IR(Assignment-3)

Product chosen - Headphones

4.

```python
merged_df['overall'] = pd.to_numeric(merged_df['overall'], errors='coerce') # errors='coerce' parameter handles any non-numeric values by converting them to NaN (Not a Number).

# Calculate the average rating score
average_rating = merged_df['overall'].mean()

print(f"Average rating score: {average_rating:.2f}")

num_unique_products = merged_df['asin'].nunique()

print(f"Number of unique products: {num_unique_products}")


# Set a threshold for good and bad ratings
threshold = 3

# Create a new column 'rating_category' based on the threshold
merged_df['rating_category'] = merged_df['overall'].apply(lambda x: 'Good' if x >= threshold else 'Bad')

# Calculate the number of good ratings
num_good_ratings = merged_df[merged_df['rating_category'] == 'Good'].shape[0]

# Calculate the number of bad ratings
num_bad_ratings = merged_df[merged_df['rating_category'] == 'Bad'].shape[0]

# Group by 'rating_category' and count the number of reviews for each rating
ratings_count = merged_df.groupby('rating_category')['reviewText'].count().reset_index()

print(f"Number of Good Ratings: {num_good_ratings}")
print(f"Number of Bad Ratings: {num_bad_ratings}")
print("Number of Reviews corresponding to each Rating:")
print(ratings_count)
print(f"Total ratings: {num_good_ratings+num_bad_ratings}")
```

```
Average rating score: 4.12
Number of unique products: 8064
Number of Good Ratings: 375724
Number of Bad Ratings: 60717
Number of Reviews corresponding to each Rating:
  rating_category  reviewText
0             Bad       60715
1            Good      375654
Total ratings: 436441
```

5.

a)

Applies BeautifulSoup to remove HTML tags from the 'date' column of merged_df, similar to the 'reviewText' column. The lambda function checks if a value is a string before applying BeautifulSoup to avoid errors with non-string values.

```python
from bs4 import BeautifulSoup
import numpy as np  # for handling NaN values

# Function to remove HTML tags from text
def remove_html_tags(text):
    if isinstance(text, str):  # Check if the value is a string
        soup = BeautifulSoup(text, "html.parser")
        cleaned_text = soup.get_text()
        return cleaned_text
    else:
        return text  # Return the original value if not a string

# Apply the function to remove HTML tags from 'reviewText' column
merged_df['reviewText'] = merged_df['reviewText'].apply(remove_html_tags)

merged_df['date'] = merged_df['date'].apply(lambda x: BeautifulSoup(x, 'html.parser').get_text() if isinstance(x, str) else x)

# Print the first few rows to check the result
print(merged_df.head())
```

```
                                                   rank  \
tech1
NaN    ['>#316,475 in Cell Phones & Accessories (See ...
NaN    ['>#316,475 in Cell Phones & Accessories (See ...
NaN    ['>#316,475 in Cell Phones & Accessories (See ...
NaN    ['>#316,475 in Cell Phones & Accessories (See ...
NaN    ['>#316,475 in Cell Phones & Accessories (See ...

                                               also_view             main_cat  \
tech1
NaN    ['B019PO1ZRI', 'B00W87LKXE', 'B019PO1ZNC', 'B0...  Musical Instruments
NaN    ['B019PO1ZRI', 'B00W87LKXE', 'B019PO1ZNC', 'B0...  Musical Instruments
NaN    ['B019PO1ZRI', 'B00W87LKXE', 'B019PO1ZNC', 'B0...  Musical Instruments
NaN    ['B019PO1ZRI', 'B00W87LKXE', 'B019PO1ZNC', 'B0...  Musical Instruments
NaN    ['B019PO1ZRI', 'B00W87LKXE', 'B019PO1ZNC', 'B0...  Musical Instruments

       ... verified   reviewTime      reviewerID  \
tech1  ...
NaN    ...     True  02 22, 2015  A38RQFVQ1AKJQQ
NaN    ...     True   05 8, 2017  A299MRB906GWDE
NaN    ...     True   11 5, 2016  A3ACFC6DQQLIQT
NaN    ...     True  09 24, 2016  A36BC0YFDBNB5X
NaN    ...     True  07 17, 2016  A212PQ0HQPNNWM

                                style   reviewerName  \
```

b)

```python
import unicodedata

# Function to remove accented characters from text
def remove_accented_chars(text):
  if isinstance(text, str):  # Check if the value is a string
      normalized_text = unicodedata.normalize('NFKD', text) #This converts accented characters into their base form and separates them from their accents.
      return normalized_text.encode('ASCII', 'ignore').decode('utf-8') # We use .encode('ASCII', 'ignore') to remove any remaining non-ASCII characters after normalization and
  else:                                                                 # .decode('utf-8') to decode the bytes back to a UTF-8 string.
      return text  # Return the original value if not a string

# Apply the function to remove accented characters from 'reviewText' column
merged_df['reviewText'] = merged_df['reviewText'].apply(remove_accented_chars)

# Print the first few rows to check the result
print(merged_df.head())
```

```
                                                   rank  \
tech1
NaN    ['>#316,475 in Cell Phones & Accessories (See ...
NaN    ['>#316,475 in Cell Phones & Accessories (See ...
NaN    ['>#316,475 in Cell Phones & Accessories (See ...
NaN    ['>#316,475 in Cell Phones & Accessories (See ...
NaN    ['>#316,475 in Cell Phones & Accessories (See ...

                                               also_view             main_cat  \
tech1
NaN    ['B019PO1ZRI', 'B00W87LKXE', 'B019PO1ZNC', 'B0...  Musical Instruments
NaN    ['B019PO1ZRI', 'B00W87LKXE', 'B019PO1ZNC', 'B0...  Musical Instruments
NaN    ['B019PO1ZRI', 'B00W87LKXE', 'B019PO1ZNC', 'B0...  Musical Instruments
NaN    ['B019PO1ZRI', 'B00W87LKXE', 'B019PO1ZNC', 'B0...  Musical Instruments
NaN    ['B019PO1ZRI', 'B00W87LKXE', 'B019PO1ZNC', 'B0...  Musical Instruments

       ... verified   reviewTime      reviewerID  \
tech1  ...
NaN    ...     True  02 22, 2015  A38RQFVQ1AKJQQ
NaN    ...     True   05 8, 2017  A299MRB906GWDE
NaN    ...     True   11 5, 2016  A3ACFC6DQQLIQT
NaN    ...     True  09 24, 2016  A36BC0YFDBNB5X
NaN    ...     True  07 17, 2016  A212PQ0HQPNNWM

                                style   reviewerName  \
tech1
NaN          {'Color:': ' Blue W/Mic'}  George Walker
```

This converts accented characters into their base form and separates them from their accents. We use .encode('ASCII', 'ignore') to remove any remaining non-ASCII characters after normalization and .decode('utf-8') to decode the bytes back to a UTF-8 string.

c)

```python
# Dictionary mapping of acronyms to their full forms
acronym_expansion = {
    'ANC': 'Active Noise Cancellation',
    'AWC': 'Ambient Sound Control',
    'BQC': 'Bass Quality Control',
    'CSR': 'Customer Service Representative',
    'DAC': 'Digital-to-Analog Converter',
    'EQ': 'Equalizer',
    'NC': 'Noise Cancellation',
    'SBC': 'Sound Balance Control',
    'SNR': 'Signal-to-Noise Ratio',
    'THD': 'Total Harmonic Distortion',
    'TW': 'True Wireless',
    'UI': 'User Interface',
    'USB': 'Universal Serial Bus',
    'VSS': 'Virtual Surround Sound',
    'WLC': 'Wireless Charging',
    'BT': 'Bluetooth Technology',
    'ACT': 'Advanced Circuit Technology',
    'AH': 'A100 Headphones',
    'AIM': 'Audio Interface Mixer',
    'ALC': 'Auto Level Control',
    'AM': 'Acoustic Meshes',
    'ASP': 'Analogue Spatial Processing',
    'ATH': 'Audio Technica Headphones'
}

# Function to expand acronyms in text
def expand_acronyms(text):
    if isinstance(text, str):  # Check if the value is a string
        words = text.split()
        expanded_words = [acronym_expansion[word] if word in acronym_expansion else word for word in words]
        return ' '.join(expanded_words)
    else:
        return text  # Return the original value if not a string


# Apply the function to expand acronyms in 'reviewText' column
merged_df['reviewText'] = merged_df['reviewText'].apply(expand_acronyms)

# Print the dataframe to check the result
print(merged_df['reviewText'].head())
```

```
tech1
NaN    Great headphones. It's just the cord is too sh...
NaN    Really like these headphone. Wanted something ...
NaN    Wire to headphone broke off in less than a mon...
NaN                                            Very good
NaN    Currently returning this product because the s...
Name: reviewText, dtype: object
```

d)

Removing special chars

```python
# Function to remove special characters from text without using regular expressions
def remove_special_characters(text):
  if isinstance(text, str):
    empty_str = ''
    for char in text:
        if char.isalnum() or char.isspace():
          empty_str += char
    return empty_str
  else:
    return text

# Apply the function to remove special characters in 'reviewText' column
merged_df['reviewText'] = merged_df['reviewText'].apply(remove_special_characters)

# Print the dataframe to check the result
print(merged_df['reviewText'][:10])
```

```
tech1
NaN      Great headphones Its just the cord is too short
NaN    Really like these headphone Wanted something f...
NaN    Wire to headphone broke off in less than a mon...
NaN                                            Very good
NaN    Currently returning this product because the s...
NaN                                     Not good quality
NaN    The headphones work perfectly fine as in I can...
NaN    Bought these for my car for the kids and the c...
NaN                                              Garbage
NaN                                      Received broken
Name: reviewText, dtype: object
```

e)

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
nltk.download('punkt')
nltk.download('wordnet')

# Initialize the WordNet lemmatizer
lemmatizer = WordNetLemmatizer()

# Function to perform lemmatization on text
def text_lemmatization(text):
  if isinstance(text, str):
    tokens = word_tokenize(text)  # Tokenize the text into words
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in tokens]  # Lemmatize each token
    return ' '.join(lemmatized_tokens)  # Join the lemmatized tokens back into text
  else:
    return text

# Apply the function to perform lemmatization in 'reviewText' column
merged_df['reviewText'] = merged_df['reviewText'].apply(text_lemmatization)

# Print the dataframe to check the result
print(merged_df['reviewText'][:10])
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
tech1
NaN      Great headphone Its just the cord is too short
NaN    Really like these headphone Wanted something f...
NaN    Wire to headphone broke off in le than a month...
NaN                                           Very good
NaN    Currently returning this product because the s...
NaN                                    Not good quality
NaN    The headphone work perfectly fine a in I can o...
NaN    Bought these for my car for the kid and the co...
NaN                                             Garbage
NaN                                     Received broken
Name: reviewText, dtype: object
```

The code preprocesses text data in the 'reviewText' column of a DataFrame by performing lemmatization, which helps in standardizing words to their base or root form for better analysis or modeling in natural language processing tasks.

6.

a)

Top 20 most reviewed brands

```
# Group by 'brand' and count the number of reviews for each brand
brand_reviews = merged_df.groupby('brand').size().reset_index(name='reviews_count') #resets the index of the grouped data and renames the count column to 'review_count'.

# Sort the brands by the number of reviews in descending order
top_brands = brand_reviews.sort_values(by='reviews_count', ascending=False).head(20)

# Print the top 20 most reviewed brands
print(top_brands)
```

```
                brand  reviews_count
1743             Sony          34662
1674       Sennheiser          23999
353              Bose          10123
204     Audio-Technica          7763
1088             Koss           7608
1449        Panasonic           7576
991               JVC           7481
664       Etre Jeune           6310
1288             Mpow           5566
1467          Philips           5387
2270          iNassen           5354
339           Bluedio           5315
631            EldHus           5217
293            Belkin           4489
1185        MEE audio           4486
1811      Symphonized           4285
2100             XBRN           4245
1849       TaoTronics           4082
1960           V-MODA           4079
289             Beats           3916
```

b)

Top 20 least reviewed brands

```
# Sort the brands by the number of reviews in ascending order (least reviewed first)
least_reviewed_brands = brand_reviews.sort_values(by='reviews_count', ascending=True).head(20)

# Print the top 20 least reviewed brands
print(least_reviewed_brands)
```

```
                brand  reviews_count
906             Honda              2
1322     NOIZY Brands              3
42          AIRDRIVES              3
521               DSI              4
553    Digital family              5
609             EUBUY              5
1466  Phil Jones Bass              5
1973           VIILER              5
1471           Phrond              5
155           Amphony              5
156          Amplivox              5
1493        Powerseed              5
370      Burson Audio              5
1496          ProMaxi              5
1953       Unpluggify              5
1455          Pashion              5
1518            R I N              5
1521           RAYWAY              5
358         Bradychan              5
1525            REMAX              5
```

c)

Most positively reviewed 'Headphone'

```
# Calculate the average overall rating for each 'Headphone' product
avg_ratings = merged_df.groupby('title')['overall'].mean().reset_index(name='avg_rating')

# Find the most positively reviewed 'Headphone'
most_positively_reviewed = avg_ratings.sort_values(by='avg_rating', ascending=False).head(1)

# Print the most positively reviewed 'Headphone'
print(most_positively_reviewed)
```

```
                                                  title  avg_rating
2473  Emopeak Wireless Stereo Headsets Bluetooth Ove...         5.0
```

d)

The DataFrame df is filtered to include only rows corresponding to the 5 consecutive years starting from start_year. This is achieved by extracting the year from the 'reviewTime' column and converting it to an integer for comparison. It extracts the year from the 'reviewTime' column, counts the occurrences of each year, and then sorts the results by year in ascending order.

Ratings count over 5 consecutive years

```
# Define the start year of the 5-year period
start_year = 2010  # Start year of the 5-year period

# Filter out rows with NaN values in the 'date' column
df = merged_df.dropna(subset=['reviewTime'])

# Filter the dataframe for the 5 consecutive years
filtered_df = df[(df['reviewTime'].str[-4:].astype(int) >= start_year) & (df['reviewTime'].str[-4:].astype(int) <= start_year + 4)]

# Count the number of ratings for each year
ratings_count = filtered_df['reviewTime'].str[-4:].value_counts().sort_index()

# Print the count of ratings for each year
print(ratings_count)
```

```
reviewTime
2010     6903
2011    10021
2012    16961
2013    38404
2014    67039
Name: count, dtype: int64
```

e)

Word Clouds

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Separate 'Good' and 'Bad' reviews based on a threshold (e.g., ratings >= 3 as 'Good' and ratings < 3 as 'Bad')
good_reviews = merged_df[merged_df['overall'] >= 3]['reviewText'].str.cat(sep=' ')
bad_reviews = merged_df[merged_df['overall'] < 3]['reviewText'].str.cat(sep=' ')

# Generate word clouds for 'Good' and 'Bad' reviews separately
wordcloud_good = WordCloud(background_color='white').generate(good_reviews)
wordcloud_bad = WordCloud(background_color='white').generate(bad_reviews)

# Plot the word clouds
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(wordcloud_good, interpolation='bilinear')
plt.title('Word Cloud for Good Ratings')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(wordcloud_bad, interpolation='bilinear')
plt.title('Word Cloud for Bad Ratings')
plt.axis('off')

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

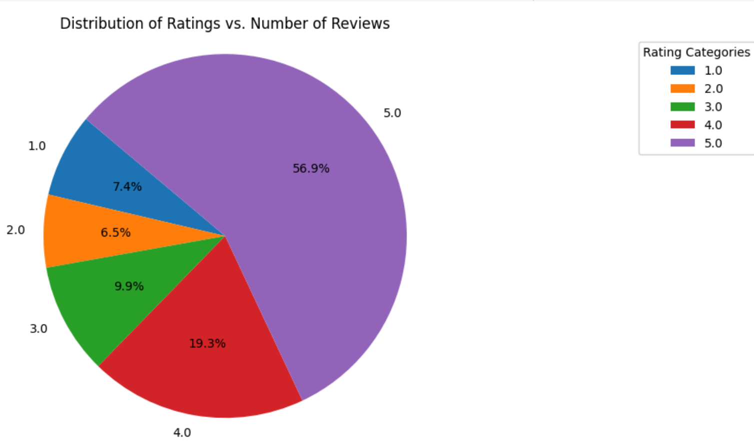
f)

Pie Chart

```
# Count the number of reviews for each rating
rating_counts = merged_df['overall'].value_counts().sort_index()

# Plotting the pie chart
plt.figure(figsize=(8, 6))
plt.pie(rating_counts, labels=rating_counts.index, autopct='%1.1f%%', startangle=140)
plt.title('Distribution of Ratings vs. Number of Reviews')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

# Displaying the legend for rating categories
plt.legend(title='Rating Categories', loc='upper right', bbox_to_anchor=(1, 0, 0.5, 1))
plt.show()
```



g)

Year with max reviews

```
# Convert the 'date' column to datetime format
merged_df['date'] = pd.to_datetime(merged_df['date'], errors='coerce')

# Extract the year from the 'date' column
merged_df['year'] = merged_df['date'].dt.year

# Count the number of reviews for each year
yearly_reviewCounts = merged_df['year'].value_counts()

# Find the year with the maximum reviews
maxReviews_year = yearly_reviewCounts.idxmax()

print(f"The year with the maximum reviews: {maxReviews_year}")


# Count the number of unique customers (reviewerID) for each year
yearly_customer_count = merged_df.groupby('year')['reviewerID'].nunique()

# Find the year with the highest number of customers
max_customer_year = yearly_customer_count.idxmax()

print(f"The year with the highest number of customers: {max_customer_year}")
```

```
The year with the maximum reviews: 2015.0
The year with the highest number of customers: 2015.0
```

7.

PART-7

[18]
```
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords
import string

# Download NLTK resources if not already downloaded
nltk.download('stopwords')

def textProcessing(text):
  if isinstance(text, str):  # Check if text is a string
    # Remove punctuations
    translator = str.maketrans('', '', string.punctuation)
    text = text.translate(translator)

    # Tokenization
    tokens = word_tokenize(text)

    # Lowercase the text
    lowercaseTokens = [token.lower() for token in tokens]

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

    # Remove blank space tokens
    tokens = [token for token in tokens if token.strip() != '']

    # Join tokens back into a single string
    processedText = ' '.join(tokens)

    return processedText

  else:
    return ''  # Return empty string for non-string values

merged_df['preprocessed_text'] = merged_df['reviewText'].apply(textProcessing)

# Initialize the TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=1000)  # You can adjust max_features as needed
```

```
        # Remove blank space tokens
        tokens = [token for token in tokens if token.strip() != '']

        # Join tokens back into a single string
        processedText = ' '.join(tokens)

        return processedText

    else:
        return ''  # Return empty string for non-string values

merged_df['preprocessed_text'] = merged_df['reviewText'].apply(textProcessing)

# Initialize the TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=1000)  # You can adjust max_features as needed

# Fit and transform the text data
tfidf_matrix_ = tfidf_vectorizer.fit_transform(merged_df['preprocessed_text'])

# Convert the TF-IDF matrix to a dataframe (optional, for visualization or further processing)
tfidf_dfr = pd.DataFrame(tfidf_matrix_.toarray(), columns=tfidf_vectorizer.get_feature_names_out())

# Display the TF-IDF dataframe
print(tfidf_dfr.head())
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
    10  100   12   14   15   20  200   25   30  300  ...  worth     would  \
0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0  0.000000
1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0  0.156612
2  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0  0.000000
3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0  0.000000
4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0  0.000000

   wouldnt  wow  wrong  year  yes  yet  youll  youre
0      0.0  0.0    0.0   0.0  0.0  0.0    0.0    0.0
1      0.0  0.0    0.0   0.0  0.0  0.0    0.0    0.0
2      0.0  0.0    0.0   0.0  0.0  0.0    0.0    0.0
3      0.0  0.0    0.0   0.0  0.0  0.0    0.0    0.0
4      0.0  0.0    0.0   0.0  0.0  0.0    0.0    0.0

[5 rows x 1000 columns]
```

8

:

```
[20] # Define a function to categorize ratings into classes
def categorize_rating(rating):
    if rating > 3:
        return 'Good'
    elif rating == 3:
        return 'Average'
    else:
        return 'Bad'

# Apply the function to create a new column 'Rating Class'
merged_df['Rating Class'] = merged_df['overall'].apply(categorize_rating)

# Print the first few rows to check the result
print(merged_df[['overall', 'Rating Class']].head(20))
```

```
       overall Rating Class
tech1
NaN        5.0         Good
NaN        5.0         Good
NaN        1.0          Bad
NaN        3.0      Average
NaN        1.0          Bad
NaN        1.0          Bad
NaN        1.0          Bad
NaN        4.0         Good
NaN        1.0          Bad
NaN        1.0          Bad
NaN        3.0      Average
NaN        1.0          Bad
NaN        4.0         Good
NaN        5.0         Good
NaN        1.0          Bad
NaN        3.0      Average
NaN        2.0          Bad
NaN        5.0         Good
NaN        1.0          Bad
NaN        1.0          Bad
```

9.

PART-9

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

# Filter out any unexpected values in the 'reviewText' column
merged_df = merged_df[merged_df['preprocessed_text'].apply(lambda x: isinstance(x, str))]

# Split the dataset into features (reviewText) and target variable (Rating Class)
X = merged_df['preprocessed_text']
Y = merged_df['Rating Class']

# Initialize TfidfVectorizer to convert text data to TF-IDF features
tfidf_vectorizer = TfidfVectorizer(max_features=1000)  # You can adjust max_features as needed

# Convert text data to TF-IDF features
X_tfidf = tfidf_vectorizer.fit_transform(X)

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, Y, test_size=0.25, random_state=42)
```

10.

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# Instantiate the model
logistic_regression_model = LogisticRegression()

# Train the model
logistic_regression_model.fit(X_train, y_train)

# Predictions
LR_y_pred = logistic_regression_model.predict(X_test)

# Classification report
LR_report = classification_report(y_test, LR_y_pred)
print("Logistic Regression Report:")
print(LR_report)
print("--------------------------------------------------")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Logistic Regression Report:
              precision    recall  f1-score   support

     Average       0.44      0.12      0.19     10156
         Bad       0.68      0.61      0.64     14512
        Good       0.86      0.96      0.91     78212

    accuracy                           0.83    102880
   macro avg       0.66      0.56      0.58    102880
weighted avg       0.80      0.83      0.80    102880

--------------------------------------------------
```

```python
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report

# Instantiate the model
decision_tree_model = DecisionTreeClassifier()

# Train the model
decision_tree_model.fit(X_train, y_train)

# Predictions
y_pred_dt = decision_tree_model.predict(X_test)

# Classification report
report_dt = classification_report(y_test, y_pred_dt)
print("Decision Tree Report:")
print(report_dt)
print("------------------------------------------------")

```

```
Decision Tree Report:
              precision    recall  f1-score   support

     Average       0.33      0.23      0.27       240
         Bad       0.63      0.50      0.56       311
        Good       0.93      0.96      0.95      3414

    accuracy                           0.88      3965
   macro avg       0.63      0.56      0.59      3965
weighted avg       0.87      0.88      0.87      3965
```

```python
# Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Instantiate the model
random_forest_model = RandomForestClassifier()

# Train the model
random_forest_model.fit(X_train, y_train)

# Predictions
y_pred_rf = random_forest_model.predict(X_test)

# Classification report
report_rf = classification_report(y_test, y_pred_rf)
print("Random Forest Report:")
print(report_rf)
print("------------------------------------------------")

```

```
Random Forest Report:
              precision    recall  f1-score   support

     Average       0.42      0.17      0.24       240
         Bad       0.67      0.53      0.59       311
        Good       0.92      0.98      0.95      3414

    accuracy                           0.89      3965
   macro avg       0.67      0.56      0.60      3965
weighted avg       0.87      0.89      0.88      3965
```

```python
# Gradient Boosting
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report

# Instantiate the model
gradient_boosting_model = GradientBoostingClassifier()

# Train the model
gradient_boosting_model.fit(X_train, y_train)

# Predictions
y_pred_gb = gradient_boosting_model.predict(X_test)

# Classification report
report_gb = classification_report(y_test, y_pred_gb)
print("Gradient Boosting Report:")
print(report_gb)
print("------------------------------------------------")

```

```
Gradient Boosting Report:
              precision    recall  f1-score   support

     Average       0.43      0.04      0.07       240
         Bad       0.79      0.30      0.43       311
        Good       0.89      1.00      0.94      3414

    accuracy                           0.88      3965
   macro avg       0.70      0.44      0.48      3965
weighted avg       0.85      0.88      0.85      3965
```

```python
# SVC
from sklearn.svm import SVC
from sklearn.metrics import classification_report

# Instantiate the model
svc_model = SVC()

# Train the model
svc_model.fit(X_train, y_train)

# Predictions
y_pred_svc = svc_model.predict(X_test)

# Classification report
report_svc = classification_report(y_test, y_pred_svc)
print("SVC Report:")
print(report_svc)
print("--------------------------------------------------")

```

```
SVC Report:
              precision    recall  f1-score   support

     Average       0.54      0.08      0.14       240
         Bad       0.77      0.50      0.60       311
        Good       0.91      0.99      0.95      3414

    accuracy                           0.90      3965
   macro avg       0.74      0.52      0.57      3965
weighted avg       0.87      0.90      0.87      3965

--------------------------------------------------
```