Name
TheDebugger

Question 1)

MSE: 0.8214836108632233
R^2: 0.41250268307820426

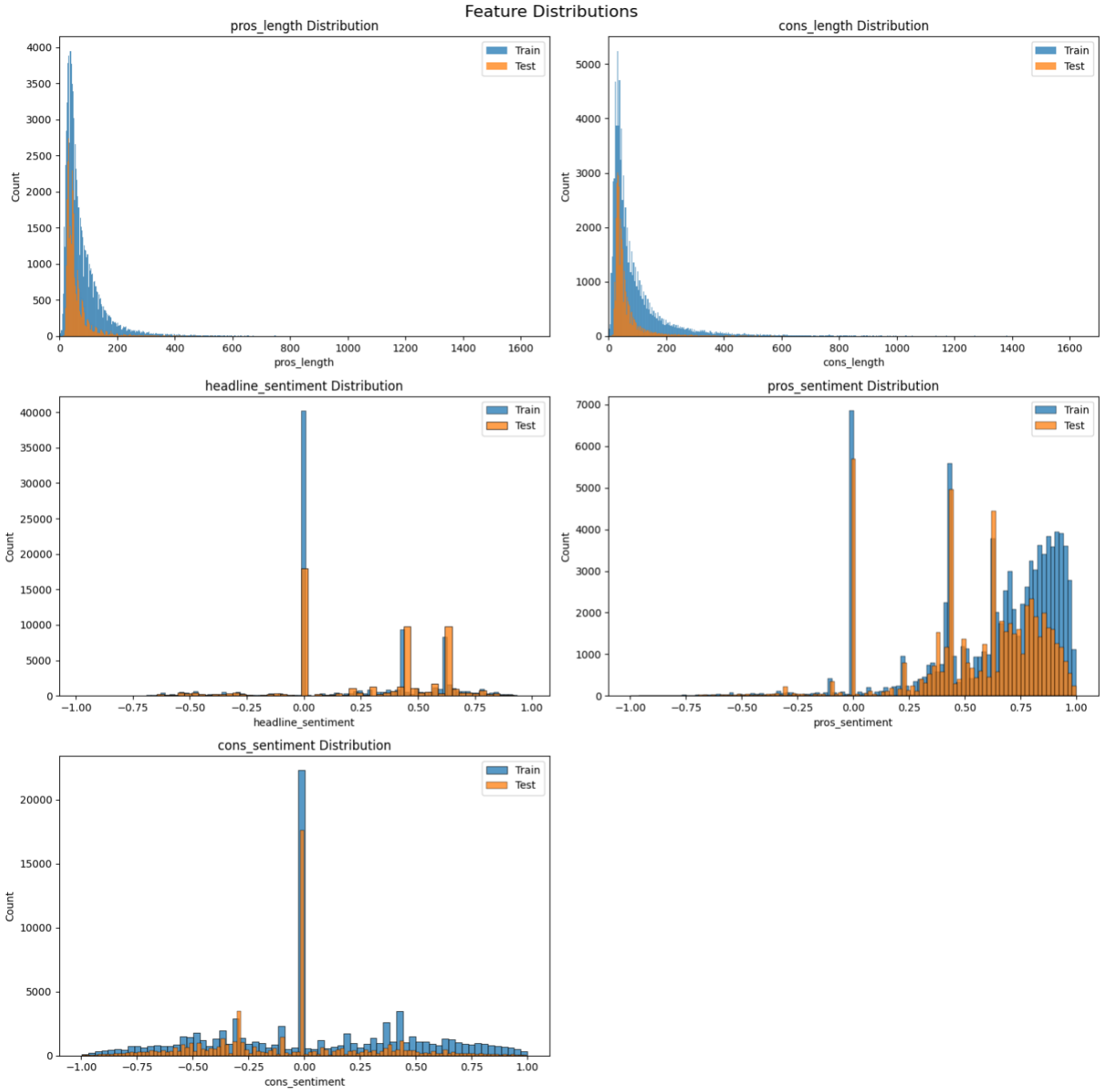
Neural Network:
Used an MLP (Multi-layer perceptron) neural network model. It has 1 hidden layer with 100 neurons. It uses a logistic sigmoid function for activation for more complex predictions as described in class.

It has 5 input features and 100 nodes/neurons in the hidden layer, so 5x100=500 weights, and 100 weights connecting to the single output of overall rating. There is also a bias for each node in the hidden layer and output layer, so a total bias of 101. Thus, a total of 600 weights and 101 biases, giving us a total number of inner parameters of 701.

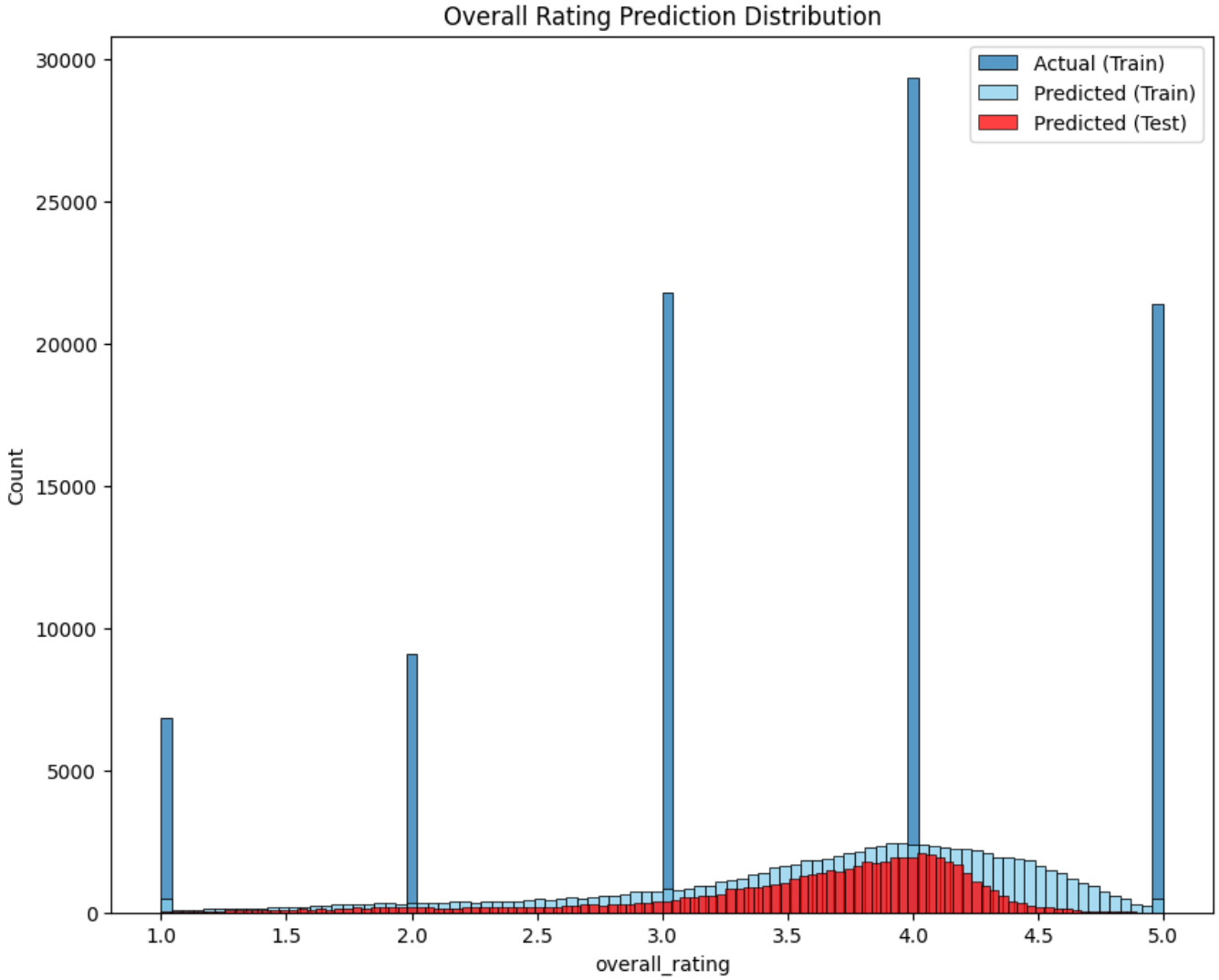
The 5 features used were: character length of cons, character length of pros, sentiment score of cons, sentiment score of pros, and sentiment score of headline. Everything else was disregarded.

Question 2)

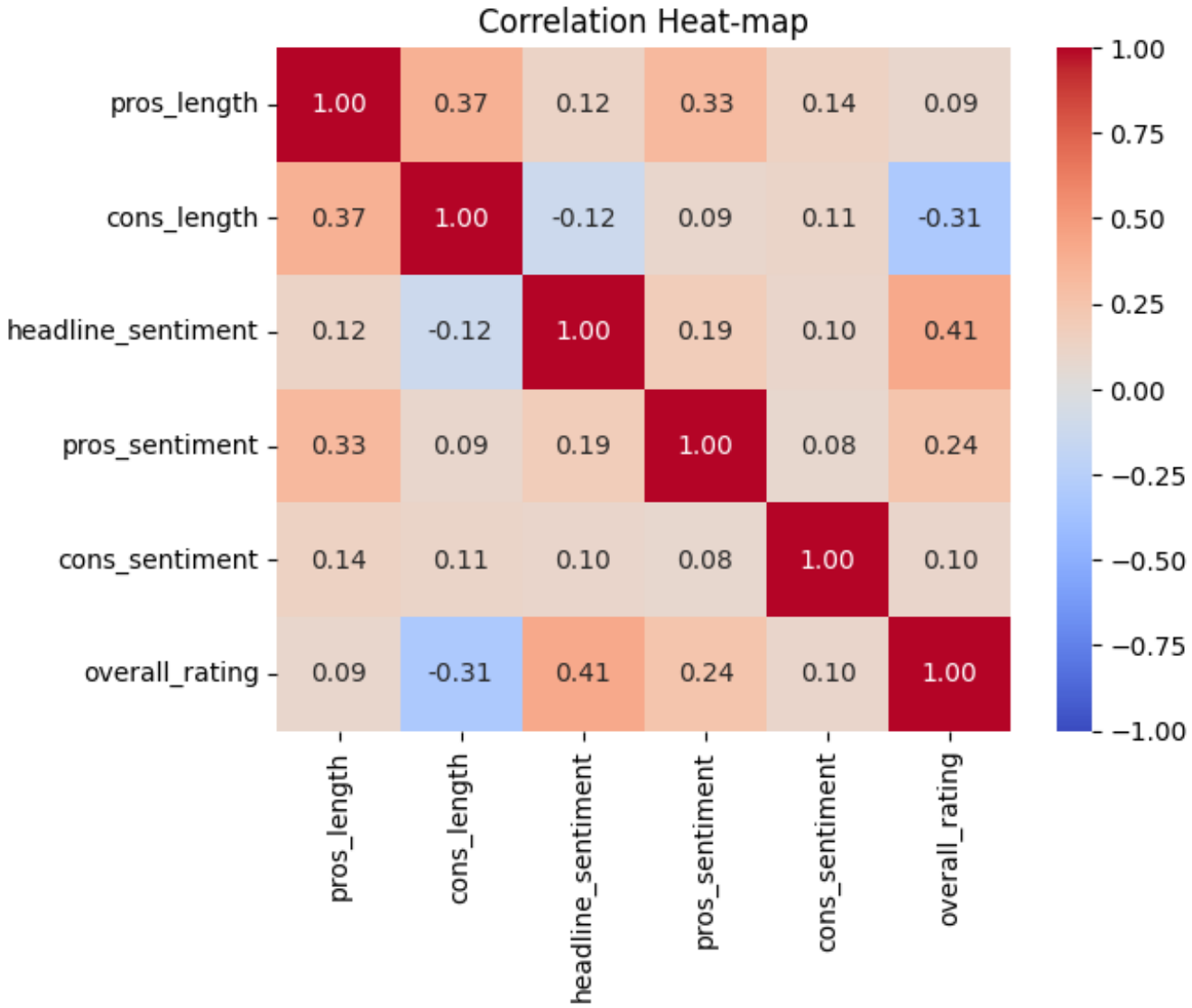
Feature Distribution Graphs in Test/Train:



Prediction/Value Distribution in Test/Train:



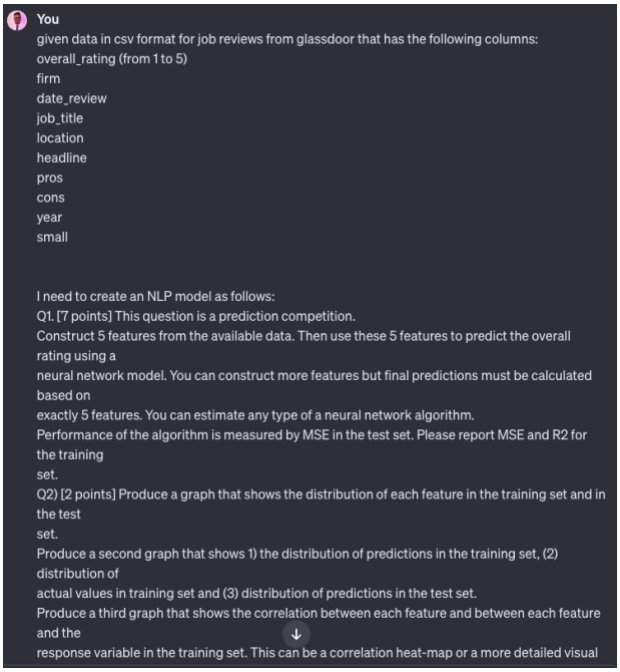
Heat Map:


Correlation Heat-map
pros_length
cons_length
headline_sentiment
pros_sentiment
cons_sentiment
overall_rating
1.00 0.37 0.12 0.33 0.14 0.09
0.37 1.00 -0.12 0.09 0.11 -0.31
0.12 -0.12 1.00 0.19 0.10 0.41
0.33 0.09 0.19 1.00 0.08 0.24
0.14 0.11 0.10 0.08 1.00 0.10
0.09 -0.31 0.41 0.24 0.10 1.00
pros_length
cons_length
headline_sentiment
pros_sentiment
cons_sentiment
overall_rating
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

Question 3)

Interactions with ChatGPT allowed me to create one well documented and working piece of code to train the neural network as required. It also allowed me to fix a bunch of bugs, understand how to calculate the parameters in the model and more.

Question:

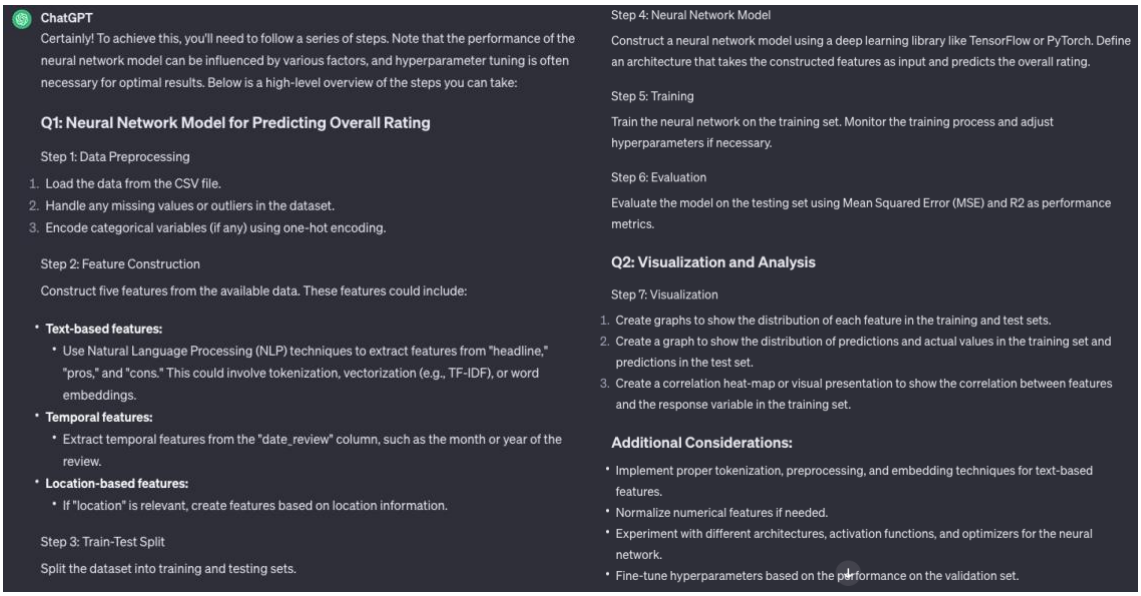


You

given data in csv format for job reviews from glassdoor that has the following columns:
overall_rating (from 1 to 5)
firm
date_review
job_title
location
headline
pros
cons
year
small

I need to create an NLP model as follows:
Q1. [7 points] This question is a prediction competition.
Construct 5 features from the available data. Then use these 5 features to predict the overall rating using a
neural network model. You can construct more features but final predictions must be calculated based on
exactly 5 features. You can estimate any type of a neural network algorithm.
Performance of the algorithm is measured by MSE in the test set. Please report MSE and R2 for the training
set.
Q2) [2 points] Produce a graph that shows the distribution of each feature in the training set and in the test
set.
Produce a second graph that shows 1) the distribution of predictions in the training set, (2) distribution of
actual values in training set and (3) distribution of predictions in the test set.
Produce a third graph that shows the correlation between each feature and between each feature and the
response variable in the training set. This can be a correlation heat-map or a more detailed visual

Answer:



ChatGPT

Certainly! To achieve this, you'll need to follow a series of steps. Note that the performance of the neural network model can be influenced by various factors, and hyperparameter tuning is often necessary for optimal results. Below is a high-level overview of the steps you can take:

Q1: Neural Network Model for Predicting Overall Rating

Step 1: Data Preprocessing

1. Load the data from the CSV file.
2. Handle any missing values or outliers in the dataset.
3. Encode categorical variables (if any) using one-hot encoding.

Step 2: Feature Construction

Construct five features from the available data. These features could include:

- **Text-based features:**
  - Use Natural Language Processing (NLP) techniques to extract features from "headline," "pros," and "cons." This could involve tokenization, vectorization (e.g., TF-IDF), or word embeddings.
- **Temporal features:**
  - Extract temporal features from the "date_review" column, such as the month or year of the review.
- **Location-based features:**
  - If "location" is relevant, create features based on location information.

Step 3: Train-Test Split

Split the dataset into training and testing sets.

Step 4: Neural Network Model

Construct a neural network model using a deep learning library like TensorFlow or PyTorch. Define an architecture that takes the constructed features as input and predicts the overall rating.

Step 5: Training

Train the neural network on the training set. Monitor the training process and adjust hyperparameters if necessary.

Step 6: Evaluation

Evaluate the model on the testing set using Mean Squared Error (MSE) and R2 as performance metrics.

Q2: Visualization and Analysis

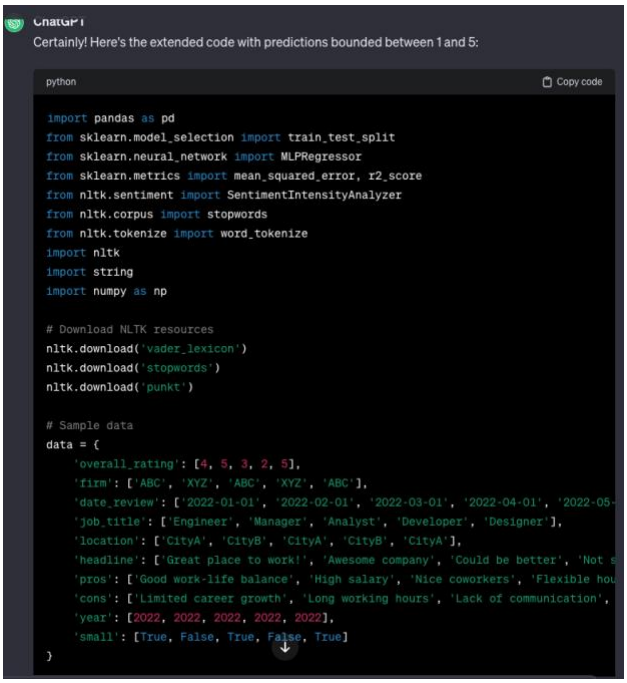
Step 7: Visualization

1. Create graphs to show the distribution of each feature in the training and test sets.
2. Create a graph to show the distribution of predictions and actual values in the training set and predictions in the test set.
3. Create a correlation heat-map or visual presentation to show the correlation between features and the response variable in the training set.

Additional Considerations:

- Implement proper tokenization, preprocessing, and embedding techniques for text-based features.
- Normalize numerical features if needed.
- Experiment with different architectures, activation functions, and optimizers for the neural network.
- Fine-tune hyperparameters based on the performance on the validation set.

After several modifications and changes requested, ChatGPT gave the following code which required very few modifications and I used.



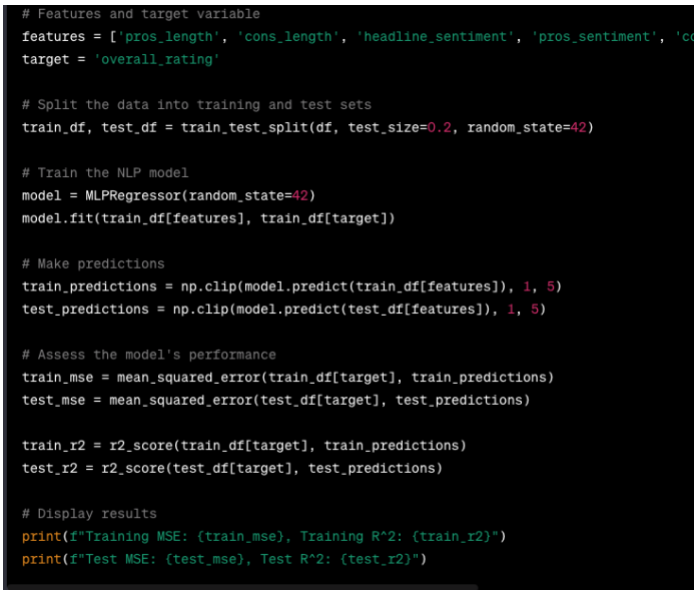
ChatGPT

Certainly! Here's the extended code with predictions bounded between 1 and 5:

python — Copy code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score
from nltk.sentiment import SentimentIntensityAnalyzer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
import string
import numpy as np

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

# Sample data
data = {
    'overall_rating': [4, 5, 3, 2, 5],
    'firm': ['ABC', 'XYZ', 'ABC', 'XYZ', 'ABC'],
    'date_review': ['2022-01-01', '2022-02-01', '2022-03-01', '2022-04-01', '2022-05-
    'job_title': ['Engineer', 'Manager', 'Analyst', 'Developer', 'Designer'],
    'location': ['CityA', 'CityB', 'CityA', 'CityB', 'CityA'],
    'headline': ['Great place to work!', 'Awesome company', 'Could be better', 'Not s
    'pros': ['Good work-life balance', 'High salary', 'Nice coworkers', 'Flexible hou
    'cons': ['Limited career growth', 'Long working hours', 'Lack of communication',
    'year': [2022, 2022, 2022, 2022, 2022],
    'small': [True, False, True, False, True]
}
```

```
# Create a DataFrame
df = pd.DataFrame(data)

# Text preprocessing function
def preprocess_text(text):
    # Tokenize
    tokens = word_tokenize(text)

    # Remove punctuation
    tokens = [word for word in tokens if word.isalpha()]

    # Remove stop words
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word.lower() not in stop_words]

    return ' '.join(tokens)

# Apply text preprocessing to 'pros' and 'cons' columns
df['pros'] = df['pros'].apply(preprocess_text)
df['cons'] = df['cons'].apply(preprocess_text)

# Feature construction
df['pros_length'] = df['pros'].apply(len)
df['cons_length'] = df['cons'].apply(len)
df['headline_sentiment'] = df['headline'].apply(lambda x: SentimentIntensityAnalyzer(
df['pros_sentiment'] = df['pros'].apply(lambda x: SentimentIntensityAnalyzer().polari
df['cons_sentiment'] = df['cons'].apply(lambda x: SentimentIntensityAnalyzer().polari

# Features and target variable
features = ['pros_length', 'cons_length', 'headline_sentiment', 'pros_sentiment', 'co
target = 'overall_rating'

# Split the data into training and test sets
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
```



```
# Features and target variable
features = ['pros_length', 'cons_length', 'headline_sentiment', 'pros_sentiment', 'co
target = 'overall_rating'

# Split the data into training and test sets
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)

# Train the NLP model
model = MLPRegressor(random_state=42)
model.fit(train_df[features], train_df[target])

# Make predictions
train_predictions = np.clip(model.predict(train_df[features]), 1, 5)
test_predictions = np.clip(model.predict(test_df[features]), 1, 5)

# Assess the model's performance
train_mse = mean_squared_error(train_df[target], train_predictions)
test_mse = mean_squared_error(test_df[target], test_predictions)

train_r2 = r2_score(train_df[target], train_predictions)
test_r2 = r2_score(test_df[target], test_predictions)

# Display results
print(f"Training MSE: {train_mse}, Training R^2: {train_r2}")
print(f"Test MSE: {test_mse}, Test R^2: {test_r2}")
```

Code Section Starts Here

```
################################################################
# Imports
#### Visualization packages
import matplotlib.pyplot as plt
import seaborn as sns

####  Import required model libraries
import numpy as np, pandas as pd
from matplotlib.pyplot import subplots
from sklearn.model_selection import (train_test_split , GridSearchCV)
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score

####  Import sentiment library
import pandas as pd
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import nltk
nltk.download('all')
import math

################################################################
# Read in Data
data = pd.read_csv("./Econ424_F2023_PC6_glassdoor_training_small_v1.csv")
print(data.head())

data.drop(['small'], axis="columns",inplace=True)


################################################################
# Preprocess Data

# create preprocess_text function
def preprocess_text(text):

    # Tokenize the text
    tokens = word_tokenize(text.lower())

    # Remove stop words
    filtered_tokens = [token for token in tokens if token not in stopwords.words('english')]

    # Lemmatize the tokens
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in filtered_tokens]

    # Join the tokens back into a string
    processed_text = ' '.join(lemmatized_tokens)
    return processed_text

# apply the function df
data['pros'] = data['pros'].apply(preprocess_text)
data['cons'] = data['cons'].apply(preprocess_text)
# output to csv file
csv_file_out = "./preprocessed.csv"

# Save the DataFrame to a CSV file
data.to_csv(csv_file_out,index=False, encoding="utf-8", float_format="%1.6f")


################################################################
# Construct Features from processed data
df = pd.read_csv("./preprocessed.csv", lineterminator='\n')
print(df.head())


# Specify the columns you want to check for missing values
columns_to_check = ['pros', 'cons', 'headline']

# Check for missing values in the specified columns
df = df.dropna(subset=columns_to_check)
missing_values = df[columns_to_check].isna()
rows_with_missing_values = df[missing_values.any(axis=1)]
print(len(rows_with_missing_values))

df['pros'] = df['pros'].astype(str)
df['cons'] = df['cons'].astype(str)

# Feature construction
df['pros_length'] = df['pros'].apply(len)
df['cons_length'] = df['cons'].apply(len)
df['headline_sentiment'] = df['headline'].apply(lambda x: SentimentIntensityAnalyzer().polarity_scores(str(x))['compound'])
df['pros_sentiment'] = df['pros'].apply(lambda x: SentimentIntensityAnalyzer().polarity_scores(str(x))['compound'])
df['cons_sentiment'] = df['cons'].apply(lambda x: SentimentIntensityAnalyzer().polarity_scores(str(x))['compound'])

# Check if any are null
print(df["headline_sentiment"].isna().values.any())
print(df["pros_sentiment"].isna().values.any())
print(df["cons_sentiment"].isna().values.any())
```

```
# output to csv file to save progress
csv_file_out = "./postsentiment.csv"
df.to_csv(csv_file_out,index=False, encoding="utf-8", float_format="%1.6f")



################################################################
# Train Model
df = pd.read_csv("./postsentiment.csv", lineterminator='\n')

# Features and target variable
features = ['pros_length', 'cons_length', 'headline_sentiment', 'pros_sentiment', 'cons_sentiment']
target = 'overall_rating'

df.drop(columns=["location"],inplace=True)

# Create training and test sets
X = df[['pros_length', 'cons_length', 'headline_sentiment', 'pros_sentiment', 'cons_sentiment']]
y = df['overall_rating']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Model Building with Sigmoid Neuron
model = MLPRegressor(hidden_layer_sizes=(100), activation='logistic', max_iter=500, solver='adam')
model.fit(X_train, y_train)

# Make predictions
y_train_pred = np.clip(model.predict(X_train),1, 5)
y_test_pred = np.clip(model.predict(X_test),1,5)

# Model Evaluation
mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
r2_train = r2_score(y_train, y_train_pred)

# Print MSE and R2 for the training set
print(f'MSE (Training Set): {mse_train}')
print(f'R2 Score (Training Set): {r2_train}')



################################################################
# Make Predictions for Submission
dataPred = pd.read_csv("./Econ424_F2023_PC6_glassdoor_test_without_response_variable_v1.csv")
print(dataPred.head())

dataPred.drop(['overall_rating','small','year'], errors='ignore',
  axis='columns', inplace=True)

# Specify the columns you want to check for missing values
columns_to_check = ['pros', 'cons', 'headline']

# Check for missing values in the specified columns
print(dataPred[columns_to_check].isna().any())

# Feature construction
dataPred['pros_length'] = dataPred['pros'].apply(len)
dataPred['cons_length'] = dataPred['cons'].apply(len)
dataPred['headline_sentiment'] = dataPred['headline'].apply(lambda x: SentimentIntensityAnalyzer().polarity_scores(str(x))['compound'])
dataPred['pros_sentiment'] = dataPred['pros'].apply(lambda x: SentimentIntensityAnalyzer().polarity_scores(str(x))['compound'])
dataPred['cons_sentiment'] = dataPred['cons'].apply(lambda x: SentimentIntensityAnalyzer().polarity_scores(str(x))['compound'])

dataPred = dataPred[["pros_length", 'cons_length','headline_sentiment','pros_sentiment','cons_sentiment']]
Y_test_pred = np.clip(model.predict(dataPred),1,5)



# output to csv file
csv_file_out = "./output.csv"

# Save the DataFrame to a CSV file
np.savetxt(csv_file_out, Y_test_pred, delimiter="\n", fmt="%1.6f")


print(len(Y_test_pred))
print(len(dataPred))



################################################################
# Graphs

### Consolidated prediction distribution graph
fig, axes = plt.subplots(figsize=(10, 8))
# Plot prediction distributions for actual and predicted values in training and test sets
sns.histplot(y_train, label='Actual (Train)', ax=axes, kde=False)
sns.histplot(y_train_pred, label='Predicted (Train)', ax=axes, kde=False, color="skyblue")
sns.histplot(Y_test_pred, label='Predicted (Test)', ax=axes, kde=False, color="red")
```

```
axes.set_title('Overall Rating Prediction Distribution')
axes.legend()

# Save the figure
plt.savefig('consolidated_prediction_distributions.png')
plt.show()




### Consolidated feature distribution graph
fig, axes = plt.subplots(3, 2, figsize=(15, 15))
fig.suptitle('Feature Distributions', fontsize=16)

# Plot feature distributions for training and test sets
for i, feature in enumerate(features):

    x = math.floor(i/2)
    y = i%(2)
    sns.histplot(X[feature], ax=axes[x, y],label='Train', kde=False)
    sns.histplot(dataPred[feature], ax=axes[x, y],label='Test', kde=False)
    axes[x, y].set_title(f'{feature} Distribution')
    axes[x, y].legend()
axes[0, 0].set_xlim(0, 1700)
axes[0, 1].set_xlim(0, 1700)
# Remove the empty subplot in the last row and second column
fig.delaxes(axes[2, 1])

# Adjust layout to prevent clipping of titles
fig.tight_layout()

# Save the figure
plt.savefig('consolidated_feature_distributions.png')
plt.show()




### Correlation heat-map
correlation_matrix = df[['pros_length', 'cons_length', 'headline_sentiment', 'pros_sentiment', 'cons_sentiment', 'overall_rating']].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", vmin=-1, vmax=1)
plt.title('Correlation Heat-map')
plt.show()
```