```
################################################################
# 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()
```