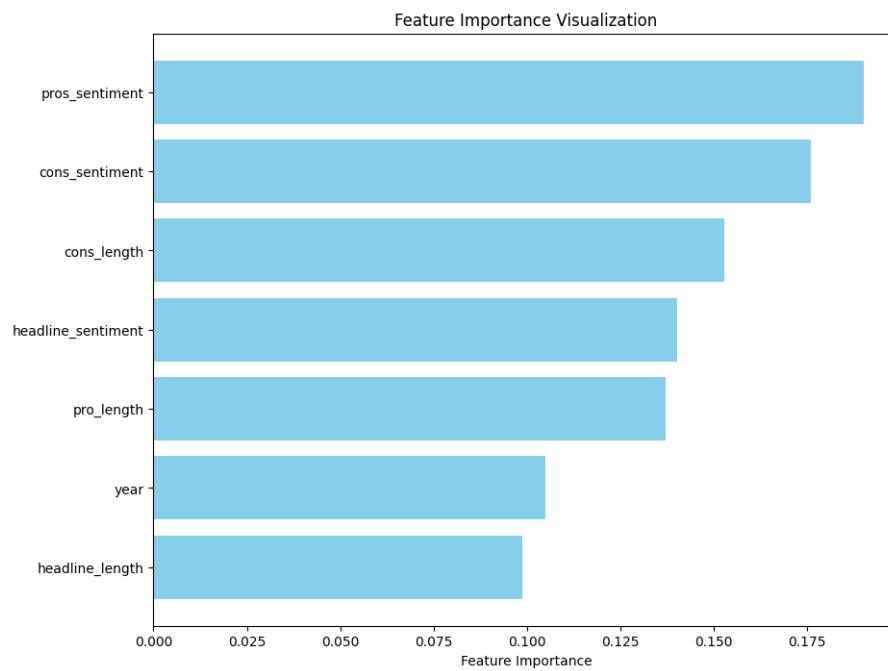


Figure 4: A horizontal bar plot of the importance of the variables in my model

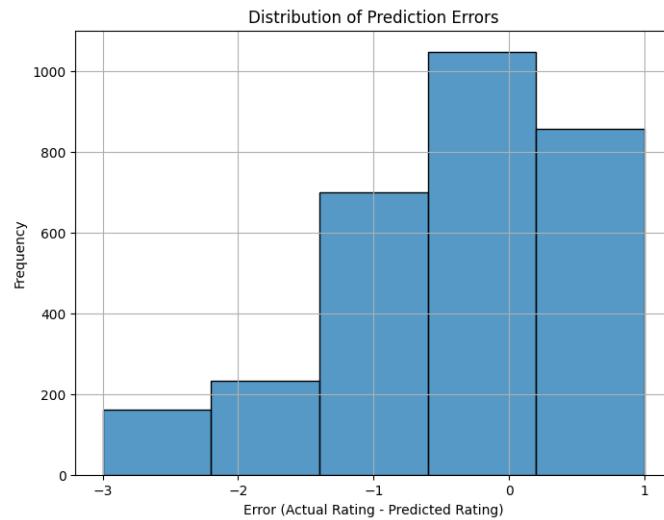


Figure 5: A bar plot of the errors in the prediction values vs the true values



Figure 6: A wordcloud of most common words on 1 star reviews



Figure 7: A wordcloud of most common words on 2 star reviews



Figure 8: A wordcloud of most common words on 3 star reviews



Figure 9: A wordcloud of most common words on 4 star reviews



Figure 10: A wordcloud of most common words on 5 star reviews

### Q3

The actionable insights would be for low-rated companies to focus on improving work-life balance and facility quality, while high-rated companies should continue to enhance employee support programs and flexible working options to maintain or improve their ratings. This tailored approach helps companies prioritize initiatives based on their current ratings and specific feedback trends.

## Q4

You  
sarah-bana-job2vec-wage-premia.pdf  
PDF

using the information discussed in this chat and this pdf can you answer this question  
[challenging] Economist Sarah Bana trained a convolutional neural network model to predict wages based on text. (Paper is available [here](#).) The analysis then uses the machine learning model to calculate the value of different certifications for a worker, and how the valuations for these certifications vary across the wage distribution.  
Use your own model to produce actionable insights into what firms at different points of the quality distribution (as measured by average overall rating) can do to improve their rating.  
Example 1: Perhaps most of the complaints on low-quality workplaces are about lack of restrooms.  
Example 2: Perhaps most of the complaints about high-quality workplaces are about lack of sushi.  
Example 3: Perhaps most of the praise for high-quality restaurants are about ability to work from home.  
In your answer, please briefly explain your methodology and then show a figure/table that summarizes the results neatly.  
The approach used for this Q3 does not have to be related to approach used in Q1 or Q2]

ChatGPT  
To answer the question of how different certifications impact wages across the wage distribution and how this can translate into actionable insights for firms looking to improve their rating, we can draw parallels to Sarah Bana's methodology and apply it to understanding overall ratings in companies based on employee feedback about various aspects (like work environment, benefits, etc.).

**Methodology**

1. **Data Collection & Preparation:**
  - Collect employee reviews, overall ratings, and specific feedback on various workplace aspects from sources like company review websites or internal survey data.
  - Process this text data using NLP techniques to extract themes or common issues, such as complaints about restrooms or praise for work-from-home policies.
2. **Feature Engineering:**
  - Create binary or categorical variables representing the presence of specific words or phrases related to workplace quality (e.g., "restrooms", "work from home").
  - Incorporate additional company metrics or contextual data if available.
3. **Model Building:**
  - Use a machine learning model, potentially a logistic regression or an ordinal regression if the ratings are ordered categories, to predict the overall rating based on extracted features from reviews.
  - Alternatively, employ a text injection methodology similar to Bana's, where synthetic reviews with altered characteristics are used to predict how changes in specific features impact overall ratings.
4. **Analysis of Results:**
  - Identify which features are most predictive of higher or lower ratings.
  - Analyze the variation in these features across different ratings to understand what typically differentiates higher-rated companies from lower-rated ones.
5. **Visualization and Communication:**
  - Create visualizations that clearly show how the presence or absence of certain features affects overall ratings.
  - Prepare a report or dashboard that summarizes these insights for decision-makers at firms.

Figure 11: Conversation with Chat GPT

# ECON 626 Final Prediction Competition Code

**Objective:** Construct features from the available data. Then use these features to predict the overall Glassdoor review rating using any machine learning model, including neural networks.

## Importing libraries

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.preprocessing import LabelEncoder
import nltk
from nltk.stem import PorterStemmer, LancasterStemmer, WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, WhitespaceTokenizer, TreebankWordTokenizer
from nltk.metrics import edit_distance
import string
import re
from bs4 import BeautifulSoup
from wordcloud import WordCloud
from collections import Counter
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn import preprocessing
from sklearn.impute import SimpleImputer
from sklearn import svm, linear_model
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report
from scipy.sparse import hstack
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
from termcolor import colored
import gc
import torch
import torch.nn as nn
import torch.optim as optim
from torchtext.vocab import FastText, vocab
from torch.utils.data import Dataset, DataLoader, TensorDataset
from gensim.models.phrases import Phrases, Phraser
from string import punctuation
from tqdm import tqdm
from tensorflow.keras.utils import to_categorical

warnings.filterwarnings("ignore")
```

```
# Download NLTK data
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package punkt to /Users/andrew/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]   /Users/andrew/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /Users/andrew/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /Users/andrew/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

Out[1]: True

## Importing Data

```
In [2]: Small_data_path = "/Users/andrew/Downloads/UW courses/ECON 626/Final Predict
Large_data_path = "/Users/andrew/Downloads/UW courses/ECON 626/Final Predict
Test_data_path = "/Users/andrew/Downloads/UW courses/ECON 626/Final Predicti

Small_df = pd.read_csv(Small_data_path)
Large_df = pd.read_csv(Large_data_path)
Test_df = pd.read_csv(Test_data_path)
```

```
In [3]: Small_df = Small_df.sample(n=10000, random_state=42)
Test_df = Test_df.sample(n=10000, random_state=42)
Test_df = Test_df.drop(['overall_rating'], axis=1)
```

In [ ]:

## Inspecting data

```
In [4]: Small_df.head()
```

Out [4] :	overall_rating	firm	date_review	job_title	location	headline
75721	5	Deloitte	2015-10-27	Anonymous Employee	South Bend, IN	Friendly culture, positive work environment
80184	5	IBM	2017-12-21	Senior Manager	Pune	Senior Manager
19864	3	McDonald-s	2018-10-18	Crew Member	Laverton, Victoria, Victoria	Great working environment
76699	5	Blue-Yonder	2016-08-10	QA Architect	Bangalore	Best Supply Chain Company
92991	5	FARFETCH	2020-11-06	Head of Product Management	Porto, Oporto, Oporto	An amazing company to work at
In [5] :	<code>Large_df.head()</code>					

Out [5]:	overall_rating	firm	date_review	job_title	location	headline	
0	1	Google	2020-11-18	Intern	NaN	intern thing	i mean was a bu fre
1	1	McDonald-s	2019-06-22	Cashier	Baltimore, MD	Terrible just like the rumors	its a job foods on
2	1	IBM	2014-03-20	Technical Lead	Pune	DOn expect any increments	Wor home, w b
3	1	Oracle	2021-05-18	Senior Sales Enablement Manager	London, England, England	People leave managers not companies	Som colleagues are skill
4	1	IBM	2009-06-01		NaN	IBM not what is once was.	Telecomr Semi-i working

```
In [6]: duplicate_rows = Small_df.duplicated(keep=False)

# Check if there are any duplicates
if duplicate_rows.any():
    print(Small_df[duplicate_rows])
    # Print the rows that are duplicates
else:
    print("No duplicates found.")

      overall_rating   firm date_review          job_title \
11446              2  Oracle  2015-07-02  Applications Sales Manager
10299              2  Oracle  2015-07-02  Applications Sales Manager

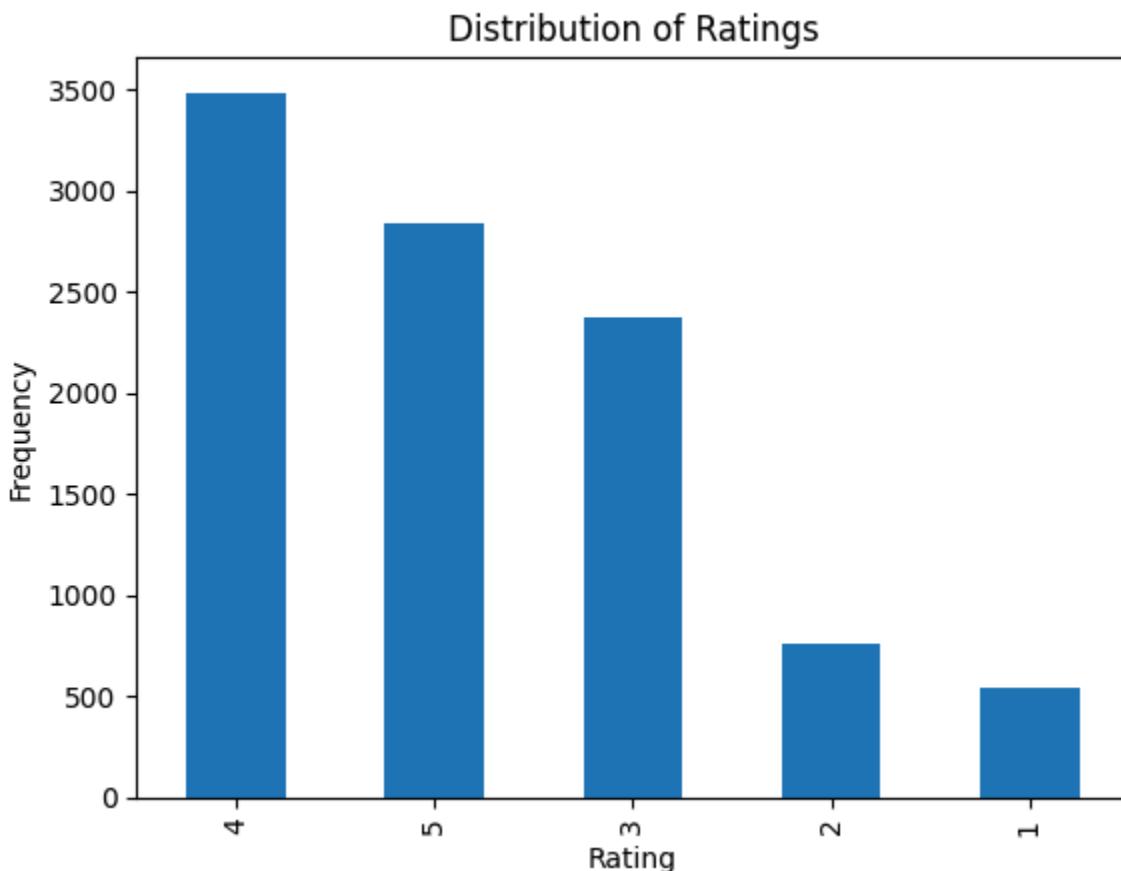
      location                                         headline \
11446 Redwood Shores, CA  Potential for money, but you're either a have ...
10299 Redwood Shores, CA  Potential for money, but you're either a have ...

      pros \
11446 Lots of potential commissions.\rGreat work/lif...
10299 Lots of potential commissions.\rGreat work/lif...

      cons   year
11446 You will definitely feel like a cog in a machi...  2015
10299 You will definitely feel like a cog in a machi...  2015
```

```
In [7]: import matplotlib.pyplot as plt

# Plot the distribution of the target variable
Small_df['overall_rating'].value_counts().plot(kind='bar')
plt.title('Distribution of Ratings')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.show()
```



```
In [ ]: # Calculate the length of each text entry
Small_df['headline_length'] = Small_df['headline'].apply(lambda x: len(x.split()))
Small_df['pro_length'] = Small_df['pros'].apply(lambda x: len(x.split()))
Small_df['cons_length'] = Small_df['cons'].apply(lambda x: len(x.split()))

# Get descriptive statistics
length_stats = df['length'].describe()
percentiles = np.percentile(df['length'], [90, 95, 99])
```

## Data Cleaning

```
In [8]: stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

def text_processing(text:str) -> str:
    # Remove punctuation
    clean_text = ''.join([char for char in text if char not in string.punctuation])
```

```

# Substitute emoticons
clean_text = re.sub(r'<3', '<heart>', clean_text)
clean_text = re.sub(r"[8:-;][`\-\]?[d]+", '<smile>', clean_text)
clean_text = re.sub(r"[8:-;][`\-\]?(+", '<sadface>', clean_text)
clean_text = re.sub(r"[8:-;][`\-\]?[\|l*]", '<neutralface>', clean_text)
clean_text = re.sub(r"[8:-;][`\-\]?p+", '<lolface>', clean_text)
# Remove non-alphabetic characters
clean_text = re.sub('[^A-Za-z ]+', '', clean_text)
# Tokenize, remove stopwords, stem, and lemmatize
clean_text = ' '.join([stemmer.stem(lemmatizer.lemmatize(word.lower()))
                      for word in word_tokenize(clean_text)
                      if word.lower() not in stopwords.words('english')])
return clean_text

```

```

In [9]: import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Download the VADER lexicon
nltk.download('vader_lexicon')

def sentiment_scores(text):
    analyzer = SentimentIntensityAnalyzer()
    scores = analyzer.polarity_scores(text)
    return scores['compound'] # Compound score ranges from -1 (most negative)

import pandas as pd
from sklearn.impute import SimpleImputer

def data_cleaning(input_df):
    # Convert 'date_review' column to datetime
    input_df['date_review'] = pd.to_datetime(input_df['date_review'])

    # Fill missing values in object type columns with 'NA'
    object_columns = input_df.select_dtypes(include='object').columns
    input_df[object_columns] = input_df[object_columns].fillna('NA')

    if 'overall_rating' in input_df.columns:
        # Impute missing values in numerical columns with mean
        numerical_columns = input_df.select_dtypes(include=[int, float]).columns
        imputer = SimpleImputer(strategy='mean')
        input_df[numerical_columns] = imputer.fit_transform(input_df[numerical_columns])
    else:
        pass

    # Remove duplicate rows based on 'headline', 'pros', and 'cons' columns
    input_df = input_df.drop_duplicates(subset=['headline', 'pros', 'cons'])

    # Reset index
    input_df = input_df.reset_index(drop=True)

    # Convert certain columns to string type and fill missing values with empty string
    for col in ['headline', 'pros', 'cons']:
        input_df[col] = input_df[col].astype(str).fillna('')

    # Calculate the length of each text entry
    input_df['headline_length'] = input_df['headline'].apply(lambda x: len(x))

```

```

input_df['pro_length'] = input_df['pros'].apply(lambda x: len(x.split()))
input_df['cons_length'] = input_df['cons'].apply(lambda x: len(x.split()))

# Calculate sentiment scores for 'headline', 'pros', and 'cons'
# Assuming sentiment_scores is a function you have defined elsewhere
input_df['headline_sentiment'] = input_df['headline'].apply(sentiment_scores)
input_df['pros_sentiment'] = input_df['pros'].apply(sentiment_scores)
input_df['cons_sentiment'] = input_df['cons'].apply(sentiment_scores)

return input_df

```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /Users/andrew/nltk_data...
[nltk_data] Package vader_lexicon is already up-to-date!
```

In [10]: `Small_df = data_cleaning(Small_df)`

In [11]: `Test_df = data_cleaning(Test_df)`

In [12]: `len(Small_df)`

Out[12]: 9999

```

In [13]: import pandas as pd
from tqdm import tqdm # Import tqdm
tqdm.pandas() # Enable progress_apply for Pandas Series

# Apply text_processing function with progress bar
Small_df['headline'] = Small_df['headline'].progress_apply(text_processing)
Small_df['pros'] = Small_df['pros'].progress_apply(text_processing)
Small_df['cons'] = Small_df['cons'].progress_apply(text_processing)

Test_df['headline'] = Test_df['headline'].progress_apply(text_processing)
Test_df['pros'] = Test_df['pros'].progress_apply(text_processing)
Test_df['cons'] = Test_df['cons'].progress_apply(text_processing)

```

100%	9999/9999	[00:03<00:00, 3001.82it/s]
100%	9999/9999	[00:08<00:00, 1208.82it/s]
100%	9999/9999	[00:11<00:00, 907.08it/s]
100%	9996/9996	[00:02<00:00, 3812.21it/s]
100%	9996/9996	[00:08<00:00, 1222.83it/s]
100%	9996/9996	[00:15<00:00, 664.32it/s]

In [14]: `Small_df`

Out[14]:	overall_rating	firm	date_review	job_title	location	headline	
0	5.0	Deloitte	2015-10-27	Anonymous Employee	South Bend, IN	friendly culture positive work environment	good place to work
1	5.0	IBM	2017-12-21	Senior Manager	Pune	senior management	flexible working
2	3.0	McDonald-s	2018-10-18	Crew Member	Laverton, Victoria, Victoria	great work environment	usually under pressure
3	5.0	Blue-Yonder	2016-08-10	QA Architect	Bangalore	best supply chain company	interesting learning challenges
4	5.0	FARFETCH	2020-11-06	Head of Product Management	Porto, Oporto, Oporto	amazon company work	opportunity lead
...	...	...	...	...	...	...	...
9994	1.0	KPMG	2021-03-30	Technical Consultant	Gurgaon, Haryana, Haryana	dynamax technical consultant	brand recognition
9995	3.0	Thomson-Reuters	2017-07-24	Order Management Specialist	NA	great experience difficult task	work demanding
9996	5.0	Microsoft	2016-08-06	Anonymous Employee	NA	great job reception	would recommend
9997	5.0	Oracle	2013-05-13	Principal Sales Consultant	NA	great place to work	factor in case of turn

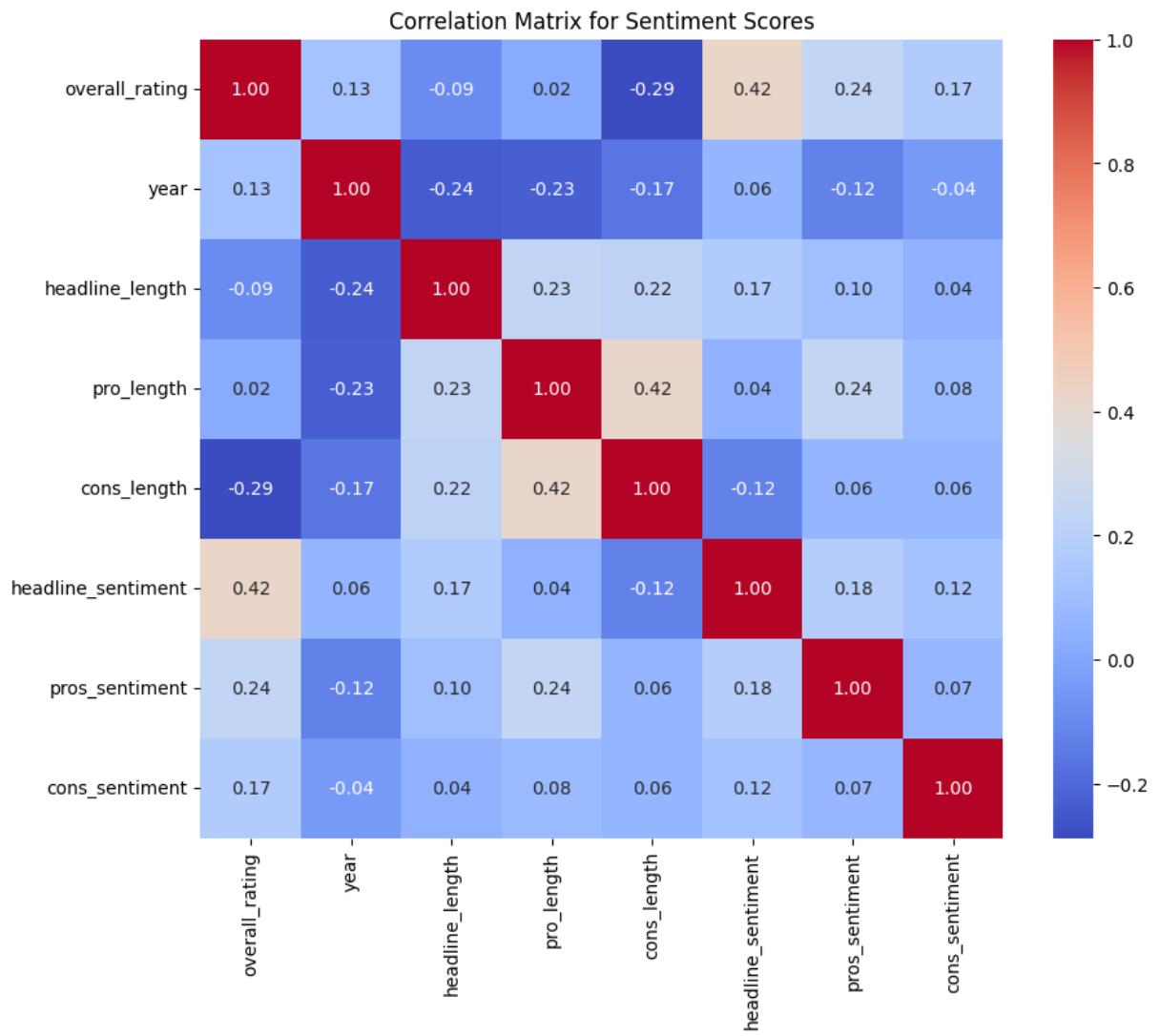
	overall_rating	firm	date_review	job_title	location	headline
9998	5.0	Microsoft	2020-07-14	Global Supplier Management	Ciudad de Mexico	excel workplace people

9999 rows × 15 columns

```
In [15]: import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'headline_sentiment', 'pros_sentiment', and 'cons_sentiment' are
numeric_columns = Small_df.select_dtypes(include=[int, float]).columns
correlation_matrix = Small_df[numeric_columns].corr()

# Plotting the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True)
plt.title('Correlation Matrix for Sentiment Scores')
plt.show()
```

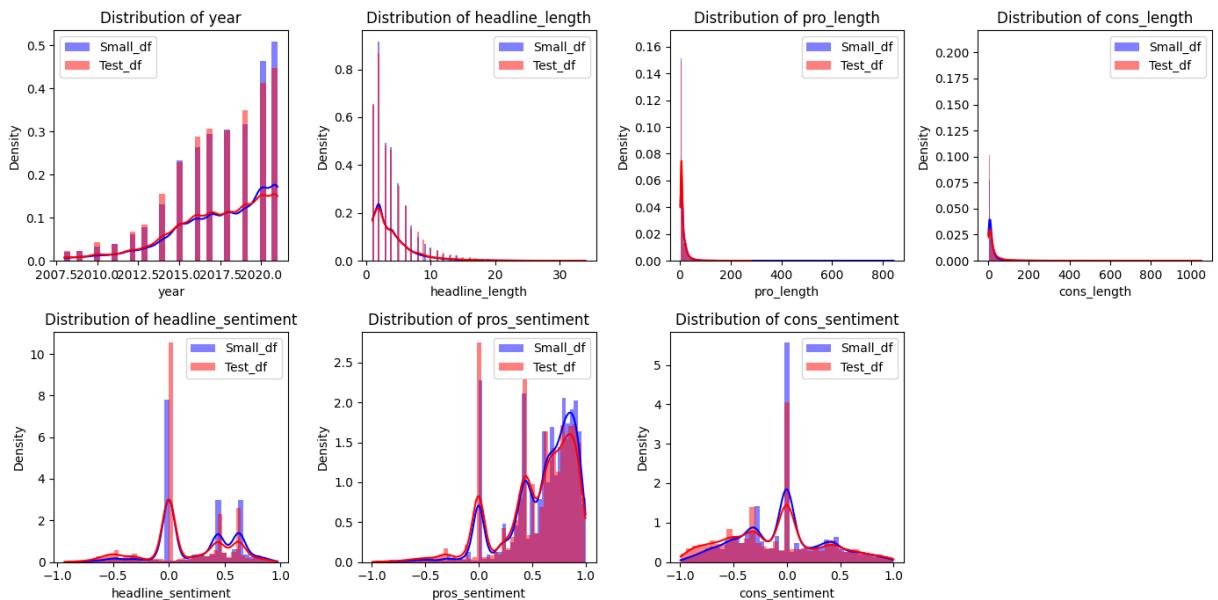


```
In [16]: numeric_columns = [column for column in numeric_columns if column != 'overall']
numeric_columns
```

```
Out[16]: ['year',
          'headline_length',
          'pro_length',
          'cons_length',
          'headline_sentiment',
          'pros_sentiment',
          'cons_sentiment']
```

```
In [17]: # Plot histograms for sentiment scores comparison
plt.figure(figsize=(14, 7))

for i, column in enumerate(numeric_columns, 1):
    plt.subplot(2, 4, i)
    sns.histplot(Small_df[column], color="blue", label='Small_df', kde=True,
                 sns.histplot(Test_df[column], color="red", label='Test_df', kde=True, st
    plt.legend()
    plt.title(f'Distribution of {column}')
```



PADDING!

```
In [18]: # Define x_cols as all columns of Small_df except 'overall_rating'
x_cols = [col for col in Small_df.columns if col != 'overall_rating']

x_cols
```

```
Out[18]: ['firm',
          'date_review',
          'job_title',
          'location',
          'headline',
          'pros',
          'cons',
          'year',
          'headline_length',
          'pro_length',
          'cons_length',
          'headline_sentiment',
          'pros_sentiment',
          'cons_sentiment']
```

```
In [ ]: numerical_columns = Small_df.select_dtypes(include=[int, float]).columns
```

## train test split

```
In [22]: from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import StandardScaler

# Combine headline, pros, and cons into a single column for tokenization
Small_df['combined_text'] = Small_df['headline'] + ' ' + Small_df['pros'] +
    Small_df['cons']

# Tokenize the combined text data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(Small_df['combined_text'])

# Convert text data to sequences
sequences = tokenizer.texts_to_sequences(Small_df['combined_text'])

# Pad sequences
max_sequence_length = 100 # Choose an appropriate sequence length
padded_sequences = pad_sequences(sequences, maxlen=max_sequence_length, padding='post')

# Standardize sentiment scores
scaler = StandardScaler()
sentiment_features = Small_df[['headline_sentiment', 'pros_sentiment', 'cons_sentiment']]
sentiment_features_standardized = scaler.fit_transform(sentiment_features)

# Combine padded sequences with sentiment features
X_combined = np.concatenate([padded_sequences, sentiment_features_standardized], axis=1)

# Split into X and y
X = X_combined
y = to_categorical(Small_df['overall_rating'] - 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y)
```

```
In [ ]: #X_train, X_test, y_train, y_test = train_test_split(Small_df[x_cols], Small_d
```

## CNN

```
In [23]: from tensorflow.keras.layers import TextVectorization, Embedding
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Set maximum vocabulary size and sequence length
max_tokens = 10000
max_len = 100 # or use the max_sequence_length you find appropriate based on your data

# Create a TextVectorization layer
vectorizer = TextVectorization(max_tokens=max_tokens, output_sequence_length=max_len)
text_data = Small_df['headline'] + ' ' + Small_df['pros'] + ' ' + Small_df['cons']
vectorizer.adapt(text_data)
```

```
# You can now vectorize your text data
text_vectorized = vectorizer(text_data)
```

```
In [24]: # Assume numerical_features are already scaled as shown in your previous code
numerical_features = Small_df[['year', 'headline_length', 'pro_length', 'correlation']]

# Concatenate numerical features and vectorized text data
import numpy as np
combined_features = np.concatenate([numerical_features, text_vectorized], axis=1)
```

```
In [27]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout

# Assuming combined_features is a NumPy array of your input data ready for training
# Define the model
model = Sequential([
    Flatten(input_shape=(combined_features.shape[1],)), # Flatten input if it's not already
    Dense(128, activation='relu'),
    Dropout(0.5), # Optional: For regularization
    Dense(10, activation='relu'),
    Dense(5, activation='softmax') # 5 output units for 5 rating classes
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Assuming you have already defined y as the one-hot encoded target variable
from sklearn.model_selection import train_test_split

# Split data into training and testing
X_train, X_test, y_train, y_test = train_test_split(combined_features, y, test_size=0.2, random_state=42)

# Train the model
history = model.fit(X_train, y_train, epochs=10, validation_split=0.1, batch_size=32)
```

```
Epoch 1/10
197/197 1s 1ms/step - accuracy: 0.2420 - loss: 219.7993
- val_accuracy: 0.3457 - val_loss: 1.5528
Epoch 2/10
197/197 0s 635us/step - accuracy: 0.3296 - loss: 4.1886
- val_accuracy: 0.3457 - val_loss: 1.5020
Epoch 3/10
197/197 0s 604us/step - accuracy: 0.3423 - loss: 2.6101
- val_accuracy: 0.3457 - val_loss: 1.4679
Epoch 4/10
197/197 0s 590us/step - accuracy: 0.3442 - loss: 1.6588
- val_accuracy: 0.3457 - val_loss: 1.4455
Epoch 5/10
197/197 0s 771us/step - accuracy: 0.3524 - loss: 1.6081
- val_accuracy: 0.3457 - val_loss: 1.4310
Epoch 6/10
197/197 0s 526us/step - accuracy: 0.3462 - loss: 1.6793
- val_accuracy: 0.3457 - val_loss: 1.4222
Epoch 7/10
197/197 0s 521us/step - accuracy: 0.3423 - loss: 1.4569
- val_accuracy: 0.3457 - val_loss: 1.4163
Epoch 8/10
197/197 0s 551us/step - accuracy: 0.3575 - loss: 1.5736
- val_accuracy: 0.3457 - val_loss: 1.4127
Epoch 9/10
197/197 0s 643us/step - accuracy: 0.3432 - loss: 1.4568
- val_accuracy: 0.3457 - val_loss: 1.4104
Epoch 10/10
197/197 0s 538us/step - accuracy: 0.3398 - loss: 1.4496
- val_accuracy: 0.3457 - val_loss: 1.4089
```

```
In [28]: from sklearn.model_selection import train_test_split

# Splitting the data - ensure y is the one-hot encoded labels
X_train, X_test, y_train, y_test = train_test_split(combined_features, y, test_size=0.2, random_state=42)

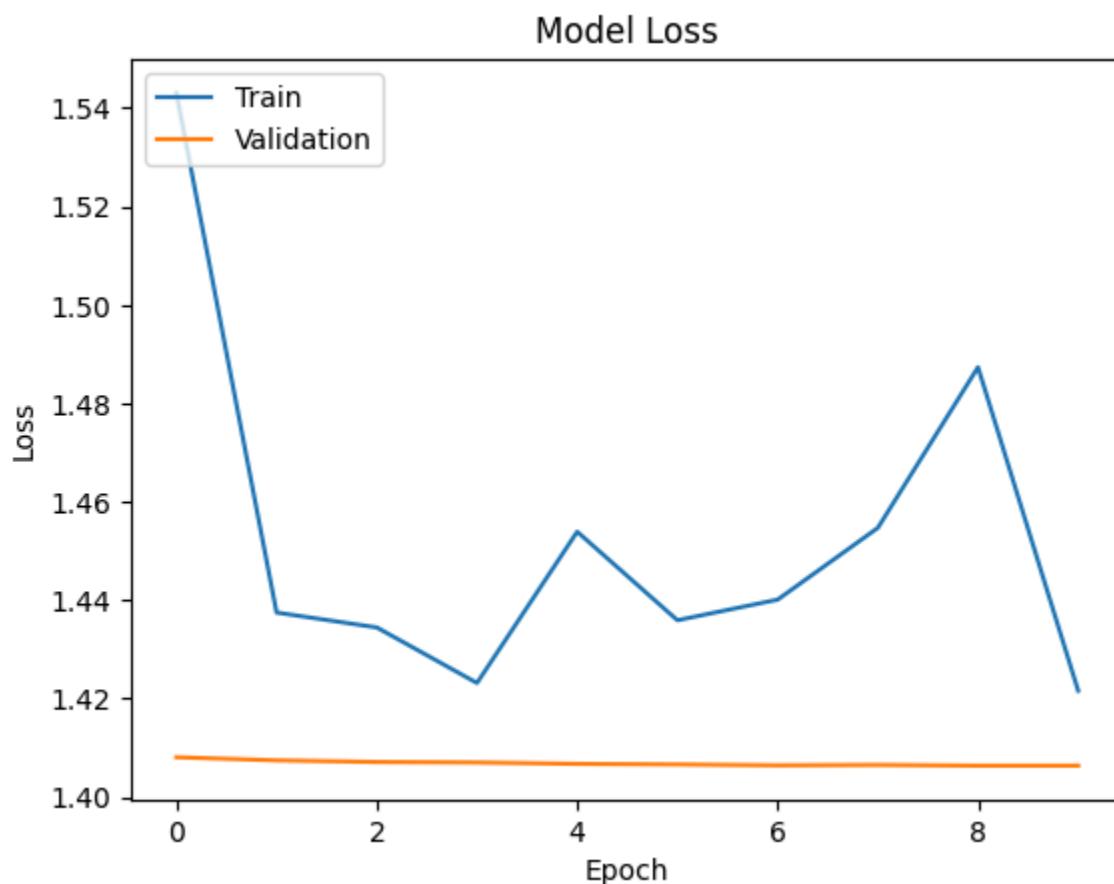
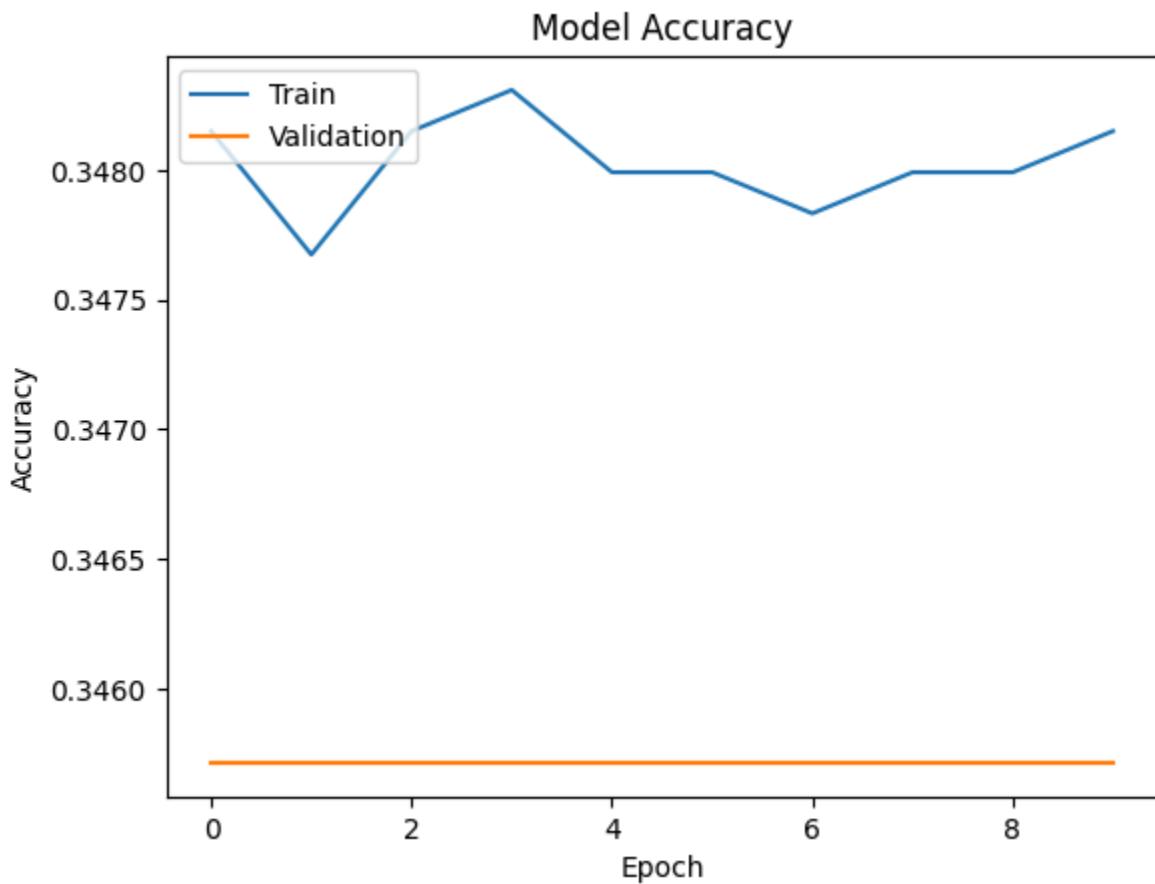
# Train the model
history = model.fit(X_train, y_train, epochs=10, validation_split=0.1, batch_size=32)
```

```
Epoch 1/10
197/197 0s 850us/step - accuracy: 0.3486 - loss: 1.5582
- val_accuracy: 0.3457 - val_loss: 1.4081
Epoch 2/10
197/197 0s 528us/step - accuracy: 0.3513 - loss: 1.4252
- val_accuracy: 0.3457 - val_loss: 1.4075
Epoch 3/10
197/197 0s 546us/step - accuracy: 0.3456 - loss: 1.4315
- val_accuracy: 0.3457 - val_loss: 1.4071
Epoch 4/10
197/197 0s 554us/step - accuracy: 0.3470 - loss: 1.4260
- val_accuracy: 0.3457 - val_loss: 1.4070
Epoch 5/10
197/197 0s 697us/step - accuracy: 0.3459 - loss: 1.4277
- val_accuracy: 0.3457 - val_loss: 1.4068
Epoch 6/10
197/197 0s 567us/step - accuracy: 0.3516 - loss: 1.4233
- val_accuracy: 0.3457 - val_loss: 1.4066
Epoch 7/10
197/197 0s 601us/step - accuracy: 0.3614 - loss: 1.4225
- val_accuracy: 0.3457 - val_loss: 1.4065
Epoch 8/10
197/197 0s 559us/step - accuracy: 0.3482 - loss: 1.4610
- val_accuracy: 0.3457 - val_loss: 1.4065
Epoch 9/10
197/197 0s 553us/step - accuracy: 0.3542 - loss: 1.5973
- val_accuracy: 0.3457 - val_loss: 1.4064
Epoch 10/10
197/197 0s 648us/step - accuracy: 0.3475 - loss: 1.4262
- val_accuracy: 0.3457 - val_loss: 1.4064
```

```
In [29]: import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



```
In [30]: # Assuming X_test is your test features and model is already trained
y_pred = model.predict(X_test)
y_pred_classes = y_pred.argmax(axis=1) # Convert probabilities to class lat
```

94/94 ━━━━━━ 0s 1ms/step

```
In [31]: y_true = y_test.argmax(axis=1)
```

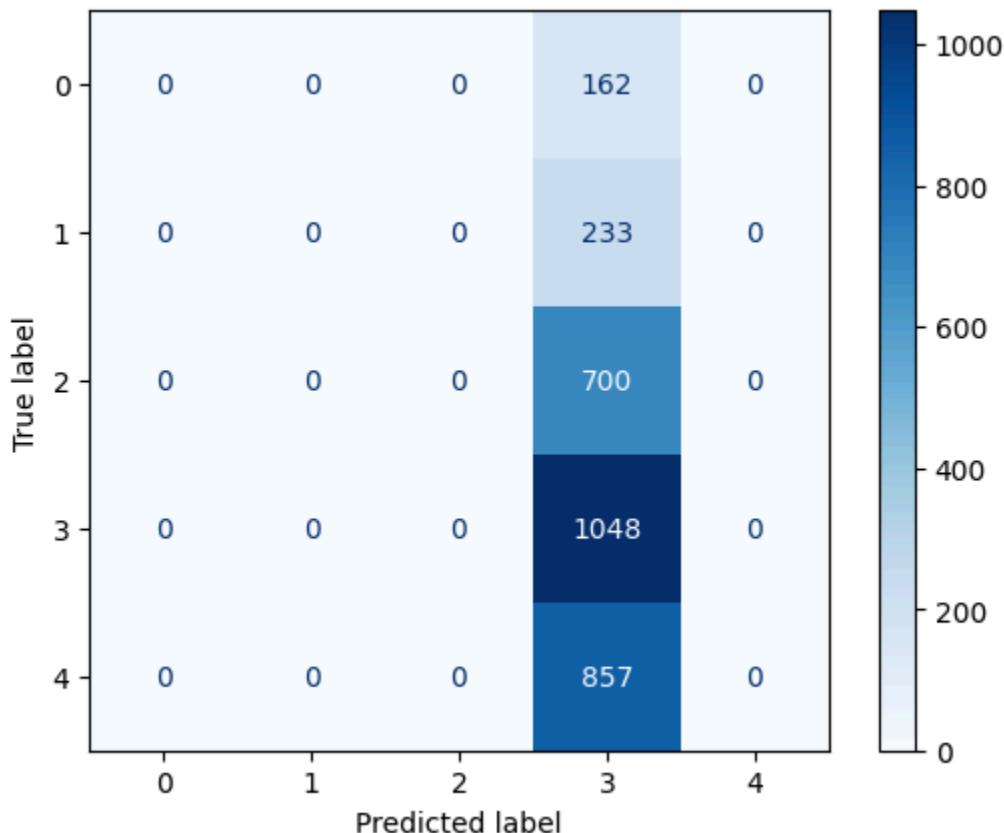
```
In [32]: from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

```
# Compute the confusion matrix
cm = confusion_matrix(y_true, y_pred_classes)

# Print the confusion matrix
print(cm)

# Optionally display the confusion matrix using ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.show()
```

```
[[ 0   0   0  162   0]
 [ 0   0   0  233   0]
 [ 0   0   0  700   0]
 [ 0   0   0 1048   0]
 [ 0   0   0  857   0]]
```



```
In [33]: from sklearn.metrics import classification_report
```

```
# Print classification report
print(classification_report(y_true, y_pred_classes))

precision    recall  f1-score   support

          0       0.00      0.00      0.00      162
          1       0.00      0.00      0.00     233
          2       0.00      0.00      0.00      700
          3       0.35      1.00      0.52    1048
          4       0.00      0.00      0.00      857

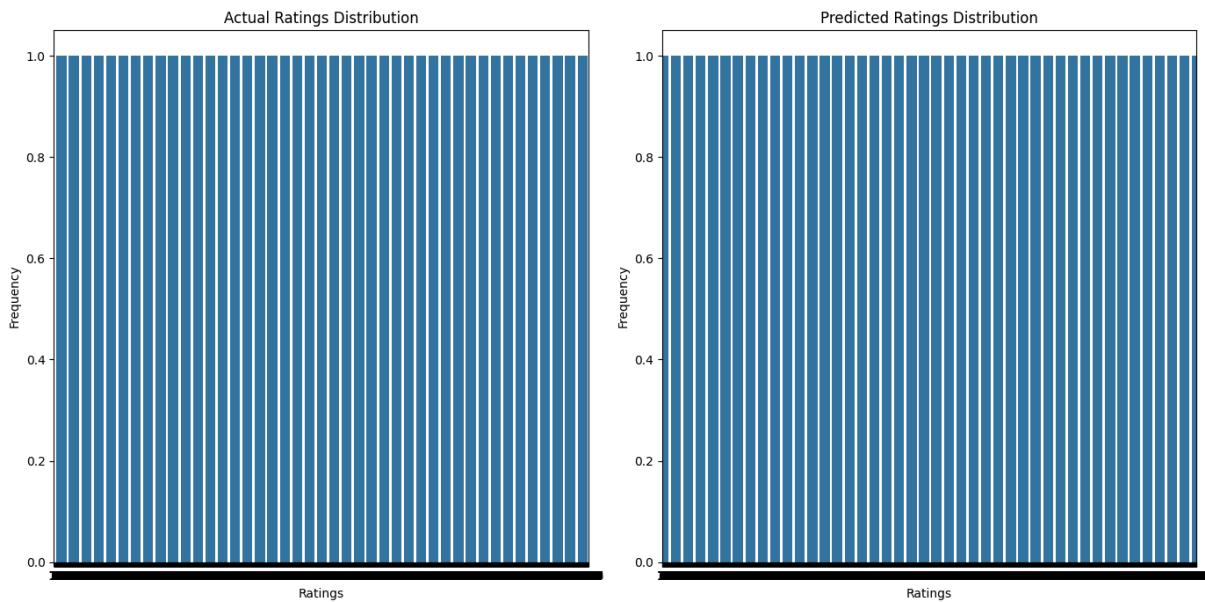
   accuracy                           0.35    3000
macro avg       0.07      0.20      0.10    3000
weighted avg    0.12      0.35      0.18    3000
```

In [37]: # Calculate errors  
errors = y\_true - y\_pred\_classes

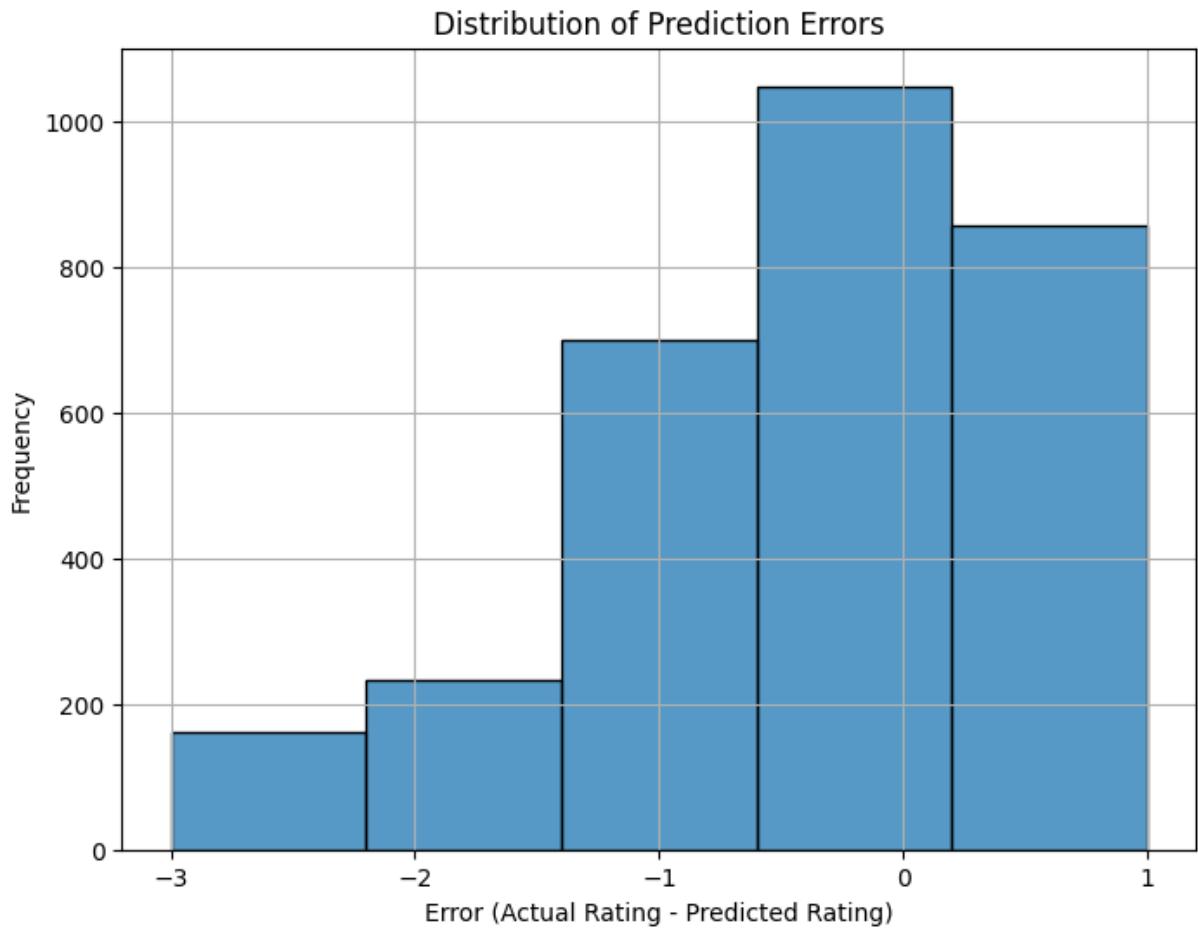
In [107...]: from sklearn.metrics import mean\_squared\_error  
  
mse = mean\_squared\_error(y\_true, y\_pred\_classes)  
print("Mean Squared Error:", mse)

Mean Squared Error: 1.3156666666666668

In [38]: import matplotlib.pyplot as plt  
import seaborn as sns  
  
plt.figure(figsize=(14, 7))  
  
# Plot actual distribution
plt.subplot(1, 2, 1)
sns.countplot(y\_true)
plt.title('Actual Ratings Distribution')
plt.xlabel('Ratings')
plt.ylabel('Frequency')  
  
# Plot predicted distribution
plt.subplot(1, 2, 2)
sns.countplot(y\_pred\_classes)
plt.title('Predicted Ratings Distribution')
plt.xlabel('Ratings')
plt.ylabel('Frequency')  
  
plt.tight\_layout()
plt.show()



```
In [39]: plt.figure(figsize=(8, 6))
sns.histplot(errors, bins=len(np.unique(errors)), kde=False)
plt.title('Distribution of Prediction Errors')
plt.xlabel('Error (Actual Rating - Predicted Rating)')
plt.ylabel('Frequency')
plt.xticks(np.unique(errors)) # Ensure ticks match error values
plt.grid(True)
plt.show()
```



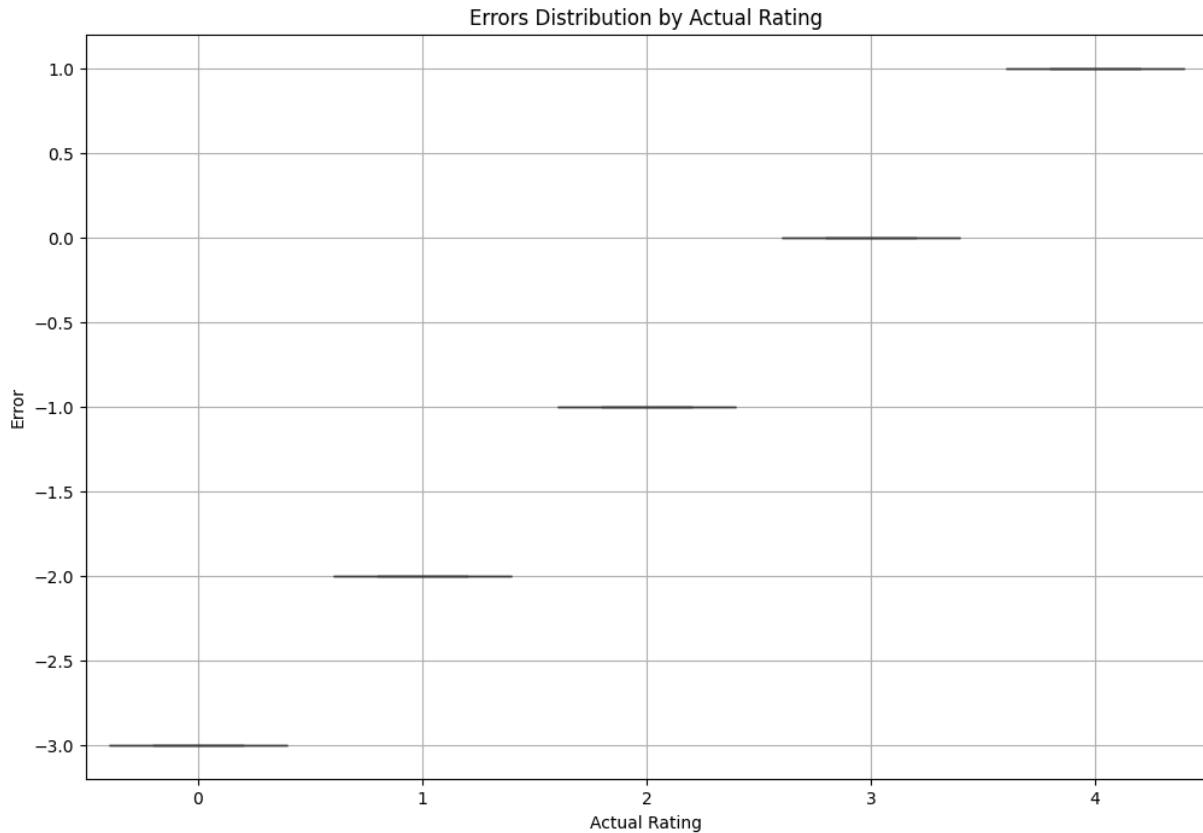
```
In [40]: from collections import Counter
```

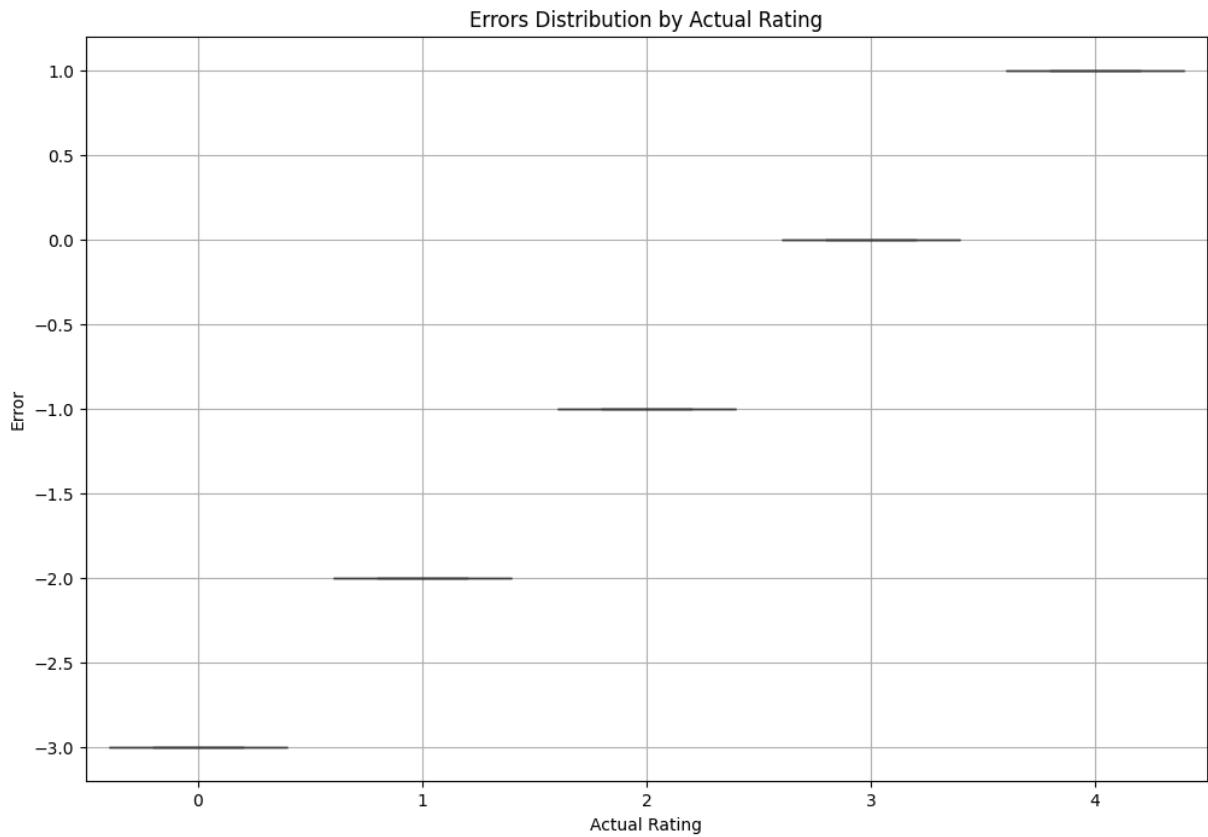
```
error_counts = Counter(errors)
print("Error Counts:", error_counts)
```

```
Error Counts: Counter({0: 1048, 1: 857, -1: 700, -2: 233, -3: 162})
```

```
In [41]: # Advanced visualization of errors by actual class
```

```
plt.figure(figsize=(12, 8))
sns.boxplot(x=y_true, y=errors)
plt.title('Errors Distribution by Actual Rating')
plt.xlabel('Actual Rating')
plt.ylabel('Error')
plt.grid(True)
plt.show()
# Advanced visualization of errors by actual class
plt.figure(figsize=(12, 8))
sns.boxplot(x=y_true, y=errors)
plt.title('Errors Distribution by Actual Rating')
plt.xlabel('Actual Rating')
plt.ylabel('Error')
plt.grid(True)
plt.show()
```





```
In [42]: from sklearn.inspection import permutation_importance

result = permutation_importance(model, X_train, y_train, n_repeats=10, random_state=42)
importances = result.importances_mean
```

```

-----  

  TypeError                                     Traceback (most recent call last)  

Cell In[42], line 3  

    1 from sklearn.inspection import permutation_importance  

----> 3 result = permutation_importance(model, X_train, y_train, n_repeats=1  

0, random_state=42)  

    4 importances = result.importances_mean  
  

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-  

packages/sklearn/utils/_param_validation.py:214, in validate_params.<locals>.  

>.decorator.<locals>.wrapper(*args, **kwargs)  

  208 try:  

  209     with config_context(  

  210         skip_parameter_validation=  

  211             prefer_skip_nested_validation or global_skip_validation  

  212         )  

  213     ):  

--> 214         return func(*args, **kwargs)  

  215 except InvalidParameterError as e:  

  216     # When the function is just a wrapper around an estimator, we al-  

low  

  217     # the function to delegate validation to the estimator, but we r-  

eplace  

  218     # the name of the estimator by the name of the function in the e-  

rror  

  219     # message to avoid confusion.  

  220     msg = re.sub(  

  221         r"parameter of \w+ must be",  

  222         f"parameter of {func.__qualname__} must be",  

  223         str(e),  

  224     )
  

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-  

packages/sklearn/inspection/_permutation_importance.py:282, in permutation_i-  

mportance(estimator, X, y, scoring, n_repeats, n_jobs, random_state, sample_-  

weight, max_samples)  

  280     scorer = scoring  

  281 elif scoring is None or isinstance(scoring, str):  

--> 282     scorer = check_scoring(estimator, scoring=scoring)  

  283 else:  

  284     scorers_dict = _check_multimetric_scoring(estimator, scoring)
  

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-  

packages/sklearn/utils/_param_validation.py:187, in validate_params.<locals>.  

>.decorator.<locals>.wrapper(*args, **kwargs)  

  185 global_skip_validation = get_config()["skip_parameter_validation"]  

  186 if global_skip_validation:  

--> 187     return func(*args, **kwargs)  

  188 func_sig = signature(func)  

  189 # Map *args/**kwargs to the function signature
  

File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-  

packages/sklearn/metrics/_scorer.py:953, in check_scoring(estimator, scorin-  

g, allow_none)  

  951     return None  

  952 else:

```

```
--> 953     raise TypeError(
 954         "If no scoring is specified, the estimator passed should "
 955         "have a 'score' method. The estimator %r does not." % estima
tor
 956     )
```

**TypeError**: If no scoring is specified, the estimator passed should have a 'score' method. The estimator <Sequential name=sequential\_3, built=True> does not.

In [ ]:

```
In [52]: # Assuming Small_df is your original DataFrame and you want to exclude 'overall_rating'
features = [col for col in Small_df.columns if col != 'overall_rating']
X = Small_df[numeric_columns]
y = Small_df['overall_rating']

# Now split the data, ensuring X remains a DataFrame
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Since X_train is a DataFrame, you can safely access columns
feature_names = X_train.columns.tolist()
```

In [53]: from sklearn.ensemble import RandomForestClassifier

```
# Assume X_train and y_train are your training features and labels
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Get feature importances
importances = model.feature_importances_
```

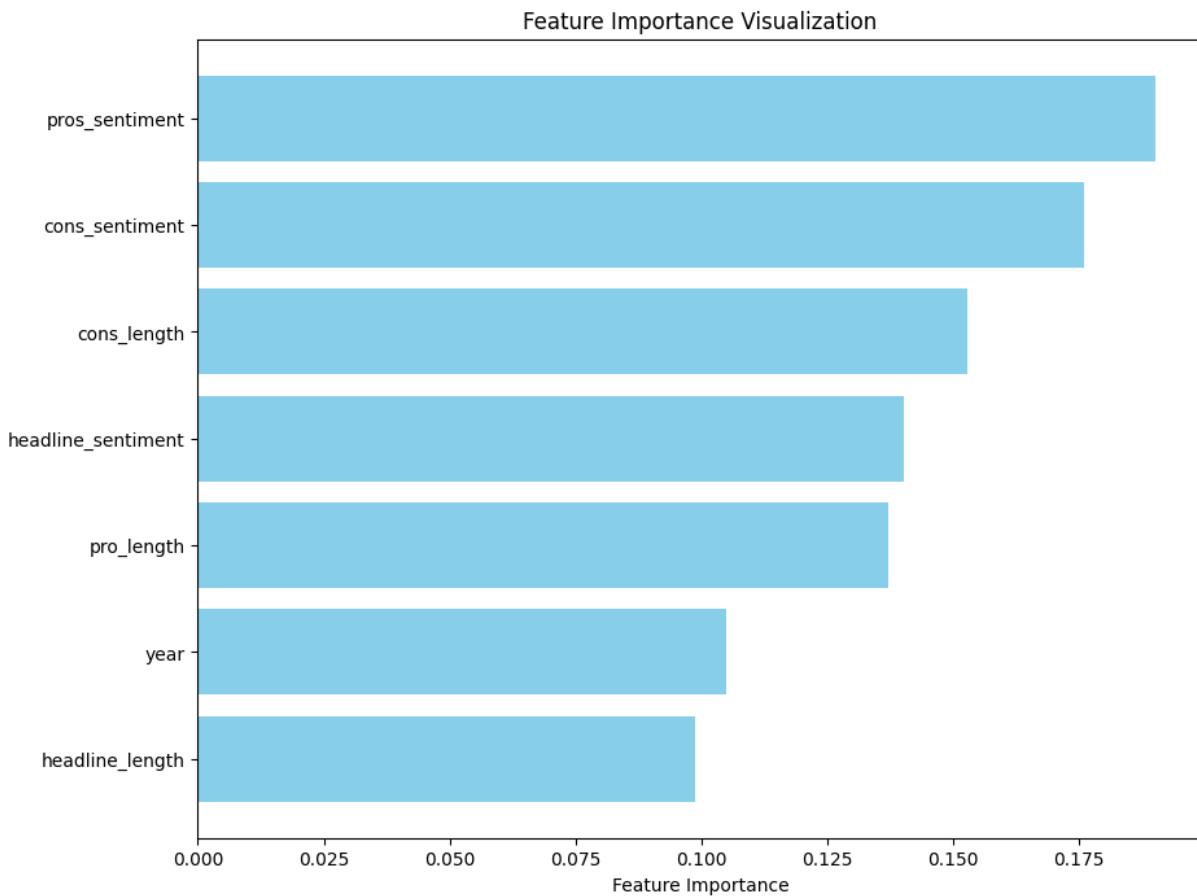
In [54]: importances

```
Out[54]: array([0.10481717, 0.09885811, 0.13723857, 0.15278767, 0.14014067,
 0.19005602, 0.17610177])
```

```
In [55]: import matplotlib.pyplot as plt
import numpy as np

# Sorting the features by importance
indices = np.argsort(importances)
sorted_feature_names = [feature_names[i] for i in indices]
sorted_importances = importances[indices]

# Creating the horizontal bar plot
plt.figure(figsize=(10, 8))
plt.barh(sorted_feature_names, sorted_importances, color='skyblue')
plt.xlabel('Feature Importance')
plt.title('Feature Importance Visualization')
plt.show()
```



## LSTM

```
In [34]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from sklearn.metrics import accuracy_score
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import LabelEncoder

# Tokenize the text data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(Small_df['headline'] + ' ' + Small_df['pros'] + ' ' +
word_index = tokenizer.word_index

# Convert text data to sequences
headline_seq = tokenizer.texts_to_sequences(Small_df['headline'])
pros_seq = tokenizer.texts_to_sequences(Small_df['pros'])
cons_seq = tokenizer.texts_to_sequences(Small_df['cons'])

# Pad sequences
max_sequence_length = 100 # Choose an appropriate sequence length
headline_seq_padded = pad_sequences(headline_seq, maxlen=max_sequence_length)
pros_seq_padded = pad_sequences(pros_seq, maxlen=max_sequence_length, padding='post')
cons_seq_padded = pad_sequences(cons_seq, maxlen=max_sequence_length, padding='post')

# Combine all sequences and sentiment scores into a single input array
X = np.concatenate((headline_seq_padded, pros_seq_padded, cons_seq_padded, S
```

```
# Encode the target labels
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(Small_df['overall_rating'])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y)

# Define the LSTM model
embedding_dim = 100 # Dimension of word embeddings
vocab_size = len(word_index) + 1 # Vocabulary size

model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim))
model.add(LSTM(units=128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(units=5, activation='softmax')) # Output layer with softmax

# Compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_steps=10)

# Make predictions on the test set
y_pred_prob = model.predict(X_test) # Predict class probabilities
y_pred = np.argmax(y_pred_prob, axis=1) # Convert probabilities to class labels

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

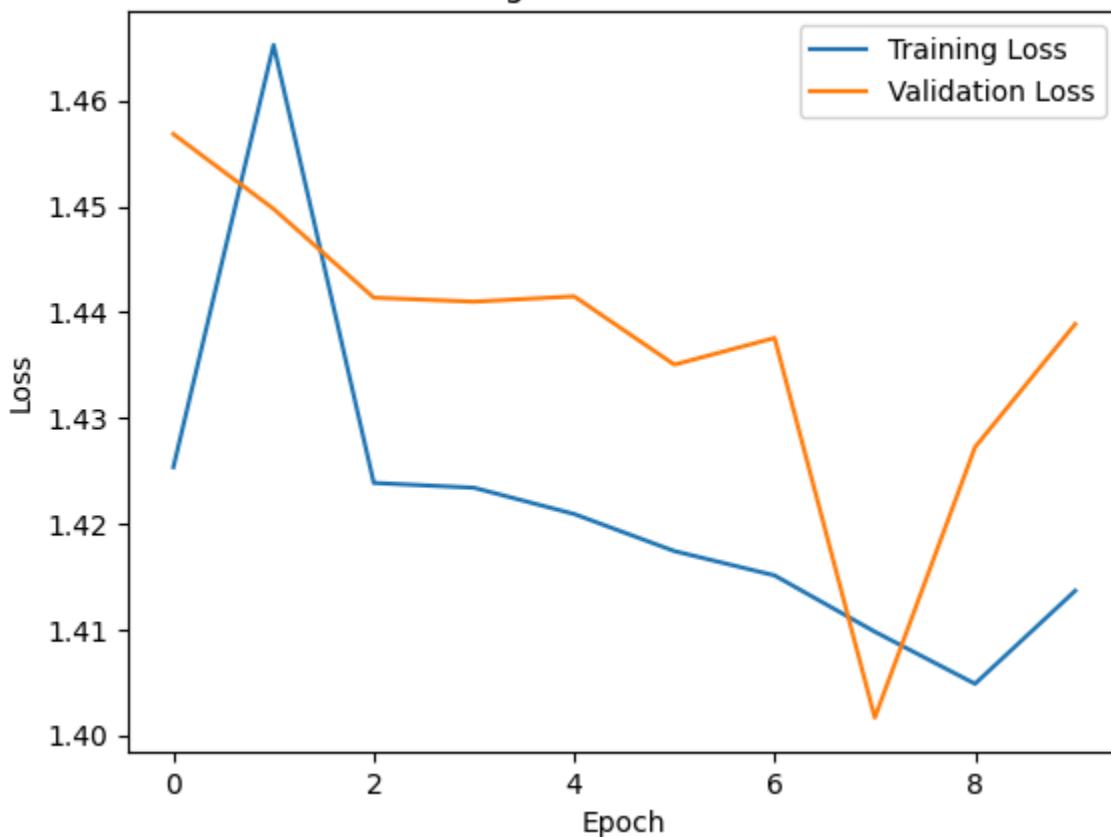
```
Epoch 1/10
197/197 47s 232ms/step - accuracy: 0.3263 - loss: 1.442
4 - val_accuracy: 0.2686 - val_loss: 1.4568
Epoch 2/10
197/197 56s 285ms/step - accuracy: 0.3211 - loss: 1.507
2 - val_accuracy: 0.3371 - val_loss: 1.4498
Epoch 3/10
197/197 57s 290ms/step - accuracy: 0.3528 - loss: 1.413
4 - val_accuracy: 0.3371 - val_loss: 1.4414
Epoch 4/10
197/197 50s 256ms/step - accuracy: 0.3287 - loss: 1.426
2 - val_accuracy: 0.3371 - val_loss: 1.4410
Epoch 5/10
197/197 56s 284ms/step - accuracy: 0.3484 - loss: 1.434
5 - val_accuracy: 0.3371 - val_loss: 1.4415
Epoch 6/10
197/197 50s 255ms/step - accuracy: 0.3557 - loss: 1.416
8 - val_accuracy: 0.3371 - val_loss: 1.4350
Epoch 7/10
197/197 56s 282ms/step - accuracy: 0.3335 - loss: 1.423
6 - val_accuracy: 0.3414 - val_loss: 1.4376
Epoch 8/10
197/197 64s 324ms/step - accuracy: 0.3564 - loss: 1.402
3 - val_accuracy: 0.3471 - val_loss: 1.4017
Epoch 9/10
197/197 51s 258ms/step - accuracy: 0.3572 - loss: 1.401
2 - val_accuracy: 0.3400 - val_loss: 1.4272
Epoch 10/10
197/197 60s 305ms/step - accuracy: 0.3435 - loss: 1.407
7 - val_accuracy: 0.3357 - val_loss: 1.4389
94/94 7s 75ms/step
Accuracy: 0.3483333333333333
```

```
In [35]: import matplotlib.pyplot as plt

# Get training and validation loss values from history
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Plot the loss curves
plt.plot(train_loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

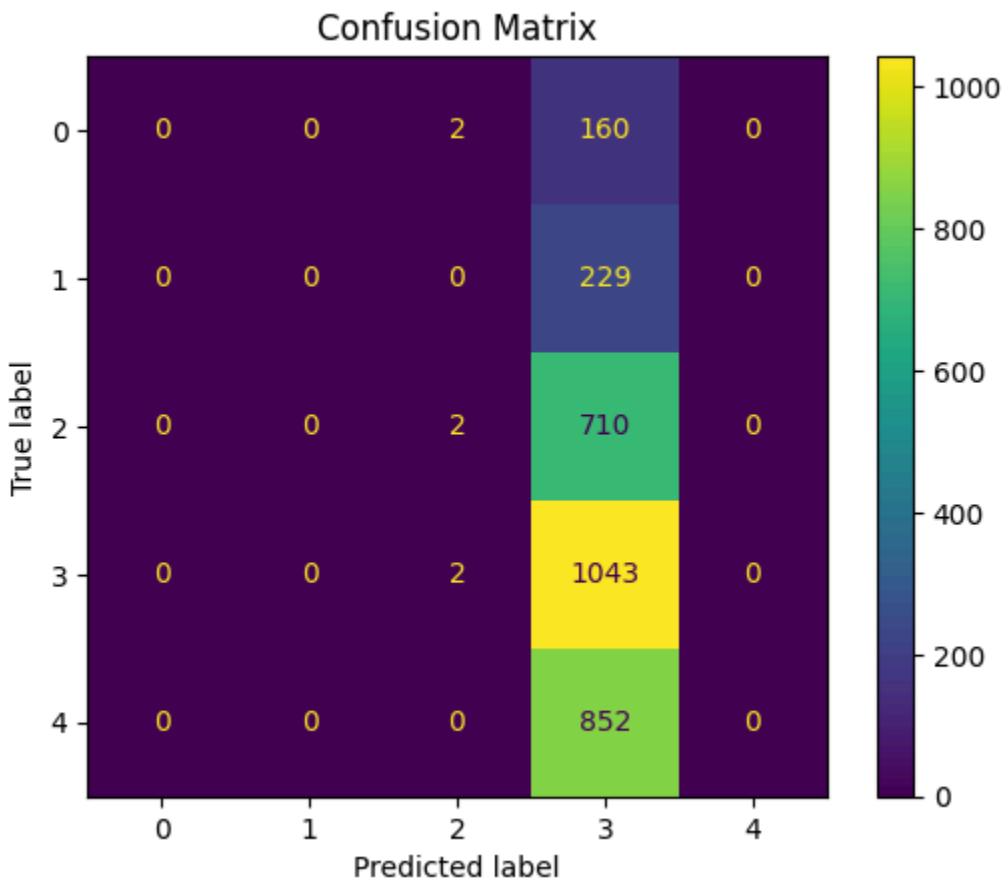
## Training and Validation Loss



```
In [36]: from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=)
disp.plot()
plt.title('Confusion Matrix')
plt.show()
```



```
In [61]: Test_df = pd.read_csv(Test_data_path)
```

```
In [63]: Test_df = Test_df.drop(['overall_rating'], axis=1)
```

```
In [65]: def test_data_cleaning(input_df):
    # Convert 'date_review' column to datetime
    input_df['date_review'] = pd.to_datetime(input_df['date_review'])

    # Fill missing values in object type columns with 'NA'
    object_columns = input_df.select_dtypes(include='object').columns
    input_df[object_columns] = input_df[object_columns].fillna('NA')

    if 'overall_rating' in input_df.columns:
        # Impute missing values in numerical columns with mean
        numerical_columns = input_df.select_dtypes(include=[int, float]).columns
        imputer = SimpleImputer(strategy='mean')
        input_df[numerical_columns] = imputer.fit_transform(input_df[numerical_columns])
    else:
        pass

    # Reset index
    input_df = input_df.reset_index(drop=True)

    # Convert certain columns to string type and fill missing values with empty string
    for col in ['headline', 'pros', 'cons']:
        input_df[col] = input_df[col].astype(str).fillna('')

    # Calculate the length of each text entry
    input_df['length'] = input_df['headline'].str.len()
```

```

input_df['headline_length'] = input_df['headline'].apply(lambda x: len(x))
input_df['pro_length'] = input_df['pros'].apply(lambda x: len(x.split()))
input_df['cons_length'] = input_df['cons'].apply(lambda x: len(x.split()))

# Calculate sentiment scores for 'headline', 'pros', and 'cons'
# Assuming sentiment_scores is a function you have defined elsewhere
input_df['headline_sentiment'] = input_df['headline'].apply(sentiment_scores)
input_df['pros_sentiment'] = input_df['pros'].apply(sentiment_scores)
input_df['cons_sentiment'] = input_df['cons'].apply(sentiment_scores)

return input_df

```

In [66]: # Example of preparing Test\_df (ensure this matches exactly what was done to  
Test\_df = test\_data\_cleaning(Test\_df) # Use the same features as used in tr  
# Apply any necessary preprocessing like scaling, encoding, etc.

In [67]: Test\_df\_prepared = Test\_df[numeric\_columns]

In [69]: # Making predictions  
y\_pred = model.predict(Test\_df\_prepared)

In [72]: y\_pred

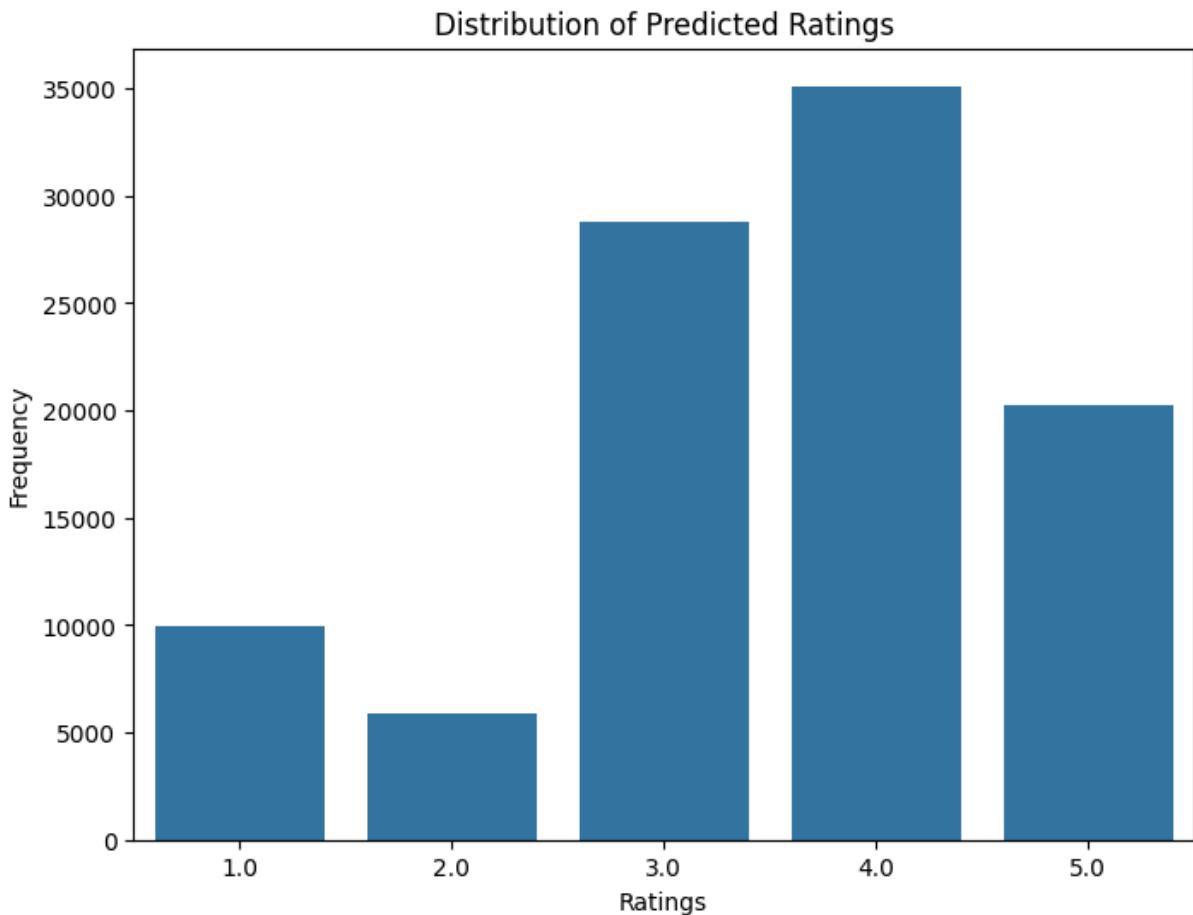
Out[72]: array([4., 4., 2., ..., 3., 5., 5.])

In [70]: print(y\_pred.shape)  
(100000,)

In [73]: # Assuming Test\_df is your test dataset  
results\_df = Test\_df.copy()  
results\_df['Predicted\_Rating'] = y\_pred # Directly use y\_pred as it already  
  
# Display the first few rows to verify  
print(results\_df[['Predicted\_Rating']].head())

	Predicted_Rating
0	4.0
1	4.0
2	2.0
3	4.0
4	5.0

In [74]: import matplotlib.pyplot as plt  
import seaborn as sns  
  
# Plotting the distribution of predicted ratings  
plt.figure(figsize=(8, 6))  
sns.countplot(x='Predicted\_Rating', data=results\_df)  
plt.title('Distribution of Predicted Ratings')  
plt.xlabel('Ratings')  
plt.ylabel('Frequency')  
plt.show()



```
In [75]: results_df.to_csv("Predicted_Ratings.csv", index=False)
```

## VOID

### Train Test Split

```
In [ ]: class ReviewsDataset(Dataset):
    def __init__(self, data, args):
        self.data = data
        self.args = args
    def __len__(self):
        return len(self.data)
    def __getitem__(self, idx):
        item = {
            'rate' : int(self.data.iloc[idx]['overall_rating'])-1, #so that it's 0-4
            'headline' : torch.as_tensor(self.data.iloc[idx]['tokenized_headline']),
            'pros' : torch.as_tensor(self.data.iloc[idx]['tokenized_pros']),
            'cons' : torch.as_tensor(self.data.iloc[idx]['tokenized_cons'])
        }
        return item
```

```
In [ ]: data_train, data_test_val = train_test_split(Small_df, test_size=0.8)
data_test, data_val = train_test_split(data_test_val, test_size=0.5)
```

```

del data_test_val

In [ ]: args = {'bsize': 16}

train_loader = DataLoader(ReviewsDataset(data_train, args), batch_size=args['bsize'])
del data_train
val_loader = DataLoader(ReviewsDataset(data_val, args), batch_size=args['bsize'])
del data_val
test_loader = DataLoader(ReviewsDataset(data_test, args), batch_size=args['bsize'])
del data_test

```

```

In [ ]: import torch
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split

# Define a custom dataset class
class ReviewsDataset(Dataset):
    def __init__(self, data):
        self.data = data

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        # Get the item at the specified index
        item = {
            'rate': int(self.data.iloc[idx]['overall_rating']) - 1, # Convert rating to 0-based index
            'headline': torch.as_tensor(self.data.iloc[idx]['tokenized_headline']),
            'pros': torch.as_tensor(self.data.iloc[idx]['tokenized_pros']),
            'cons': torch.as_tensor(self.data.iloc[idx]['tokenized_cons'])
        }
        return item

# Split the data into training, validation, and test sets
data_train, data_test_val = train_test_split(Small_df, test_size=0.8)
data_test, data_val = train_test_split(data_test_val, test_size=0.5)
del data_test_val

# Create data loaders for training, validation, and test sets
train_loader = DataLoader(ReviewsDataset(data_train), batch_size=16, shuffle=True)
del data_train

val_loader = DataLoader(ReviewsDataset(data_val), batch_size=16, shuffle=True)
del data_val

test_loader = DataLoader(ReviewsDataset(data_test), batch_size=16, shuffle=True)
del data_test

```

The ReviewsDataset class is a custom dataset class inheriting from PyTorch's Dataset class. It initializes with the data and implements the **len** and **getitem** methods. The **len** method returns the length of the dataset, and the **getitem** method retrieves an item at the specified index, converting ratings to a 0-based index.

The data is split into training, validation, and test sets using `train_test_split` function from scikit-learn.

Data loaders for training, validation, and test sets are created using PyTorch's `DataLoader` class, which loads data in batches and shuffles it if necessary. The `drop_last=True` argument ensures that the last incomplete batch is dropped if its size is less than the specified batch size.

```
In [ ]: def train(model, optimizer, ep, args):
    model.train()
    loss_it, acc_it = [], []
    for it, batch in tqdm(enumerate(train_loader), desc="Epoch %s:" % (ep+1)):
        # Move batch to device
        batch = {key: value.to(device) for key, value in batch.items()}
        optimizer.zero_grad()
        logits = model(batch['headline'], batch['pros'], batch['cons'])

        # Calculate class weights for unbalanced dataset
        b_counter = Counter(batch['rate'].detach().cpu().tolist())
        b_weights = torch.tensor([sum(batch['rate'].detach().cpu().tolist())])
        b_weights = b_weights.to(device)

        # Define loss function with class weights
        loss_function = nn.CrossEntropyLoss(weight=b_weights)
        loss = loss_function(logits, batch['rate'])

        loss.backward()
        optimizer.step()

        loss_it.append(loss.item())

        _, tag_seq = torch.max(logits, 1)
        correct = (tag_seq.flatten() == batch['rate'].flatten()).float().sum()
        acc = correct / batch['rate'].flatten().size(0)
        acc_it.append(acc.item())

        loss_it_avg = sum(loss_it) / len(loss_it)
        acc_it_avg = sum(acc_it) / len(acc_it)
        print("%s : (%s %s) (%s %s)" % (colored('Training', 'blue'), colored('loss_it_avg', 'red'), colored('loss_it', 'green'), colored('acc_it_avg', 'blue'), colored('acc_it', 'green')))
    return loss_it, loss_it_avg
```

This function trains the model for one epoch.

It sets the model to training mode (`model.train()`) and iterates over batches in the training data loader (`train_loader`).

Each batch is moved to the specified device (e.g., GPU) using `to(device)`.

The optimizer gradients are zeroed (`optimizer.zero_grad()`), and the model predictions (`logits`) are calculated.

Class weights are computed for the unbalanced dataset.

The loss is computed using nn.CrossEntropyLoss with the calculated class weights.

Backpropagation is performed (loss.backward()), followed by optimization step (optimizer.step()).

Training loss and accuracy for each iteration are computed and stored.

Finally, the average training loss and accuracy for the epoch are printed and returned.

```
In [ ]: def inference(target, loader, model):
    model.eval()
    loss_it, acc_it = [], []
    preds, trues = [], []

    for it, batch in enumerate(loader):
        with torch.no_grad():
            batch = {key: value.to(device) for key, value in batch.items()}
            logits = model(batch['headline'], batch['pros'], batch['cons'])

            b_counter = Counter(batch['rate'].detach().cpu().tolist())
            b_weights = torch.tensor([sum(b_counter[key]) for key in b_counter])
            b_weights = b_weights.to(device)

            loss_function = nn.CrossEntropyLoss(weight=b_weights)
            loss = loss_function(logits, batch['rate'])

            loss_it.append(loss.item())

            _, tag_seq = torch.max(logits, 1)
            correct = (tag_seq.flatten() == batch['rate'].flatten()).float()
            acc = correct / batch['rate'].flatten().size(0)
            acc_it.append(acc.item())

            preds.extend(tag_seq.cpu().detach().tolist())
            trues.extend(batch['rate'].cpu().detach().tolist())

            loss_it_avg = sum(loss_it) / len(loss_it)
            acc_it_avg = sum(acc_it) / len(acc_it)
            print("%s : (%s %s) (%s %s)" % (colored(target, 'green'), colored('loss', 'red'), loss_it_avg, colored('acc', 'blue'), acc_it_avg))

    return trues, preds, loss_it_avg, acc_it_avg, loss_it, acc_it
```

This function performs inference (evaluation) on the validation or test set.

It sets the model to evaluation mode (model.eval()) and iterates over batches in the specified data loader (loader). Each batch is moved to the specified device (to(device)).

The model predictions (logits) are computed.

Class weights are calculated for the unbalanced dataset.

The loss is computed using nn.CrossEntropyLoss with the calculated class weights.

Inference loss and accuracy for each iteration are computed and stored.

Predictions and true labels are collected for further analysis.

Finally, the average loss and accuracy for the inference set are printed and returned.

```
In [ ]: def run_epochs(model, args):
    args['device'] = device
    optimizer = optim.Adam(model.parameters(), lr=args['lr'])
    val_ep_losses = []
    val_losses = []
    train_ep_losses = []

    for ep in range(args['max_eps']):
        train_loss_it, train_loss_it_ep = train(model, optimizer, ep, args)
        trues, preds, val_loss_it_avg, val_acc_it_avg, val_loss_it, val_acc_
        val_ep_losses.append(val_loss_it_avg)
        train_ep_losses.append(train_loss_it_ep)
        val_losses = val_losses + val_loss_it

    return val_ep_losses, val_losses, train_ep_losses
```

This function executes the training loop over multiple epochs.

It initializes the optimizer (Adam optimizer in this case) with the provided learning rate (args['lr']).

Lists val\_ep\_losses, val\_losses, and train\_ep\_losses are initialized to store validation epoch losses, validation losses, and training epoch losses, respectively.

Inside the loop over epochs, it calls the train function to train the model for one epoch and stores the training losses.

Then, it calls the inference function to evaluate the model on the validation set, storing the validation losses and accuracies.

Finally, it returns the validation epoch losses (val\_ep\_losses), validation losses (val\_losses), and training epoch losses (train\_ep\_losses).

This function essentially orchestrates the training process, performing training and validation for each epoch and collecting relevant metrics for analysis.

```
In [ ]: class LSTMModel(torch.nn.Module):
    def __init__(self, input_dim, hidden_dim, hidden_dim2, output_dim, batch_
        super(LSTMModel, self).__init__()
        self.hidden_dim = hidden_dim
        self.batch_size = batch_size
        self.ebd = torch.nn.Embedding.from_pretrained(pretrained_vectors, fr_
        self.lstm_headline = torch.nn.LSTM(input_dim,hidden_dim, batch_first_
        self.lstm_pros = torch.nn.LSTM(input_dim,hidden_dim, 2, batch_first_
        self.lstm_cons = torch.nn.LSTM(input_dim,hidden_dim, 2, batch_first_
        self.softmax = torch.nn.Softmax(dim=1)
```

```

    self.relu = torch.nn.ReLU()
    self.hidden_headline = self.init_hidden() #hidden state of the lstm
    self.hidden_pros = self.init_hidden() #same for pros
    self.hidden_cons = self.init_hidden() #same for cons
    self.hidden_layer2 = torch.nn.Linear(hidden_dim*3, hidden_dim2)
    self.classification_layer = torch.nn.Linear(hidden_dim2, output_dim)
    self.dropout = nn.Dropout(0.25) #we add dropout to prevent overfitting

    def init_hidden(self):
        return (torch.autograd.Variable(torch.zeros(1, self.batch_size, self.hidden_dim)),
                torch.autograd.Variable(torch.zeros(1, self.batch_size, self.hidden_dim)))

    def forward(self, x_headline, x_pros, x_cons):
        h_headline = self.ebd(x_headline)
        h_pros = self.ebd(x_pros)
        h_cons = self.ebd(x_cons)
        h_headline, self.hidden_headline = self.lstm_headline(h_headline)
        h_pros, self.hidden_pros = self.lstm_pros(h_pros)
        h_cons, self.hidden_cons = self.lstm_cons(h_cons)
        h = torch.cat((h_headline[:, -1, :], h_pros[:, -1, :], h_cons[:, -1, :]), 1)
        h = self.relu(self.hidden_layer2(h))
        h = self.dropout(h)
        h = self.classification_layer(h)
        logits = self.softmax(h)
        return logits

```

## LSTM Model

```

In [ ]: class LSTMModel(torch.nn.Module):
    def __init__(self, input_dim, hidden_dim, hidden_dim2, output_dim, batch_size):
        super(LSTMModel, self).__init__()
        self.hidden_dim = hidden_dim
        self.batch_size = batch_size
        self.ebd = torch.nn.Embedding.from_pretrained(pretrained_vectors, freeze=True)
        self.lstm_headline = torch.nn.LSTM(input_dim, hidden_dim, batch_first=True)
        self.lstm_pros = torch.nn.LSTM(input_dim, hidden_dim, 2, batch_first=True)
        self.lstm_cons = torch.nn.LSTM(input_dim, hidden_dim, 2, batch_first=True)
        self.softmax = torch.nn.Softmax(dim=1)
        self.relu = torch.nn.ReLU()
        self.hidden_layer2 = torch.nn.Linear(hidden_dim * 3 + 3, hidden_dim2)
        self.classification_layer = torch.nn.Linear(hidden_dim2, output_dim)
        self.dropout = nn.Dropout(0.25)

    def init_hidden(self):
        return (torch.autograd.Variable(torch.zeros(1, self.batch_size, self.hidden_dim)),
                torch.autograd.Variable(torch.zeros(1, self.batch_size, self.hidden_dim)))

    def forward(self, x_headline, x_pros, x_cons, sentiment_headline, sentiment_pros, sentiment_cons):
        h_headline = self.ebd(x_headline)
        h_pros = self.ebd(x_pros)
        h_cons = self.ebd(x_cons)

        # LSTM layers for headline, pros, and cons
        h_headline, _ = self.lstm_headline(h_headline)

```

```

    h_pros, _ = self.lstm_pros(h_pros)
    h_cons, _ = self.lstm_cons(h_cons)

    # Extract the last hidden state from each LSTM layer
    h_headline = h_headline[:, -1, :]
    h_pros = h_pros[:, -1, :]
    h_cons = h_cons[:, -1, :]

    # Concatenate headline, pros, and cons features along with sentiment
    h = torch.cat((h_headline, h_pros, h_cons, sentiment_headline, senti

    # Pass through hidden layer 2
    h = self.relu(self.hidden_layer2(h))

    # Apply dropout
    h = self.dropout(h)

    # Final classification layer
    h = self.classification_layer(h)

    # Apply softmax activation
    logits = self.softmax(h)

    return logits

```

This code defines a PyTorch neural network model named LSTMModel.

In the constructor (`__init__`), the model architecture is defined:

Embedding layer (`self.ebd`) initialized with pretrained word vectors.

LSTM layers for headline, pros, and cons text data.

Activation functions (softmax and relu) and a dropout layer (dropout) to prevent overfitting.

Linear layers for hidden layer 2 (`hidden_layer2`) and the final classification layer (`classification_layer`).

The `init_hidden` method initializes the hidden states for the LSTM layers.

In the forward method, the input sequences for headline, pros, and cons are passed through the embedding layers and LSTM layers.

The outputs from the last time step of each LSTM layer are concatenated.

The concatenated output is passed through the hidden layer 2, followed by dropout.

Finally, the output is passed through the classification layer, and softmax activation is applied to obtain the logits.

```
In [ ]: if torch.cuda.is_available():
    device = 'cuda'
```

```

    print('DEVICE = ', colored(torch.cuda.get_device_name(0), "green" ) )
else:
    device = 'cpu'
    print('DEVICE = ', colored('CPU', "blue"))

```

```

In [ ]: pretrained_vectors = FastText(language='en')

# Define model parameters
batch_size = 16
input_dim = 30 # Dimension of fastText word embeddings
hidden_dim = 10 # Hidden dimension size for LSTM layers
hidden_dim2 = 5 # Hidden dimension size for the second linear layer
output_dim = 5 # Number of output classes (grades)

# Initialize the LSTM model
lstm_model = LSTMModel(input_dim, hidden_dim, hidden_dim2, output_dim, batch_size)

# Move the model to the specified device (e.g., GPU if available)
lstm_model.to(device)

```

```

In [ ]: args.update({'max_eps': 20, 'lr': 0.002, 'device': 'cuda', 'cuda': 0})
print('device', device)

loss_list_ep_val_lstm, loss_list_val_lstm, loss_list_ep_train_lstm = run_epochs()

```

```

In [ ]: plt.plot(range(1,args['max_eps']+1),loss_list_ep_val_lstm, color='blue', label='validation')
plt.plot(range(1,args['max_eps']+1), loss_list_ep_train_lstm, color='orange', label='training')
plt.legend(['validation','training'])
plt.title('Evolution of the loss over epochs for the LSTM model')
plt.show()

```

```

In [ ]: import sklearn
lstm_model.eval() #switch to eval mode
trues, preds, _, _, _ = inference('test', test_loader, lstm_model)
print('MSE :',sklearn.metrics.mean_squared_error(trues, preds))
print('MAE :',sklearn.metrics.mean_absolute_error(trues, preds))
print('Accuracy :',sklearn.metrics.accuracy_score(trues, preds))

```

```

In [ ]: sns.heatmap(sklearn.metrics.confusion_matrix(trues, preds, normalize='true'))
plt.show()

```

## UNVOID

```

In [ ]: import pandas as pd

# Assuming 'Small_df' is your DataFrame and it includes 'headline', 'pros', 'cons'
Small_df['combined_text'] = Small_df['headline'].fillna('') + ' ' + Small_df['pros'].fillna('')

```

```

In [ ]: import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

```

```
# Download necessary NLTK resources
nltk.download('punkt')
nltk.download('stopwords')

# Set stopwords
stop_words = set(stopwords.words('english'))

# Function to clean and tokenize text
def clean_tokenize(text):
    # Convert to lowercase
    tokens = word_tokenize(text.lower())
    # Remove stopwords and non-alphabetic tokens
    tokens = [word for word in tokens if word.isalpha() and word not in stop_words]
    return tokens

Small_df['tokens'] = Small_df['combined_text'].apply(clean_tokenize)
```

```
In [ ]: from collections import Counter

# Function to get most common words
def get_most_common_words(tokens_list, num_words=10):
    all_words = [word for tokens in tokens_list for word in tokens]
    word_freq = Counter(all_words)
    return word_freq.most_common(num_words)

# Group by 'overall_rating' and apply the function
common_words_by_rating = Small_df.groupby('overall_rating')['tokens'].agg(lambda x: get_most_common_words(x))

print(common_words_by_rating)
```

```
In [ ]: from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Generate a word cloud for each rating
for rating, freq in common_words_by_rating.items():
    wordcloud = WordCloud(width=800, height=400, background_color='white').generate_from_frequencies(freq)
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(f'Word Cloud for {rating}-Star Reviews')
    plt.axis("off")
    plt.show()
```

```
In [76]: predictions1 = pd.read_csv('/Users/andrew/Downloads/UW courses/ECON 626/Final Predictions.csv')
```

```
In [84]: np.median(predictions1)
```

```
Out[84]: 4.0
```

```
In [85]: len(predictions1)
```

```
Out[85]: 84253
```

```
In [86]: predictions1.loc[84254:100000, 'predictions1'] = 4
```

```
In [100... additional_rows = 100000 - len(predictions1)
new_df = pd.DataFrame({'Predicted_Rating': [4]*additional_rows}) # Assuming
df = pd.concat([predictions1, new_df], ignore_index=True)
```

```
In [101... len(df)
```

```
Out[101... 100000
```

```
In [102... df.head()
```

```
Out[102... Predicted_Rating predictions1
```

0	4	NaN
1	4	NaN
2	5	NaN
3	2	NaN
4	3	NaN

```
In [103... df.tail()
```

```
Out[103... Predicted_Rating predictions1
```

99995	4	NaN
99996	4	NaN
99997	4	NaN
99998	4	NaN
99999	4	NaN

```
In [105... df['Predicted_Rating'].to_csv('predictions1.csv', index=False)
```

```
In [ ]:
```