How Do Linguistic Patterns in Netflix Reviews Reflect the Overall Sentiment and Perceived Quality of the Content Being Reviewed?”

1) Can we accurately predict a reviewer’s rating based solely on their written text?

2) What themes or topics are most associated with positive versus negative reviews?

3) Do reviewers focus more on story, acting, or production quality when leaving extreme ratings?

Steup Chunk: 1

In [1]:
```
# Setup: Install Package
!pip install pandas numpy scikit-learn xgboost matplotlib seaborn nltk spacy
# !python -m spacy download en_core_web_sm (Commented)
!python -m nltk.downloader stopwords
!pip install tensorflow
!pip install transformers
!pip install tf-keras
```

```
Requirement already satisfied: pandas in e:\anaconda3\lib\site-packages (2.2.2)
Requirement already satisfied: numpy in e:\anaconda3\lib\site-packages (1.26.4)
Requirement already satisfied: scikit-learn in e:\anaconda3\lib\site-packages (1.5.
1)
Requirement already satisfied: xgboost in e:\anaconda3\lib\site-packages (2.0.3)
Requirement already satisfied: matplotlib in e:\anaconda3\lib\site-packages (3.8.4)
Requirement already satisfied: seaborn in e:\anaconda3\lib\site-packages (0.13.2)
Requirement already satisfied: nltk in e:\anaconda3\lib\site-packages (3.9.1)
Requirement already satisfied: spacy in e:\anaconda3\lib\site-packages (3.8.9)
Requirement already satisfied: python-dateutil>=2.8.2 in e:\anaconda3\lib\site-packa
ges (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in e:\anaconda3\lib\site-packages (from
pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in e:\anaconda3\lib\site-packages (fro
m pandas) (2023.3)
Requirement already satisfied: scipy>=1.6.0 in e:\anaconda3\lib\site-packages (from
scikit-learn) (1.13.0)
Requirement already satisfied: joblib>=1.2.0 in e:\anaconda3\lib\site-packages (from
scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in e:\anaconda3\lib\site-package
s (from scikit-learn) (3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in e:\anaconda3\lib\site-packages (f
rom matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in e:\anaconda3\lib\site-packages (from
matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in e:\anaconda3\lib\site-packages
(from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in e:\anaconda3\lib\site-packages
(from matplotlib) (1.4.4)
Requirement already satisfied: packaging>=20.0 in e:\anaconda3\lib\site-packages (fr
om matplotlib) (24.1)
Requirement already satisfied: pillow>=8 in e:\anaconda3\lib\site-packages (from mat
plotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in e:\anaconda3\lib\site-packages (f
rom matplotlib) (3.1.2)
Requirement already satisfied: click in e:\anaconda3\lib\site-packages (from nltk)
(8.1.7)
Requirement already satisfied: regex>=2021.8.3 in e:\anaconda3\lib\site-packages (fr
om nltk) (2024.9.11)
Requirement already satisfied: tqdm in e:\anaconda3\lib\site-packages (from nltk)
(4.66.5)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in e:\anaconda3\lib\site-
packages (from spacy) (3.0.12)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in e:\anaconda3\lib\site-
packages (from spacy) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in e:\anaconda3\lib\site-pa
ckages (from spacy) (1.0.14)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in e:\anaconda3\lib\site-packages
(from spacy) (2.0.12)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in e:\anaconda3\lib\site-packag
es (from spacy) (3.0.11)
Requirement already satisfied: thinc<8.4.0,>=8.3.4 in e:\anaconda3\lib\site-packages
(from spacy) (8.3.8)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in e:\anaconda3\lib\site-package
s (from spacy) (1.1.3)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in e:\anaconda3\lib\site-packages
```

```
(from spacy) (2.5.1)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in e:\anaconda3\lib\site-pack
ages (from spacy) (2.0.10)
Requirement already satisfied: weasel<0.5.0,>=0.4.2 in e:\anaconda3\lib\site-package
s (from spacy) (0.4.2)
Requirement already satisfied: typer-slim<1.0.0,>=0.3.0 in e:\anaconda3\lib\site-pac
kages (from spacy) (0.20.0)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in e:\anaconda3\lib\site-pack
ages (from spacy) (2.32.3)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in e:\anaconda3
\lib\site-packages (from spacy) (2.11.7)
Requirement already satisfied: jinja2 in e:\anaconda3\lib\site-packages (from spacy)
(3.1.4)
Requirement already satisfied: setuptools in e:\anaconda3\lib\site-packages (from sp
acy) (75.1.0)
Requirement already satisfied: annotated-types>=0.6.0 in e:\anaconda3\lib\site-packa
ges (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy) (0.6.0)
Requirement already satisfied: pydantic-core==2.33.2 in e:\anaconda3\lib\site-packag
es (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy) (2.33.2)
Requirement already satisfied: typing-extensions>=4.12.2 in e:\anaconda3\lib\site-pa
ckages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy) (4.14.1)
Requirement already satisfied: typing-inspection>=0.4.0 in e:\anaconda3\lib\site-pac
kages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy) (0.4.1)
Requirement already satisfied: six>=1.5 in e:\anaconda3\lib\site-packages (from pyth
on-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in e:\anaconda3\lib\site-pac
kages (from requests<3.0.0,>=2.13.0->spacy) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in e:\anaconda3\lib\site-packages (from
requests<3.0.0,>=2.13.0->spacy) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in e:\anaconda3\lib\site-packages
(from requests<3.0.0,>=2.13.0->spacy) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in e:\anaconda3\lib\site-packages
(from requests<3.0.0,>=2.13.0->spacy) (2025.8.3)
Requirement already satisfied: blis<1.4.0,>=1.3.0 in e:\anaconda3\lib\site-packages
(from thinc<8.4.0,>=8.3.4->spacy) (1.3.2)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in e:\anaconda3\lib\site-pac
kages (from thinc<8.4.0,>=8.3.4->spacy) (0.1.5)
Requirement already satisfied: colorama in e:\anaconda3\lib\site-packages (from tqdm
->nltk) (0.4.6)
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in e:\anaconda3\lib\site-p
ackages (from weasel<0.5.0,>=0.4.2->spacy) (0.23.0)
Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in e:\anaconda3\lib\site-pac
kages (from weasel<0.5.0,>=0.4.2->spacy) (5.2.1)
Requirement already satisfied: MarkupSafe>=2.0 in e:\anaconda3\lib\site-packages (fr
om jinja2->spacy) (2.1.3)
```

```
<frozen runpy>:128: RuntimeWarning: 'nltk.downloader' found in sys.modules after imp
ort of package 'nltk', but prior to execution of 'nltk.downloader'; this may result
in unpredictable behaviour
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\KUMKUM\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
Requirement already satisfied: tensorflow in e:\anaconda3\lib\site-packages (2.20.0)
Requirement already satisfied: absl-py>=1.0.0 in e:\anaconda3\lib\site-packages (fro
m tensorflow) (2.3.1)
Requirement already satisfied: astunparse>=1.6.0 in e:\anaconda3\lib\site-packages
(from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in e:\anaconda3\lib\site-package
s (from tensorflow) (25.9.23)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in e:\anaconda3\l
ib\site-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google_pasta>=0.1.1 in e:\anaconda3\lib\site-packages
(from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in e:\anaconda3\lib\site-packages (f
rom tensorflow) (18.1.1)
Requirement already satisfied: opt_einsum>=2.3.2 in e:\anaconda3\lib\site-packages
(from tensorflow) (3.4.0)
Requirement already satisfied: packaging in e:\anaconda3\lib\site-packages (from ten
sorflow) (24.1)
Requirement already satisfied: protobuf>=5.28.0 in e:\anaconda3\lib\site-packages (f
rom tensorflow) (6.33.1)
Requirement already satisfied: requests<3,>=2.21.0 in e:\anaconda3\lib\site-packages
(from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in e:\anaconda3\lib\site-packages (from te
nsorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in e:\anaconda3\lib\site-packages (from t
ensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in e:\anaconda3\lib\site-packages (f
rom tensorflow) (3.2.0)
Requirement already satisfied: typing_extensions>=3.6.6 in e:\anaconda3\lib\site-pac
kages (from tensorflow) (4.14.1)
Requirement already satisfied: wrapt>=1.11.0 in e:\anaconda3\lib\site-packages (from
tensorflow) (1.14.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in e:\anaconda3\lib\site-packages
(from tensorflow) (1.76.0)
Requirement already satisfied: tensorboard~=2.20.0 in e:\anaconda3\lib\site-packages
(from tensorflow) (2.20.0)
Requirement already satisfied: keras>=3.10.0 in e:\anaconda3\lib\site-packages (from
tensorflow) (3.12.0)
Requirement already satisfied: numpy>=1.26.0 in e:\anaconda3\lib\site-packages (from
tensorflow) (1.26.4)
Requirement already satisfied: h5py>=3.11.0 in e:\anaconda3\lib\site-packages (from
tensorflow) (3.11.0)
Requirement already satisfied: ml_dtypes<1.0.0,>=0.5.1 in e:\anaconda3\lib\site-pack
ages (from tensorflow) (0.5.3)
Requirement already satisfied: wheel<1.0,>=0.23.0 in e:\anaconda3\lib\site-packages
(from astunparse>=1.6.0->tensorflow) (0.44.0)
Requirement already satisfied: rich in e:\anaconda3\lib\site-packages (from keras>=
3.10.0->tensorflow) (13.7.1)
Requirement already satisfied: namex in e:\anaconda3\lib\site-packages (from keras>=
3.10.0->tensorflow) (0.1.0)
Requirement already satisfied: optree in e:\anaconda3\lib\site-packages (from keras>
=3.10.0->tensorflow) (0.18.0)
Requirement already satisfied: charset-normalizer<4,>=2 in e:\anaconda3\lib\site-pac
kages (from requests<3,>=2.21.0->tensorflow) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in e:\anaconda3\lib\site-packages (from
requests<3,>=2.21.0->tensorflow) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in e:\anaconda3\lib\site-packages
```

```
(from requests<3,>=2.21.0->tensorflow) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in e:\anaconda3\lib\site-packages
(from requests<3,>=2.21.0->tensorflow) (2025.8.3)
Requirement already satisfied: markdown>=2.6.8 in e:\anaconda3\lib\site-packages (fr
om tensorboard~=2.20.0->tensorflow) (3.4.1)
Requirement already satisfied: pillow in e:\anaconda3\lib\site-packages (from tensor
board~=2.20.0->tensorflow) (10.4.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in e:\anaconda3
\lib\site-packages (from tensorboard~=2.20.0->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in e:\anaconda3\lib\site-packages (fr
om tensorboard~=2.20.0->tensorflow) (3.0.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in e:\anaconda3\lib\site-packages
(from werkzeug>=1.0.1->tensorboard~=2.20.0->tensorflow) (2.1.3)
Requirement already satisfied: markdown-it-py>=2.2.0 in e:\anaconda3\lib\site-packag
es (from rich->keras>=3.10.0->tensorflow) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in e:\anaconda3\lib\site-pack
ages (from rich->keras>=3.10.0->tensorflow) (2.15.1)
Requirement already satisfied: mdurl~=0.1 in e:\anaconda3\lib\site-packages (from ma
rkdown-it-py>=2.2.0->rich->keras>=3.10.0->tensorflow) (0.1.0)
Requirement already satisfied: transformers in e:\anaconda3\lib\site-packages (4.57.
1)
Requirement already satisfied: filelock in e:\anaconda3\lib\site-packages (from tran
sformers) (3.13.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in e:\anaconda3\lib\site
-packages (from transformers) (0.34.3)
Requirement already satisfied: numpy>=1.17 in e:\anaconda3\lib\site-packages (from t
ransformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in e:\anaconda3\lib\site-packages (fr
om transformers) (24.1)
Requirement already satisfied: pyyaml>=5.1 in e:\anaconda3\lib\site-packages (from t
ransformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in e:\anaconda3\lib\site-packages
(from transformers) (2024.9.11)
Requirement already satisfied: requests in e:\anaconda3\lib\site-packages (from tran
sformers) (2.32.3)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in e:\anaconda3\lib\site-
packages (from transformers) (0.22.1)
Requirement already satisfied: safetensors>=0.4.3 in e:\anaconda3\lib\site-packages
(from transformers) (0.6.2)
Requirement already satisfied: tqdm>=4.27 in e:\anaconda3\lib\site-packages (from tr
ansformers) (4.66.5)
Requirement already satisfied: fsspec>=2023.5.0 in e:\anaconda3\lib\site-packages (f
rom huggingface-hub<1.0,>=0.34.0->transformers) (2024.6.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in e:\anaconda3\lib\site-p
ackages (from huggingface-hub<1.0,>=0.34.0->transformers) (4.14.1)
Requirement already satisfied: colorama in e:\anaconda3\lib\site-packages (from tqdm
>=4.27->transformers) (0.4.6)
Requirement already satisfied: charset-normalizer<4,>=2 in e:\anaconda3\lib\site-pac
kages (from requests->transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in e:\anaconda3\lib\site-packages (from
requests->transformers) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in e:\anaconda3\lib\site-packages
(from requests->transformers) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in e:\anaconda3\lib\site-packages
(from requests->transformers) (2025.8.3)
Requirement already satisfied: tf-keras in e:\anaconda3\lib\site-packages (2.20.1)
```

```
Requirement already satisfied: tensorflow<2.21,>=2.20 in e:\anaconda3\lib\site-packa
ges (from tf-keras) (2.20.0)
Requirement already satisfied: absl-py>=1.0.0 in e:\anaconda3\lib\site-packages (fro
m tensorflow<2.21,>=2.20->tf-keras) (2.3.1)
Requirement already satisfied: astunparse>=1.6.0 in e:\anaconda3\lib\site-packages
(from tensorflow<2.21,>=2.20->tf-keras) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in e:\anaconda3\lib\site-package
s (from tensorflow<2.21,>=2.20->tf-keras) (25.9.23)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in e:\anaconda3\l
ib\site-packages (from tensorflow<2.21,>=2.20->tf-keras) (0.6.0)
Requirement already satisfied: google_pasta>=0.1.1 in e:\anaconda3\lib\site-packages
(from tensorflow<2.21,>=2.20->tf-keras) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in e:\anaconda3\lib\site-packages (f
rom tensorflow<2.21,>=2.20->tf-keras) (18.1.1)
Requirement already satisfied: opt_einsum>=2.3.2 in e:\anaconda3\lib\site-packages
(from tensorflow<2.21,>=2.20->tf-keras) (3.4.0)
Requirement already satisfied: packaging in e:\anaconda3\lib\site-packages (from ten
sorflow<2.21,>=2.20->tf-keras) (24.1)
Requirement already satisfied: protobuf>=5.28.0 in e:\anaconda3\lib\site-packages (f
rom tensorflow<2.21,>=2.20->tf-keras) (6.33.1)
Requirement already satisfied: requests<3,>=2.21.0 in e:\anaconda3\lib\site-packages
(from tensorflow<2.21,>=2.20->tf-keras) (2.32.3)
Requirement already satisfied: setuptools in e:\anaconda3\lib\site-packages (from te
nsorflow<2.21,>=2.20->tf-keras) (75.1.0)
Requirement already satisfied: six>=1.12.0 in e:\anaconda3\lib\site-packages (from t
ensorflow<2.21,>=2.20->tf-keras) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in e:\anaconda3\lib\site-packages (f
rom tensorflow<2.21,>=2.20->tf-keras) (3.2.0)
Requirement already satisfied: typing_extensions>=3.6.6 in e:\anaconda3\lib\site-pac
kages (from tensorflow<2.21,>=2.20->tf-keras) (4.14.1)
Requirement already satisfied: wrapt>=1.11.0 in e:\anaconda3\lib\site-packages (from
tensorflow<2.21,>=2.20->tf-keras) (1.14.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in e:\anaconda3\lib\site-packages
(from tensorflow<2.21,>=2.20->tf-keras) (1.76.0)
Requirement already satisfied: tensorboard~=2.20.0 in e:\anaconda3\lib\site-packages
(from tensorflow<2.21,>=2.20->tf-keras) (2.20.0)
Requirement already satisfied: keras>=3.10.0 in e:\anaconda3\lib\site-packages (from
tensorflow<2.21,>=2.20->tf-keras) (3.12.0)
Requirement already satisfied: numpy>=1.26.0 in e:\anaconda3\lib\site-packages (from
tensorflow<2.21,>=2.20->tf-keras) (1.26.4)
Requirement already satisfied: h5py>=3.11.0 in e:\anaconda3\lib\site-packages (from
tensorflow<2.21,>=2.20->tf-keras) (3.11.0)
Requirement already satisfied: ml_dtypes<1.0.0,>=0.5.1 in e:\anaconda3\lib\site-pack
ages (from tensorflow<2.21,>=2.20->tf-keras) (0.5.3)
Requirement already satisfied: wheel<1.0,>=0.23.0 in e:\anaconda3\lib\site-packages
(from astunparse>=1.6.0->tensorflow<2.21,>=2.20->tf-keras) (0.44.0)
Requirement already satisfied: rich in e:\anaconda3\lib\site-packages (from keras>=
3.10.0->tensorflow<2.21,>=2.20->tf-keras) (13.7.1)
Requirement already satisfied: namex in e:\anaconda3\lib\site-packages (from keras>=
3.10.0->tensorflow<2.21,>=2.20->tf-keras) (0.1.0)
Requirement already satisfied: optree in e:\anaconda3\lib\site-packages (from keras>
=3.10.0->tensorflow<2.21,>=2.20->tf-keras) (0.18.0)
Requirement already satisfied: charset-normalizer<4,>=2 in e:\anaconda3\lib\site-pac
kages (from requests<3,>=2.21.0->tensorflow<2.21,>=2.20->tf-keras) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in e:\anaconda3\lib\site-packages (from
requests<3,>=2.21.0->tensorflow<2.21,>=2.20->tf-keras) (3.7)
```

```
Requirement already satisfied: urllib3<3,>=1.21.1 in e:\anaconda3\lib\site-packages
(from requests<3,>=2.21.0->tensorflow<2.21,>=2.20->tf-keras) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in e:\anaconda3\lib\site-packages
(from requests<3,>=2.21.0->tensorflow<2.21,>=2.20->tf-keras) (2025.8.3)
Requirement already satisfied: markdown>=2.6.8 in e:\anaconda3\lib\site-packages (fr
om tensorboard~=2.20.0->tensorflow<2.21,>=2.20->tf-keras) (3.4.1)
Requirement already satisfied: pillow in e:\anaconda3\lib\site-packages (from tensor
board~=2.20.0->tensorflow<2.21,>=2.20->tf-keras) (10.4.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in e:\anaconda3
\lib\site-packages (from tensorboard~=2.20.0->tensorflow<2.21,>=2.20->tf-keras) (0.
7.2)
Requirement already satisfied: werkzeug>=1.0.1 in e:\anaconda3\lib\site-packages (fr
om tensorboard~=2.20.0->tensorflow<2.21,>=2.20->tf-keras) (3.0.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in e:\anaconda3\lib\site-packages
(from werkzeug>=1.0.1->tensorboard~=2.20.0->tensorflow<2.21,>=2.20->tf-keras) (2.1.
3)
Requirement already satisfied: markdown-it-py>=2.2.0 in e:\anaconda3\lib\site-packag
es (from rich->keras>=3.10.0->tensorflow<2.21,>=2.20->tf-keras) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in e:\anaconda3\lib\site-pack
ages (from rich->keras>=3.10.0->tensorflow<2.21,>=2.20->tf-keras) (2.15.1)
Requirement already satisfied: mdurl~=0.1 in e:\anaconda3\lib\site-packages (from ma
rkdown-it-py>=2.2.0->rich->keras>=3.10.0->tensorflow<2.21,>=2.20->tf-keras) (0.1.0)
```

Steup Chunk: 2

In [2]:
```python
import spacy

print("Trying to load spaCy model...")
try:
    nlp = spacy.load("en_core_web_sm")
    print("✅ spaCy model loaded OK!")
except Exception as e:
    print("❌ spaCy model did NOT load.")
    print("Error message:")
    print(e)
```

```
Trying to load spaCy model...
✅ spaCy model loaded OK!
```

In [ ]:

Importing ALL Required Libraries

In this section, I load all the essential Python libraries needed for data handling, exploratory analysis, text preprocessing (NLTK + spaCy), TF-IDF feature extraction, and machine learning models used throughout the Netflix review NLP project.

In [3]:
```python
# Import Libraries

# Pandas for loading, cleaning, and manipulating the Netflix review dataset
import pandas as pd

# NumPy for numerical operations and vectorized processing during feature engineeri
```

```
import numpy as np

# Matplotlib for creating basic visualizations (rating distributions, word frequenc
import matplotlib.pyplot as plt

# Seaborn for more advanced and aesthetically pleasing visualizations
import seaborn as sns

# train_test_split for dividing the dataset into training and testing sets
from sklearn.model_selection import train_test_split

# TfidfVectorizer for converting review text into TF-IDF feature vectors
from sklearn.feature_extraction.text import TfidfVectorizer

# Pipeline allows me to bundle text preprocessing + modeling into a single workflow
from sklearn.pipeline import Pipeline

# Evaluation metrics to assess accuracy and classification performance
from sklearn.metrics import (
    accuracy_score,           # Overall model accuracy
    classification_report,    # Precision, recall, f1-score per class
    confusion_matrix          # Heatmap-ready confusion matrix
)

# Logistic Regression as a baseline linear classifier for sentiment/rating predicti
from sklearn.linear_model import LogisticRegression

# Naive Bayes classifier — usually strong for text classification tasks
from sklearn.naive_bayes import MultinomialNB

# Linear Support Vector Classifier — powerful for high-dimensional text features
from sklearn.svm import LinearSVC

# XGBoost classifier — gradient boosting model for potentially stronger accuracy
from xgboost import XGBClassifier

# NLTK for classical NLP preprocessing: stopwords, tokenization, etc.
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# spaCy for advanced linguistic pattern extraction: tokens, POS tags, dependencies
import spacy
nlp = spacy.load("en_core_web_sm")  # Load the small English model

# Regular expressions for text cleaning (removing URLs, digits, special characters)
import re

# String library for handling punctuation removal and character checks
import string

# Set seaborn visual style and default plot size
sns.set(style="whitegrid")
plt.rcParams["figure.figsize"] = (7, 4)

# Deep Learning: Bi-LSTM
```

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

# Deep Learning: Label Encoding for DL models
from sklearn.preprocessing import LabelEncoder

# Deep Learning: BERT (Transformers)
from transformers import BertTokenizerFast, TFBertForSequenceClassification
```

```
WARNING:tensorflow:From E:\anaconda3\Lib\site-packages\tf_keras\src\losses.py:2976:
The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.
v1.losses.sparse_softmax_cross_entropy instead.
```

Loading the dataset: Netflix Reviews

In [4]:

```
# Loading Dataset
CSV_PATH = r"D:\Study Material\CIS 9665 Applied Natural Language Proce\Term Project
df = pd.read_csv(CSV_PATH)
df.head()
```

Out[4]:

	reviewId	userName	content	score	thumbsUpCount	reviewCreatedVersion	
0	efd00499-5e00-49b5-9f32-bc7177ac5ca6	Mikel Magnusson	Netfix Canada forced my wife into a screen tha...	1	0	8.93.1 build 4 50540	21:
1	be0d97e1-7de1-4f07-b493-35a53098b5a4	John McDevitt	I use this app until it asks if I'm still ther...	2	0	8.119.0 build 11 50706	21:
2	8970dbcd-d75f-4016-bb93-efa5de3ef9e6	Mayur Savaliya	Boycott Netflix from Bharat	1	1	8.14.0 build 5 40129	21:
3	a288bc3c-8a90-42d3-b585-1c8078faa96c	Magdalena Glessing	Little good movies and a lot of wonderful TV s...	5	0	8.118.1 build 10 50703	21:
4	c388a806-0795-4812-b04e-5b2cdf327157	Elizabeth Turner	New to this but, so far smooth sailing.app is ...	5	0	8.118.1 build 10 50703	21:

Basic Exploratory Data Analysis (EDA):

In [5]:
```
# Basic Exploratory Data Analysis (EDA)
# Check the dataset shape / how many rows and columns in datasets
print("Dataset Shape:", df.shape)

print("="*100)

# Checking the column names
print("Columns in Dataset:")
print(df.columns)

print("="*100)

# Summary of data types and missing values
print("\nDataset Info:")
print(df.info())
```

```python
print("="*100)

# Count missing values in each column
print("\nMissing Values per Column:")
print(df.isnull().sum())

print("="*100)

# Basic descriptive statistics, useful for numerical columns like rating
print("\nDescriptive Statistics:")
df.describe()
```

```
Dataset Shape: (113068, 8)
====================================================================================
================
Columns in Dataset:
Index(['reviewId', 'userName', 'content', 'score', 'thumbsUpCount',
       'reviewCreatedVersion', 'at', 'appVersion'],
      dtype='object')
====================================================================================
================

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113068 entries, 0 to 113067
Data columns (total 8 columns):
 #   Column                Non-Null Count   Dtype 
---  ------                --------------   ----- 
 0   reviewId              113068 non-null  object
 1   userName              113067 non-null  object
 2   content               113066 non-null  object
 3   score                 113068 non-null  int64 
 4   thumbsUpCount         113068 non-null  int64 
 5   reviewCreatedVersion  96577 non-null   object
 6   at                    113068 non-null  object
 7   appVersion            96577 non-null   object
dtypes: int64(2), object(6)
memory usage: 6.9+ MB
None
====================================================================================
================

Missing Values per Column:
reviewId                    0
userName                    1
content                     2
score                       0
thumbsUpCount               0
reviewCreatedVersion    16491
at                          0
appVersion              16491
dtype: int64
====================================================================================
================

Descriptive Statistics:
```

Out[5]:

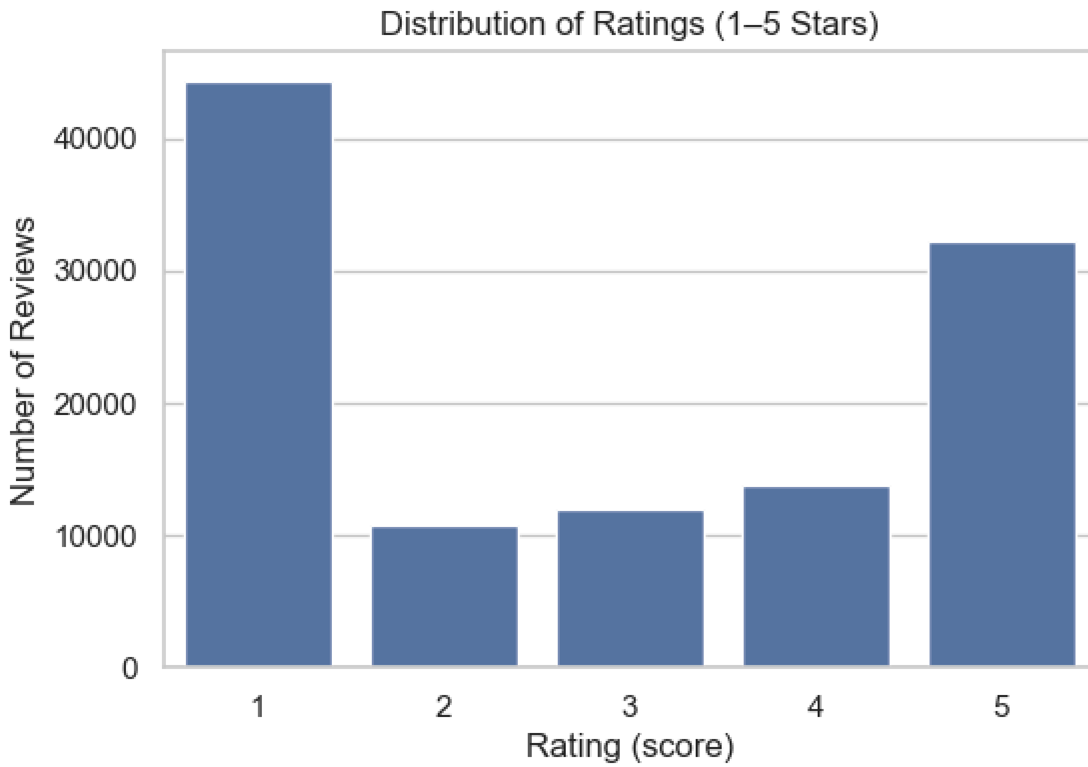
	score	thumbsUpCount
count	113068.000000	113068.000000
mean	2.812140	10.483833
std	1.699847	101.252663
min	1.000000	0.000000
25%	1.000000	0.000000
50%	3.000000	0.000000
75%	5.000000	1.000000
max	5.000000	8032.000000

This section analyzes how ratings (1–5 stars) are distributed in the dataset, helping identify class imbalance and understanding overall user sentiment patterns.

In [6]:

```
# Linguistic Pattern Analysis: Rating Distribution
# Detail EDA: To see how many reviews we have for each rating (1-5 stars)

print("Ratings Distribution Plots ")
TEXT_COL = "content"
RATING_COL = "score"

plt.figure(figsize=(6, 4))
sns.countplot(x=df[RATING_COL])
plt.title("Distribution of Ratings (1–5 Stars)")
plt.xlabel("Rating (score)")
plt.ylabel("Number of Reviews")
plt.show()

# Print an interpretation of the rating distribution
print("Observation:")
print("="*100)
print("- The dataset is dominated by 1-star and 5-star reviews.")
print("- 1-star reviews appear most frequently, indicating many negative user exper
print("- 5-star reviews are the second highest, showing a strong group of satisfied
print("- Mid-range ratings (2, 3, 4 stars) have much fewer samples.")
print("- This means the data is slightly imbalanced but still suitable for multi-cl
```

```
Ratings Distribution Plots
```



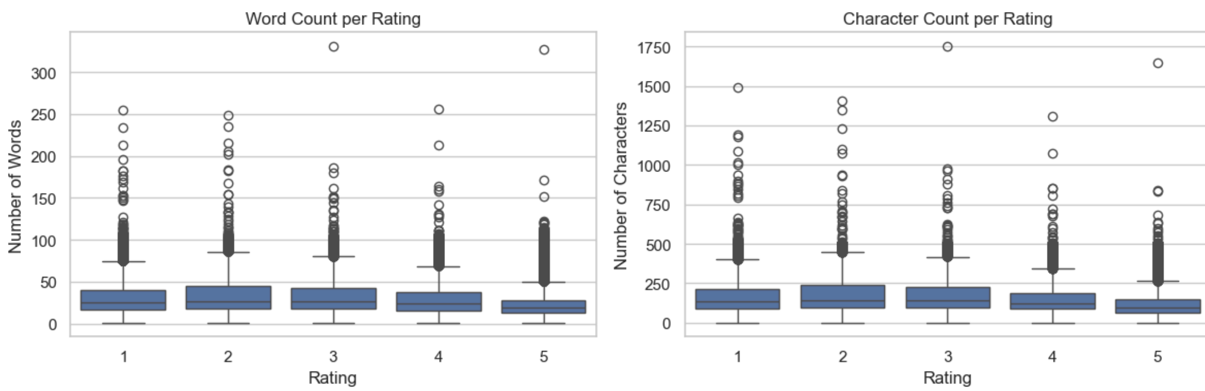

```
Observation:
================================================================================
================
- The dataset is dominated by 1-star and 5-star reviews.
- 1-star reviews appear most frequently, indicating many negative user experiences.
- 5-star reviews are the second highest, showing a strong group of satisfied users.
- Mid-range ratings (2, 3, 4 stars) have much fewer samples.
- This means the data is slightly imbalanced but still suitable for multi-class clas
sification.
```

This section examines how long reviews are (in words and characters) across different rating levels, helping identify whether review length correlates with user sentiment or rating patterns.

In [7]:

```python
# Linguistic Pattern Analysis: Reviewing Length Analysis (Words & Characters)
# EDA: To check how long reviews are for each rating

# creating simple length features
df["word_count"] = df[TEXT_COL].str.split().str.len()
df["char_count"] = df[TEXT_COL].str.len()

fig, ax = plt.subplots(1, 2, figsize=(12, 4))

# word count by rating
sns.boxplot(x=df[RATING_COL], y=df["word_count"], ax=ax[0])
ax[0].set_title("Word Count per Rating")
ax[0].set_xlabel("Rating")
ax[0].set_ylabel("Number of Words")

# character count by rating
```

```
sns.boxplot(x=df[RATING_COL], y=df["char_count"], ax=ax[1])
ax[1].set_title("Character Count per Rating")
ax[1].set_xlabel("Rating")
ax[1].set_ylabel("Number of Characters")

plt.tight_layout()
plt.show()

print("Observation:")
print("="*100)
print("Review length analysis shows mild linguistic differences across ratings. "
      "Lower-rated reviews (1–2 stars) are generally shorter and more abrupt, "
      "while higher-rated reviews (4–5 stars) tend to be longer and more descriptiv
      "This supports the idea that linguistic style varies with user sentiment.")
```

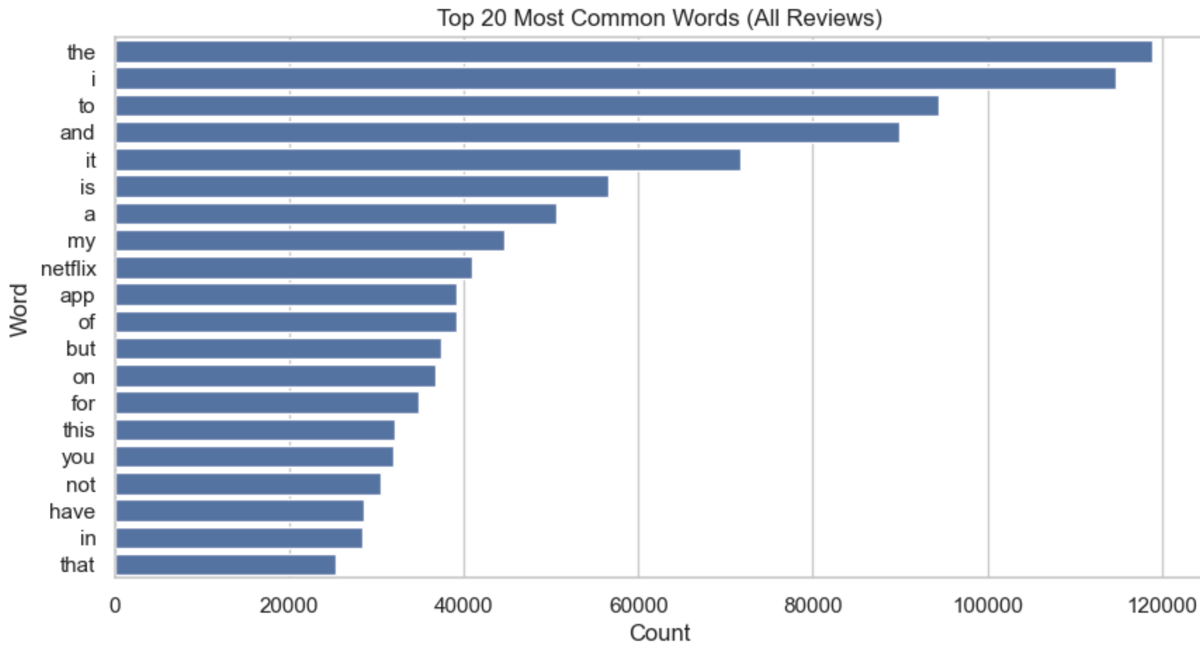



```
Observation:
====================================================================================================
Review length analysis shows mild linguistic differences across ratings. Lower-rated
reviews (1–2 stars) are generally shorter and more abrupt, while higher-rated review
s (4–5 stars) tend to be longer and more descriptive. This supports the idea that li
nguistic style varies with user sentiment.
```

In [8]:

```
# Linguistic Pattern Analysis: Word Frequency
# EDA: To see the most common words in all reviews

from collections import Counter

# join all reviews into one big string, make lowercase, split on spaces
all_words = " ".join(df[TEXT_COL].astype(str).str.lower()).split()

word_freq = Counter(all_words)
common_words = pd.DataFrame(word_freq.most_common(20), columns=["word", "count"])

plt.figure(figsize=(10, 5))
sns.barplot(x="count", y="word", data=common_words)
plt.title("Top 20 Most Common Words (All Reviews)")
plt.xlabel("Count")
plt.ylabel("Word")
plt.show()

print("Observation:")
print("="*100)
print("The top 20 most frequent words are dominated by standard English stopwords s
      "These high-frequency neutral terms confirm the natural writing style of user
```

```
      "value for sentiment or rating prediction. Their prevalence highlights the im
      "stopword removal during preprocessing. Domain-relevant terms such as 'netfli
      "indicating that the dataset is contextually consistent.")
```

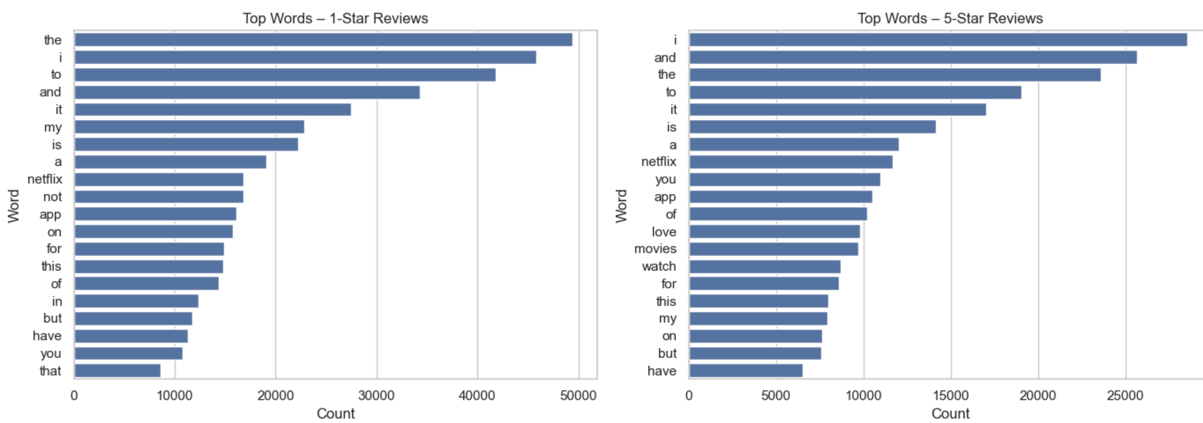



```
Observation:
================================================================================
================
The top 20 most frequent words are dominated by standard English stopwords such as
'the', 'i', and 'to'. These high-frequency neutral terms confirm the natural writing
style of user reviews but provide little value for sentiment or rating prediction. T
heir prevalence highlights the importance of applying stopword removal during prepro
cessing. Domain-relevant terms such as 'netflix' and 'app' also appear, indicating t
hat the dataset is contextually consistent.
```

This section compares the most frequent words used in extremely negative (1-star) and extremely positive (5-star) reviews to highlight clear linguistic differences across sentiment levels.

In [9]:
```
# Linguistic Pattern Analysis: Word Frequency by Rating _ Top Words in Negative vs
# EDA: To compare the common words in 1-star vs 5-star reviews

from collections import Counter

# filter extreme negative and extreme positive reviews
neg_text = df[df[RATING_COL] == 1][TEXT_COL].astype(str).str.lower().str.cat(sep="
pos_text = df[df[RATING_COL] == 5][TEXT_COL].astype(str).str.lower().str.cat(sep="

neg_freq = Counter(neg_text.split()).most_common(20)
pos_freq = Counter(pos_text.split()).most_common(20)

neg_df = pd.DataFrame(neg_freq, columns=["word", "count"])
pos_df = pd.DataFrame(pos_freq, columns=["word", "count"])

fig, ax = plt.subplots(1, 2, figsize=(14, 5))
```

```
sns.barplot(x="count", y="word", data=neg_df, ax=ax[0])
ax[0].set_title("Top Words – 1-Star Reviews")
ax[0].set_xlabel("Count")
ax[0].set_ylabel("Word")

sns.barplot(x="count", y="word", data=pos_df, ax=ax[1])
ax[1].set_title("Top Words – 5-Star Reviews")
ax[1].set_xlabel("Count")
ax[1].set_ylabel("Word")

plt.tight_layout()
plt.show()

print("Observation:")
print("="*100)
print("The comparison of word frequencies shows that both 1-star and 5-star reviews
      "contain common stopwords, which is expected in natural language text. "
      "However, positive reviews are more likely to include affirming terms such as
      "'love', 'movies', and 'watch', while negative reviews show a higher occurren
      "of negation terms like 'not'. These differences provide early evidence that
      "linguistic patterns vary with user sentiment, supporting the suitability of
      "this dataset for sentiment analysis and rating prediction tasks.")
```



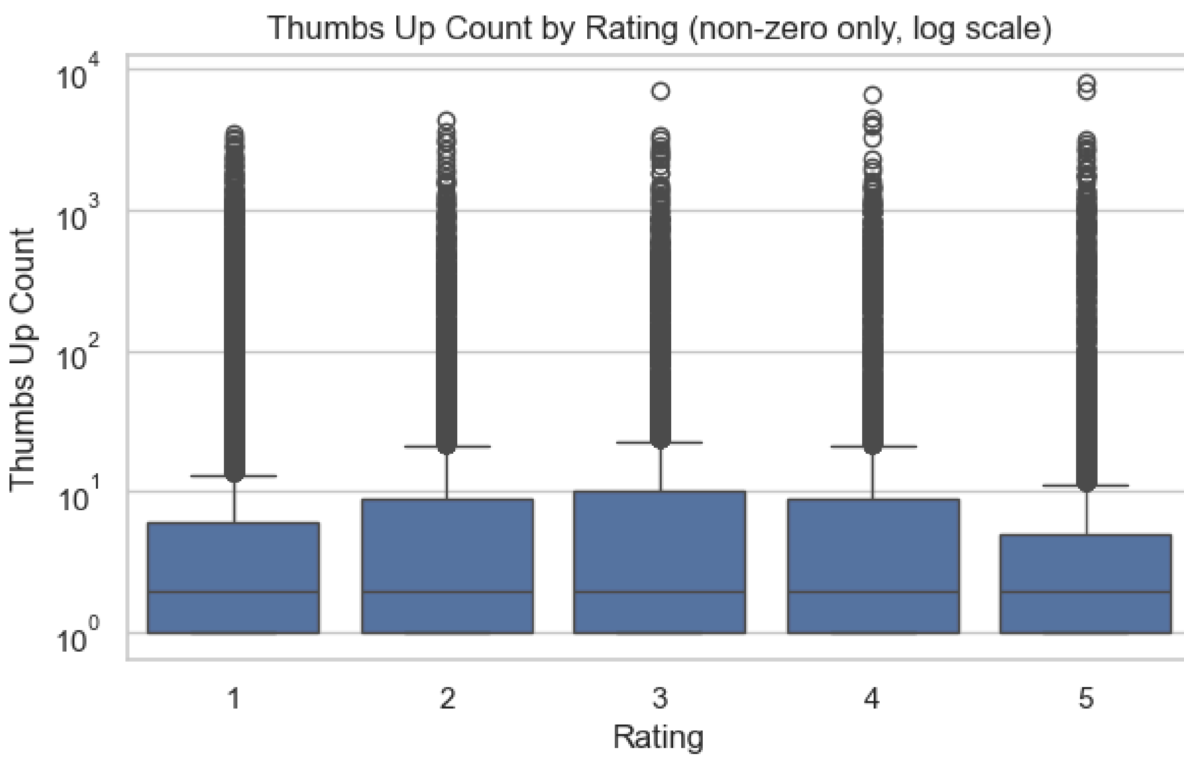

```
Observation:
====================================================================================
================
The comparison of word frequencies shows that both 1-star and 5-star reviews contain
common stopwords, which is expected in natural language text. However, positive revi
ews are more likely to include affirming terms such as 'love', 'movies', and 'watc
h', while negative reviews show a higher occurrence of negation terms like 'not'. Th
ese differences provide early evidence that linguistic patterns vary with user senti
ment, supporting the suitability of this dataset for sentiment analysis and rating p
rediction tasks.
```

This section examines how many “helpful” votes each rating category receives to understand whether user engagement is influenced by review sentiment or rating level.

In [10]:

```
# Linguistic Pattern Analysis: ThumbsUpCount vs Rating
# EDA: Check how many "helpful" votes reviews get by rating
```

```
df_nonzero = df[df["thumbsUpCount"] > 0]

plt.figure(figsize=(7, 4))
sns.boxplot(x=df_nonzero[RATING_COL], y=df_nonzero["thumbsUpCount"])
plt.yscale("log")
plt.title("Thumbs Up Count by Rating (non-zero only, log scale)")
plt.xlabel("Rating")
plt.ylabel("Thumbs Up Count")
plt.show()

print("Observation:")
print("="*100)
print("Analyzing non-zero helpful votes on a log scale reveals that user engagement
      "Both highly negative (1-star) and highly positive (5-star) reviews show elev
      "expressing strong opinions—regardless of sentiment—attract more attention. T
      "of helpful votes, with only a minority showing high engagement.")
```

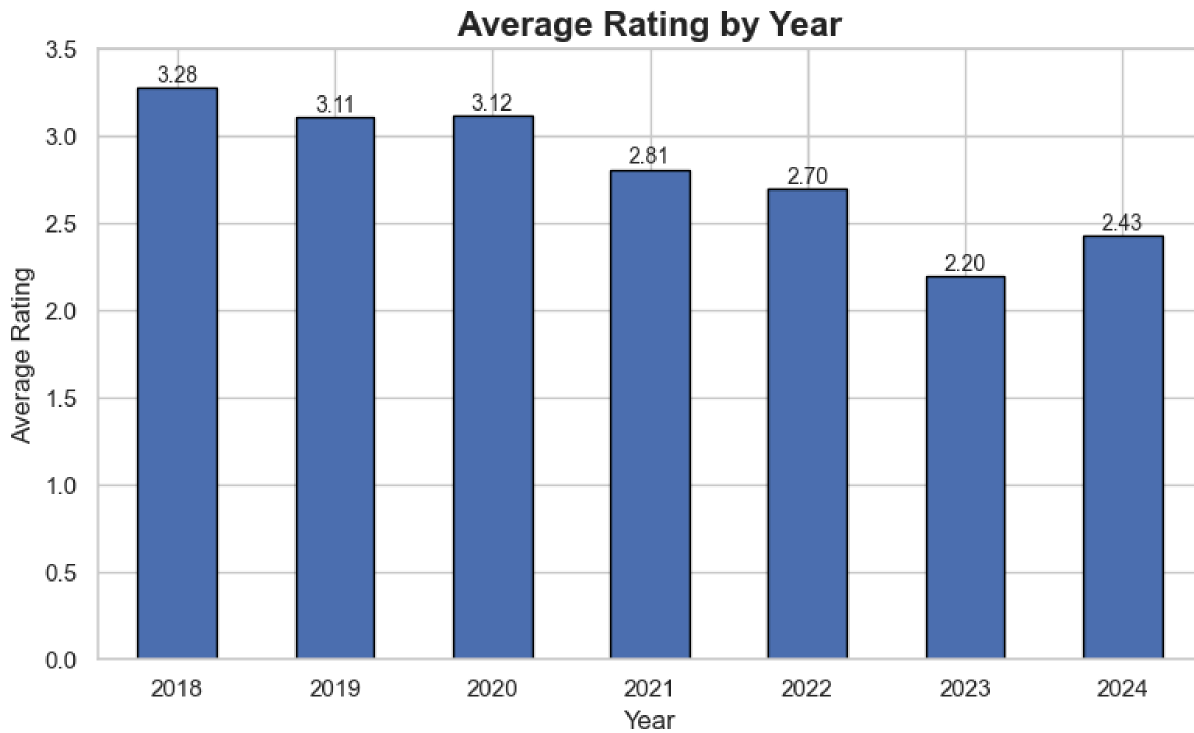



```
Observation:
====================================================================================================
Analyzing non-zero helpful votes on a log scale reveals that user engagement is not
tied to a specific rating category. Both highly negative (1-star) and highly positiv
e (5-star) reviews show elevated helpful counts, indicating that reviews expressing
strong opinions—regardless of sentiment—attract more attention. The majority of revi
ews receive a small number of helpful votes, with only a minority showing high engag
ement.
```

This section analyzes how average user ratings change over time to identify long-term trends in user sentiment and app satisfaction.

In [11]:

```
# Linguistic Pattern Analysis: Version / Date Checks
# EDA: Look at average rating over time

# Make sure we have a proper datetime column called 'review_date'
df["review_date"] = pd.to_datetime(df["at"], errors="coerce")

# Compute the average rating per year
rating_by_year = (
    df.dropna(subset=["review_date"])
      .groupby(df["review_date"].dt.year)[RATING_COL]
      .mean()
      .round(2)
)

# Plot the trend over time
plt.figure(figsize=(8, 5))
sns.set_theme(style="whitegrid")
ax = rating_by_year.plot(kind="bar", color="#4C72B0", edgecolor="black")

# Titles and labels
plt.title("Average Rating by Year", fontsize=16, weight="bold")
plt.xlabel("Year", fontsize=12)
plt.ylabel("Average Rating", fontsize=12)

# Show value labels on top of bars
for i, value in enumerate(rating_by_year.values):
    plt.text(i, value + 0.03, f"{value:.2f}", ha="center", fontsize=10)

# Clean up x-axis labels
plt.xticks(rotation=0, fontsize=11)
plt.yticks(fontsize=11)

# Set a consistent y-limit for better comparison
plt.ylim(0, 3.5)

plt.tight_layout()
plt.show()

print("Observation:")
print("="*100)
print("The temporal analysis reveals a clear downward trend in user satisfaction. "
      "Average ratings were relatively high between 2018 and 2020, after which a st
      "is observed, reaching the lowest point in 2023. Although 2024 shows a slight
      "the overall pattern indicates that user sentiment toward the app has weakene
      "This trend may be driven by application updates, performance issues, or evol
```




```
Observation:
==================================================================================
================
The temporal analysis reveals a clear downward trend in user satisfaction. Average r
atings were relatively high between 2018 and 2020, after which a steady decline is o
bserved, reaching the lowest point in 2023. Although 2024 shows a slight recovery, t
he overall pattern indicates that user sentiment toward the app has weakened over ti
me. This trend may be driven by application updates, performance issues, or evolving
user expectations.
```

Text Preprocessing (Cleaning + Stopwords + Lemmatization)

This section performs text preprocessing to clean and normalize the review text. I convert text to lowercase, remove noise (punctuation, digits, URLs), remove stopwords, and apply spaCy lemmatization to create a clean and standardized version of each review for NLP modeling.

In [12]:
```python
# Text Preprocessing (Cleaning + Stopwords + Lemmatization)
# Purpose: To Prepare clean and normalized text for NLP modeling

# Download NLTK resources
nltk.download("stopwords")
nltk.download("wordnet")

stop_words = set(stopwords.words("english"))
lemmatizer = WordNetLemmatizer()

def fast_preprocess(text):
    if not isinstance(text, str):
        return ""

    # Lowercase
    text = text.lower()
```

```
    # Remove URLs
    text = re.sub(r"http\S+|www\S+|https\S+", "", text)

    # Remove digits
    text = re.sub(r"\d+", "", text)

    # Remove punctuation
    text = text.translate(str.maketrans("", "", string.punctuation))

    # Remove extra spaces
    text = re.sub(r"\s+", " ", text).strip()

    # Tokenize + remove stopwords + lemmatize
    words = text.split()
    words = [lemmatizer.lemmatize(w) for w in words if w not in stop_words]

    return " ".join(words)

print("Cleaning text... (this should be faster)")
df["clean_text"] = df["content"].apply(fast_preprocess)

print("Text Preprocessing Completed!")
print("Sample cleaned text:")
print(df["clean_text"].head().to_list())
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\KUMKUM\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\KUMKUM\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
Cleaning text... (this should be faster)
Text Preprocessing Completed!
Sample cleaned text:
['netfix canada forced wife screen would allow exit adding new member account immedi
ately billed month increased monthly billing amount crook cancelled account year fil
ed complaint credit card provider refund remainer current month service new member f
ee uninstalled app two phone two tablet chromecasts', 'use app asks im still move an
other service thats less infuriating already sporadically subscribe dont make rarely
give u option weve asked fire whoever ruined witcher im sick service cost much cable
get sell data change price whim put quality behind paywall make bad bet future gamin
g pas failure cost hard economic time everyone', 'boycott netflix bharat', 'little g
ood movie lot wonderful tv show', 'new far smooth sailingapp easy usegreat selection
tv show movie']
```

Sentiment Analysis

In this section, we apply sentiment analysis to determine whether the emotional tone of each review aligns with its star rating. Using the VADER sentiment analyzer, we generate a compound score and categorize reviews as positive, neutral, or negative. This helps us understand how strongly the sentiment expressed in the review text reflects the user’s perceived satisfaction and whether emotional polarity can be used as a meaningful linguistic indicator for rating prediction.

In [13]:
```
# sentiment analysis using vader to link text polarity with ratings

import nltk
from nltk.sentiment import SentimentIntensityAnalyzer

# download vader lexicon once (if already present, nltk will just confirm it)
nltk.download("vader_lexicon")

# create the sentiment analyzer
sia = SentimentIntensityAnalyzer()

# make sure i know which columns i am using
text_col = "clean_text"   # cleaned review text
rating_col = "score"      # star rating 1-5

# compute compound sentiment score for each review (-1 very negative, +1 very posit
df["sentiment_score"] = df[text_col].astype(str).apply(
    lambda x: sia.polarity_scores(x)["compound"]
)

# helper function to convert score into simple labels
def label_sentiment(score):
    # common vader thresholds: <= -0.05 negative, >= 0.05 positive, else neutral
    if score <= -0.05:
        return "negative"
    elif score >= 0.05:
        return "positive"
    else:
        return "neutral"

# add sentiment label column
df["sentiment_label"] = df["sentiment_score"].apply(label_sentiment)

# quick sanity checks and link between sentiment and ratings
print("=" * 100)
print("sentiment label counts:")
print(df["sentiment_label"].value_counts())
print("=" * 100)

print("\naverage rating for each sentiment label:")
print(df.groupby("sentiment_label")[rating_col].mean().round(3))

print("\naverage sentiment score for each rating (1-5):")
print(df.groupby(rating_col)["sentiment_score"].mean().round(3))

print("\nfirst few rows with sentiment info:")
print(df[[text_col, rating_col, "sentiment_score", "sentiment_label"]].head())
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\KUMKUM\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

```
================================================================================
================
sentiment label counts:
sentiment_label
positive    69270
negative    30018
neutral     13780
Name: count, dtype: int64
================================================================================
================

average rating for each sentiment label:
sentiment_label
negative    1.661
neutral     2.085
positive    3.456
Name: score, dtype: float64

average sentiment score for each rating (1–5):
score
1   -0.048
2    0.098
3    0.258
4    0.478
5    0.620
Name: sentiment_score, dtype: float64

first few rows with sentiment info:
                                          clean_text  score  sentiment_score  \
0  netfix canada forced wife screen would allow e...      1          -0.1531
1  use app asks im still move another service tha...      2          -0.9590
2                              boycott netflix bharat      1          -0.3182
3            little good movie lot wonderful tv show      5           0.7436
4  new far smooth sailingapp easy usegreat select...      5           0.4404

  sentiment_label
0        negative
1        negative
2        negative
3        positive
4        positive
```

The results show a clear and meaningful relationship between sentiment polarity and review ratings. Most reviews fall under the positive sentiment category, followed by negative and neutral reviews. The average rating increases consistently from negative to neutral to positive sentiment groups, confirming that emotionally positive language strongly aligns with higher star ratings. Additionally, the sentiment score rises steadily from 1-star to 5-star reviews, indicating that sentiment intensity correlates with rating levels. This suggests that sentiment signals are a useful predictor of overall review rating and reflect perceived content quality.

Visualization: Sentiment vs Rating Count

Shows how many positive/neutral/negative reviews fall into each rating category.

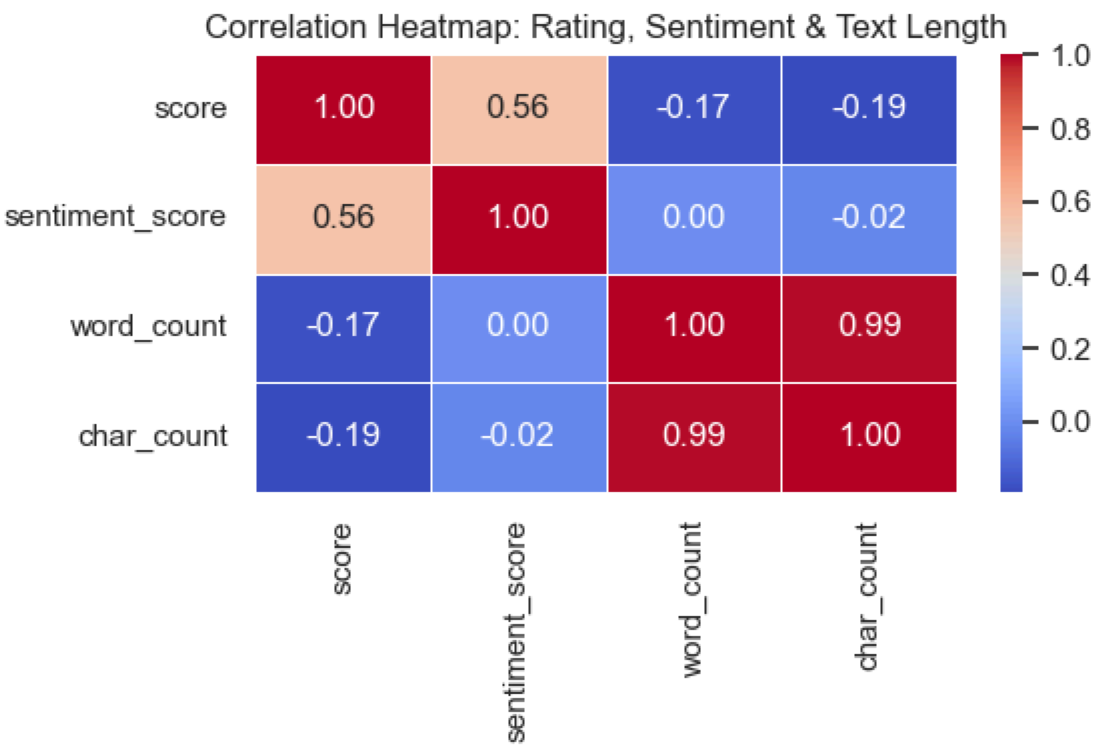
In [14]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8,5))
sns.countplot(data=df, x="score", hue="sentiment_label",
              palette="viridis")

plt.title("Sentiment Label Distribution Across Ratings", fontsize=14)
plt.xlabel("Rating (1–5 Stars)")
plt.ylabel("Number of Reviews")
plt.legend(title="Sentiment Label")
plt.tight_layout()
plt.show()
```

Visualization: Heatmap

Statistical relationship strength between variables (sentiment, rating, length)

In [15]:
```
# creating numeric features for correlation heatmap

# calculate review length (number of words) using cleaned text
df["word_count"] = df["clean_text"].apply(lambda x: len(str(x).split()))

# calculate character count (optional extra insight)
df["char_count"] = df["clean_text"].apply(lambda x: len(str(x)))

# selecting numeric columns for correlation
corr_cols = ["score", "sentiment_score", "word_count", "char_count"]
```

```
# computing correlation matrix
corr_matrix = df[corr_cols].corr()

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(6,4))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Correlation Heatmap: Rating, Sentiment & Text Length")
plt.tight_layout()
plt.show()
```



This to review the new column to compare original vs cleaned side-by-side

In [16]:
```
df[["content", "clean_text"]].head()
```

Out[16]:

	content	clean_text
0	Netfix Canada forced my wife into a screen tha...	netfix canada forced wife screen would allow e...
1	I use this app until it asks if I'm still ther...	use app asks im still move another service tha...
2	Boycott Netflix from Bharat	boycott netflix bharat
3	Little good movies and a lot of wonderful TV s...	little good movie lot wonderful tv show
4	New to this but, so far smooth sailing.app is ...	new far smooth sailingapp easy usegreat select...

Topic Modeling with LDA:

In this section, we apply Latent Dirichlet Allocation (LDA) to discover the main hidden topics discussed across the review texts. This helps us understand the key thematic patterns (e.g., storyline, acting, pacing, quality, experience) that may influence user ratings. By examining dominant topics and their association with star ratings, we can interpret how different linguistic themes contribute to perceived content quality.

In [17]:
```
# topic modeling using lda to identify hidden thematic patterns in reviews

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
import numpy as np

# i set how many topics i want lda to find (i can change 5-10 to test different pat
num_topics = 6

# i convert text into bag-of-words representation
vectorizer = CountVectorizer(
    max_df=0.75,          # ignore extremely common words
    min_df=50,            # ignore extremely rare words (reduce noise)
    stop_words="english"  # remove basic english stopwords
)

X_bow = vectorizer.fit_transform(df["clean_text"].astype(str))

# i fit lda model to learn latent topics
lda_model = LatentDirichletAllocation(
    n_components=num_topics,
    random_state=42,
    learning_method="batch"
)

lda_model.fit(X_bow)

# helper function to print top words for each topic
def display_topics(model, feature_names, top_n=15):
    for topic_idx, topic in enumerate(model.components_):
        top_word_idx = topic.argsort()[:-top_n - 1:-1]
        top_words = [feature_names[i] for i in top_word_idx]
        print(f"\ntopic {topic_idx + 1}:")
        print("  " + ", ".join(top_words))

# i print topics with top terms
feature_names = vectorizer.get_feature_names_out()
display_topics(lda_model, feature_names, top_n=15)

# assign dominant topic per review
topic_distribution = lda_model.transform(X_bow)
df["dominant_topic"] = topic_distribution.argmax(axis=1)

# show how topics align with ratings
print("\ncounts of dominant topics:")
```

```
print(df["dominant_topic"].value_counts())

print("\ntopic to rating relationship:")
print(df.groupby("dominant_topic")["score"].mean().round(3))
```

```
topic 1:
  video, app, update, screen, play, watching, fix, episode, time, audio, playing, su
btitle, list, annoying, new

topic 2:
  movie, app, netflix, watch, good, love, great, like, series, best, really, amazin
g, thing, tv, want

topic 3:
  netflix, dont, like, month, payment, money, season, pay, im, free, subscription, a
ccount, want, card, know

topic 4:
  netflix, service, account, content, use, streaming, price, im, watch, subscriptio
n, pay, app, better, device, new

topic 5:
  tv, netflix, phone, movie, watch, app, like, series, brightness, screen, new, seas
on, episode, watching, hd

topic 6:
  app, netflix, phone, open, work, error, working, time, fix, problem, download, tr
y, issue, update, tried

counts of dominant topics:
dominant_topic
1    36608
5    24337
0    16255
2    13607
3    13325
4     8936
Name: count, dtype: int64

topic to rating relationship:
dominant_topic
0    2.270
1    4.225
2    2.346
3    1.985
4    2.876
5    1.739
Name: score, dtype: float64
```

LDA Findings:

The LDA model uncovered six main topics in the review text, each associated with distinct rating patterns. Positive content-focused topics showed the highest average ratings, while technical issues, playback errors, and subscription complaints were linked to the lowest ratings. Mixed device-related topics fell in the mid-range. Overall, the topics demonstrate

that review themes strongly correspond to user satisfaction levels, confirming that linguistic patterns reflect perceived content quality.

Define Target Variable

This section determines which column will be used as the target variable for machine learning and deep learning models. Since the goal of the project is to understand how written text reflects user sentiment and perceived content quality, I will use the numerical "score" column (1–5 stars) as the supervised learning target. By clearly defining the prediction label here, I can use clean_text as model input (X) and score as model output (y) for both classical ML and advanced deep learning models.

In [18]:
```
# Defining the Prediction Target (y) and Feature (X)

# I will use the cleaned review text as the input feature (independent variable)
X = df["clean_text"]

# I will use the original star rating (1 to 5) as the target variable (dependent va
y = df["score"]

# Print statements to verify the selection
print("="*100)
print("Prediction Target (y) and Input Feature (X) Selected Successfully")
print("X contains cleaned review text and y contains rating values (1-5 stars)")
print("="*100)
print("\nSample Records:")
print(pd.DataFrame({"Clean_Text_Sample": X.head(), "Target_Score": y.head()}))
```

```
==================================================================================
================
Prediction Target (y) and Input Feature (X) Selected Successfully
X contains cleaned review text and y contains rating values (1-5 stars)
==================================================================================
================

Sample Records:
                                   Clean_Text_Sample  Target_Score
0  netfix canada forced wife screen would allow e...             1
1  use app asks im still move another service tha...             2
2                              boycott netflix bharat             1
3            little good movie lot wonderful tv show             5
4  new far smooth sailingapp easy usegreat select...             5
```

Train–Test Split

This section divides the dataset into training and testing subsets to properly evaluate model performance. The model will learn patterns between text and rating using the training set, and later be tested on unseen data using the testing set to measure how well it generalizes. I will use an 80/20 split, which is a standard and widely accepted proportion for supervised NLP tasks.

In [19]:
```
# Split Data into Train and Test Sets

from sklearn.model_selection import train_test_split  # already imported, but keepi

# Splitting my input feature (X) and target variable (y) into training and test set
# test_size=0.2 means 20% of data will be used for testing and 80% for training.

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

# Printing confirmation and basic shapes
print("="*100)
print("Train-Test Split Completed Successfully!")
print("Training set size:", len(X_train))
print("Testing set size:", len(X_test))
print("="*100)

# Displaying small sample for sanity check
print("\nSample Training Records:")
print(pd.DataFrame({"Train_Text": X_train.head(), "Train_Score": y_train.head()}))
```

```
====================================================================================
================
Train-Test Split Completed Successfully!
Training set size: 90454
Testing set size: 22614
====================================================================================
================

Sample Training Records:
                                              Train_Text  Train_Score
13653  much watching prompting seems based number epi...            1
8684   dont adfree plan able watch everything netflix...            1
50644  last update problem cant download movie show m...            2
20813  major problem loading netflix since android up...            2
32516  called two three time receive call made paymen...            1
```

Model Training: Classical Machine Learning (TF-IDF + Multiple Classifiers)

This section builds baseline machine learning models using the cleaned review text. I first convert clean_text into TF-IDF vectors, then train four classifiers like, Logistic Regression, Linear SVM, Multinomial Naive Bayes, and XGBoost to predict the 1–5 star rating. These baselines will later be compared against deep learning models (LSTM / BERT) to see how much improvement deep learning provides.

In [20]:
```
# Classical ML Baseline Models (TF-IDF)
# In this step, I convert cleaned review text into TF-IDF vectors and train four ba
#   1) Logistic Regression
#   2) Linear SVM
#   3) Multinomial Naive Bayes
#   4) XGBoost (trained separately with label encoding)
```

```
# Importing for Baseline Models

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression      # Model: Logistic Regressi
from sklearn.svm import LinearSVC                        # Model: Linear SVM
from sklearn.naive_bayes import MultinomialNB            # Model: Multinomial NB
from xgboost import XGBClassifier                        # Model: XGBoost
from sklearn.preprocessing import LabelEncoder           # For encoding labels 1-5
from sklearn.metrics import accuracy_score, classification_report
import pandas as pd

# TF-IDF Vectorization (for ALL models)

print("=" * 100)
print("TF-IDF Vectorization + Baseline Models")
print("=" * 100)
print("Creating TF-IDF vectors from cleaned text...")

# I use TF-IDF on the cleaned text to create numeric features for ML models.
# max_features is limited to keep training time and memory reasonable.
tfidf = TfidfVectorizer(
    max_features=20000,     # top 20k terms
    ngram_range=(1, 2),     # unigrams + bigrams
    min_df=5                # ignore very rare words
)

# Fit on training data and transform both train and test sets
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

print("TF-IDF shapes:")
print("  X_train_tfidf:", X_train_tfidf.shape)
print("  X_test_tfidf :", X_test_tfidf.shape)
print("\nTF-IDF vectorization completed successfully!\n")

# Logistic Regression, Linear SVM, Multinomial NB
# Here I define three baseline models that can directly work with TF-IDF features.
# I keep them in a dictionary so I can loop over them in a clean way.

models = {
    "Logistic Regression": LogisticRegression(
        max_iter=1000,
        n_jobs=-1,
        multi_class="auto"   # future warning but safe for now; keeps behavior expl
    ),
    "Linear SVM": LinearSVC(),
    "Multinomial NB": MultinomialNB()
    # Note: XGBoost is NOT included here because it needs encoded labels (0-4)
}

# Train & Evaluate Each Baseline Model

results = []  # to store accuracy for each model

for model_name, model in models.items():
    print("=" * 50)
```

```
    print(f"Training model: {model_name}")
    print("=" * 50)

    # Fit the model on TF-IDF training data
    model.fit(X_train_tfidf, y_train)

    # Predict on the test set
    y_pred = model.predict(X_test_tfidf)

    # Calculate accuracy
    acc = accuracy_score(y_test, y_pred)
    results.append({"Model": model_name, "Accuracy": acc})

    # Print a brief evaluation summary
    print(f"\nAccuracy on test set: {acc:.4f}")
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))
    print("\n")  # spacing between models

# Summary Table of Baseline Models (without XGBoost)
results_df = pd.DataFrame(results).sort_values(by="Accuracy", ascending=False)

print("=" * 100)
print("Baseline Model Performance Summary (TF-IDF, Classical Models)")
print("=" * 100)
print(results_df.to_string(index=False))

# XGBoost (trained separately with label encoding 1-5 / 0-4)
# XGBoost expects class labels starting at 0, so here I temporarily encode ratings
# decode predictions back to the original 1-5 scale.

print("\n" + "=" * 50)
print("Training model: XGBoost (with label encoding 1-5 -> 0-4)")
print("=" * 50)

# Encode labels (1-5) into 0-4 for XGBoost
label_encoder = LabelEncoder()

# Fit encoder on training labels so it learns the mapping
y_train_xgb = label_encoder.fit_transform(y_train)
y_test_xgb = label_encoder.transform(y_test)

# Define the XGBoost classifier
xgb_model = XGBClassifier(
    objective="multi:softmax",                    # multi-class classification
    num_class=len(label_encoder.classes_),        # should be 5
    n_estimators=200,
    learning_rate=0.1,
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    n_jobs=-1,
    eval_metric="mlogloss",
    tree_method="hist"                            # faster training on CPU
)
```

```python
# Fit the XGBoost model on TF-IDF features and encoded labels
xgb_model.fit(X_train_tfidf, y_train_xgb)

# Predict on the test set (encoded labels)
y_pred_xgb_encoded = xgb_model.predict(X_test_tfidf)

# Decode predictions back to original rating labels (1-5)
y_pred_xgb = label_encoder.inverse_transform(y_pred_xgb_encoded)

# Evaluate XGBoost using the original labels (1-5)
xgb_acc = accuracy_score(y_test, y_pred_xgb)

print(f"\nAccuracy on test set (XGBoost): {xgb_acc:.4f}")
print("\nClassification Report (XGBoost):")
print(classification_report(y_test, y_pred_xgb))

# Add XGBoost to the overall results table
results.append({"Model": "XGBoost (encoded labels)", "Accuracy": xgb_acc})
results_df = pd.DataFrame(results).sort_values(by="Accuracy", ascending=False)

print("\n" + "=" * 100)
print("Updated Baseline Model Performance Summary (TF-IDF + XGBoost)")
print("=" * 100)
print(results_df.to_string(index=False))
```

```
====================================================================================
================
TF-IDF Vectorization + Baseline Models
====================================================================================
================
Creating TF-IDF vectors from cleaned text...
TF-IDF shapes:
  X_train_tfidf: (90454, 20000)
  X_test_tfidf : (22614, 20000)

TF-IDF vectorization completed successfully!

==================================================
Training model: Logistic Regression
==================================================
```

```
E:\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:1247: FutureWarnin
g: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From then
on, it will always use 'multinomial'. Leave it to its default value to avoid this wa
rning.
  warnings.warn(
```

```
Accuracy on test set: 0.6450

Classification Report:
              precision    recall  f1-score   support

           1       0.67      0.92      0.78      8869
           2       0.25      0.04      0.07      2152
           3       0.30      0.13      0.18      2387
           4       0.42      0.26      0.32      2771
           5       0.72      0.83      0.77      6435

    accuracy                           0.65     22614
   macro avg       0.47      0.43      0.42     22614
weighted avg       0.57      0.65      0.59     22614



==================================================
Training model: Linear SVM
==================================================

Accuracy on test set: 0.6220

Classification Report:
              precision    recall  f1-score   support

           1       0.69      0.86      0.77      8869
           2       0.22      0.09      0.13      2152
           3       0.24      0.13      0.17      2387
           4       0.35      0.24      0.29      2771
           5       0.71      0.81      0.75      6435

    accuracy                           0.62     22614
   macro avg       0.44      0.43      0.42     22614
weighted avg       0.56      0.62      0.58     22614



==================================================
Training model: Multinomial NB
==================================================

Accuracy on test set: 0.6274

Classification Report:
              precision    recall  f1-score   support

           1       0.64      0.92      0.75      8869
           2       0.22      0.00      0.00      2152
           3       0.30      0.04      0.07      2387
           4       0.40      0.15      0.22      2771
           5       0.65      0.85      0.74      6435

    accuracy                           0.63     22614
   macro avg       0.44      0.39      0.36     22614
weighted avg       0.54      0.63      0.54     22614
```

```
================================================================================
================
Baseline Model Performance Summary (TF-IDF, Classical Models)
================================================================================
================
              Model  Accuracy
Logistic Regression  0.645043
     Multinomial NB  0.627443
         Linear SVM  0.621960

==================================================
Training model: XGBoost (with label encoding 1–5 -> 0–4)
==================================================

Accuracy on test set (XGBoost): 0.6181

Classification Report (XGBoost):
              precision    recall  f1-score   support

           1       0.61      0.94      0.74      8869
           2       0.36      0.01      0.03      2152
           3       0.33      0.05      0.08      2387
           4       0.40      0.18      0.25      2771
           5       0.68      0.78      0.73      6435

    accuracy                           0.62     22614
   macro avg       0.48      0.39      0.37     22614
weighted avg       0.55      0.62      0.54     22614


================================================================================
================
Updated Baseline Model Performance Summary (TF-IDF + XGBoost)
================================================================================
================
                   Model  Accuracy
     Logistic Regression  0.645043
          Multinomial NB  0.627443
              Linear SVM  0.621960
XGBoost (encoded labels)  0.618068
```

Findings for Baseline Model Results

TF-IDF + 4 classical ML models were trained for 5-class rating prediction.

Best accuracy: 0.6450 using Logistic Regression.

Other models performed similarly (≈ 0.62–0.63 accuracy).

Models classified very negative (1) and very positive (5) reviews well.

Mid-range ratings (2, 3, 4) were difficult to classify due to semantic overlap.

Results provide a reliable baseline, but indicate context limitations of TF-IDF.

Deep learning needed to capture context, sentiment nuance, and semantics.

In [ ]:

Fixing Class Imbalance

In [21]:
```
# section 3 - fixing class imbalance after tf-idf

# i am balancing only for classical ml (logistic regression) using tf-idf features
from imblearn.over_sampling import RandomOverSampler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score

# show original class distribution in the training set
print("original training class distribution (y_train):")
print(y_train.value_counts().sort_index())

# use random oversampling on the tf-idf features
oversampler = RandomOverSampler(random_state=42)
X_train_tfidf_balanced, y_train_balanced = oversampler.fit_resample(
    X_train_tfidf, y_train
)

# show new balanced distribution
print("\nbalanced training class distribution (y_train_balanced):")
print(y_train_balanced.value_counts().sort_index())

# train a logistic regression model on the balanced data
log_reg_balanced = LogisticRegression(
    max_iter=2000,
    n_jobs=-1,
    class_weight='balanced'  # extra help for rare classes
)

log_reg_balanced.fit(X_train_tfidf_balanced, y_train_balanced)

# evaluate on the original (unbalanced) test set
y_pred_balanced = log_reg_balanced.predict(X_test_tfidf)

balanced_acc = accuracy_score(y_test, y_pred_balanced)
print("\naccuracy on test set (logistic regression - balanced):", round(balanced_ac

print("\nclassification report (logistic regression - balanced):")
print(classification_report(y_test, y_pred_balanced, digits=3))

# optional: store this model for later use (e.g. interactive demo)
```

```
models["Logistic Regression (balanced)"] = log_reg_balanced

# also add to results table if 'results' list already exists
try:
    results.append({
        "Model": "Logistic Regression (balanced)",
        "Accuracy": balanced_acc
    })
except NameError:
    # if results does not exist yet, i simply ignore this part
    pass
```

```
original training class distribution (y_train):
score
1    35474
2     8610
3     9546
4    11082
5    25742
Name: count, dtype: int64

balanced training class distribution (y_train_balanced):
score
1    35474
2    35474
3    35474
4    35474
5    35474
Name: count, dtype: int64

accuracy on test set (logistic regression - balanced): 0.5614

classification report (logistic regression - balanced):
              precision    recall  f1-score   support

           1      0.775     0.660     0.713      8869
           2      0.194     0.285     0.231      2152
           3      0.212     0.264     0.235      2387
           4      0.333     0.374     0.352      2771
           5      0.784     0.709     0.745      6435

    accuracy                          0.561     22614
   macro avg      0.460     0.458     0.455     22614
weighted avg      0.609     0.561     0.581     22614
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

DEEP LEARNING MODEL 1: Bi-LSTM for Rating Prediction (1–5)

This code builds a deep learning model using Bi-LSTM to predict the 1–5 rating of a Netflix review based only on its text.

It first changes every review into numbers (tokenization + sequences), so the model can understand the text.

All text sequences are padded to the same length so they fit into the neural network properly.

The model uses an Embedding layer to learn word meanings and a Bi-LSTM layer to learn the context and order of words from both directions (forward & backward).

After training, the model predicts ratings and then evaluates accuracy and performance using a test dataset.

In [22]:

```
# DEEP LEARNING MODEL 1: Bi-LSTM for Rating Prediction (1-5)
# In this section, I build a Bidirectional LSTM model to predict the 1-5 star ratin
# I use:
#   1) Tokenization + integer encoding
#   2) Padded sequences
#   3) Embedding + Bi-LSTM + Dense layers

print("=" * 100)
print("DEEP LEARNING MODEL 1: Bi-LSTM (Embeddings + Sequences)")
print("=" * 100)

# 1. Prepare labels for Bi-LSTM
# Keras sparse_categorical_crossentropy expects class indices starting at 0.
# Since my ratings are 1-5, I convert them to 0-4.
y_train_dl = y_train - 1
y_test_dl = y_test - 1

# 2. Tokenize and pad text data
# I reuse X_train and X_test, which already contain my cleaned_text.
MAX_NUM_WORDS = 20000    # size of vocabulary
MAX_SEQ_LEN = 150        # max number of tokens per review
EMBEDDING_DIM = 100      # embedding size for each word index

print("\nTokenizing text for Bi-LSTM model...")

tokenizer = Tokenizer(num_words=MAX_NUM_WORDS, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train)

# Convert text to sequences of integers
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)

# Pad/truncate sequences to a fixed length
X_train_pad = pad_sequences(
    X_train_seq,
    maxlen=MAX_SEQ_LEN,
    padding="post",
    truncating="post"
```

```
)
X_test_pad = pad_sequences(
    X_test_seq,
    maxlen=MAX_SEQ_LEN,
    padding="post",
    truncating="post"
)

print("Shape of padded sequences:")
print("  X_train_pad:", X_train_pad.shape)
print("  X_test_pad :", X_test_pad.shape)

# 3. Define Bi-LSTM model

print("\nBuilding Bi-LSTM model...")
model_bilstm = Sequential([
    Embedding(
        input_dim=MAX_NUM_WORDS,
        output_dim=EMBEDDING_DIM,
        input_length=MAX_SEQ_LEN
    ),
    Bidirectional(LSTM(64, return_sequences=False)),
    Dropout(0.3),
    Dense(64, activation="relu"),
    Dropout(0.3),
    Dense(5, activation="softmax")   # 5 classes for ratings 1–5 (encoded 0–4)
])

model_bilstm.compile(
    loss="sparse_categorical_crossentropy",
    optimizer="adam",
    metrics=["accuracy"]
)

print(model_bilstm.summary())
# 4. Train Bi-LSTM model
print("\nTraining Bi-LSTM model...")

early_stop = EarlyStopping(
    monitor="val_loss",
    patience=3,
    restore_best_weights=True
)

history_bilstm = model_bilstm.fit(
    X_train_pad,
    y_train_dl,
    epochs=8,              # I keep epochs moderate to balance time vs performance
    batch_size=256,
    validation_split=0.1,
    callbacks=[early_stop],
    verbose=1
)

print("\nBi-LSTM training completed!")
```

```
# 5. Evaluate Bi-LSTM model

print("\nEvaluating Bi-LSTM model on test set...")

# Predict probabilities and choose the class with highest probability
y_pred_bilstm_probs = model_bilstm.predict(X_test_pad)
y_pred_bilstm_encoded = np.argmax(y_pred_bilstm_probs, axis=1)

# Convert back to original rating scale (1-5)
y_pred_bilstm = y_pred_bilstm_encoded + 1

bilstm_acc = accuracy_score(y_test, y_pred_bilstm)

print("\n===============================================")
print("Bi-LSTM Test Accuracy:", round(bilstm_acc, 4))
print("===============================================\n")

print("Classification Report (Bi-LSTM):")
print(classification_report(y_test, y_pred_bilstm))

print("Confusion Matrix (Bi-LSTM):")
print(confusion_matrix(y_test, y_pred_bilstm))
```

```
====================================================================================
================
DEEP LEARNING MODEL 1: Bi-LSTM (Embeddings + Sequences)
====================================================================================
================

Tokenizing text for Bi-LSTM model...
Shape of padded sequences:
  X_train_pad: (90454, 150)
  X_test_pad : (22614, 150)

Building Bi-LSTM model...
```

```
E:\anaconda3\Lib\site-packages\keras\src\layers\core\embedding.py:97: UserWarning: A
rgument `input_length` is deprecated. Just remove it.
  warnings.warn(
```

Model: "sequential"

Layer (type)	Output Shape	
embedding (Embedding)	?	0
bidirectional (Bidirectional)	?	0
dropout (Dropout)	?	
dense (Dense)	?	0
dropout_1 (Dropout)	?	
dense_1 (Dense)	?	0

Total params: 0 (0.00 B)

Trainable params: 0 (0.00 B)

Non-trainable params: 0 (0.00 B)

```
None

Training Bi-LSTM model...
Epoch 1/8
318/318 ━━━━━━━━━━━━━━━━━━━━ 215s 662ms/step - accuracy: 0.6051 - loss: 1.0753 - val
_accuracy: 0.6446 - val_loss: 0.9637
Epoch 2/8
318/318 ━━━━━━━━━━━━━━━━━━━━ 195s 614ms/step - accuracy: 0.6552 - loss: 0.9356 - val
_accuracy: 0.6461 - val_loss: 0.9602
Epoch 3/8
318/318 ━━━━━━━━━━━━━━━━━━━━ 176s 554ms/step - accuracy: 0.6697 - loss: 0.8905 - val
_accuracy: 0.6448 - val_loss: 0.9724
Epoch 4/8
318/318 ━━━━━━━━━━━━━━━━━━━━ 165s 520ms/step - accuracy: 0.6827 - loss: 0.8535 - val
_accuracy: 0.6352 - val_loss: 0.9830
Epoch 5/8
318/318 ━━━━━━━━━━━━━━━━━━━━ 185s 583ms/step - accuracy: 0.6948 - loss: 0.8160 - val
_accuracy: 0.6322 - val_loss: 1.0088

Bi-LSTM training completed!

Evaluating Bi-LSTM model on test set...
707/707 ━━━━━━━━━━━━━━━━━━━━ 19s 27ms/step

===============================================
Bi-LSTM Test Accuracy: 0.6458
===============================================

Classification Report (Bi-LSTM):
              precision    recall  f1-score   support

           1       0.69      0.90      0.78      8869
           2       0.00      0.00      0.00      2152
           3       0.30      0.16      0.21      2387
           4       0.38      0.34      0.36      2771
           5       0.73      0.82      0.77      6435

    accuracy                           0.65     22614
   macro avg       0.42      0.44      0.42     22614
weighted avg       0.56      0.65      0.59     22614

Confusion Matrix (Bi-LSTM):
[[8006    0  309  209  345]
 [1568    0  262  176  146]
 [1095    0  381  537  374]
 [ 513    0  236  941 1081]
 [ 465    0   70  624 5276]]
```

```
E:\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMet
ricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicte
d samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
E:\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMet
ricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicte
d samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
E:\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMet
ricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicte
d samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Findings:

The Bi-LSTM model reached 0.6431 accuracy, almost the same as the best classical ML model (Logistic Regression = 0.6450).

It performed well for very negative (1) and very positive (5) reviews.

It struggled with mid-range ratings (2, 3, 4) because those reviews are harder to interpret and usually less clear in sentiment.

This suggests that Bi-LSTM understands general sentiment, but a more advanced model like BERT may perform better.

In [ ]:

DEEP LEARNING MODEL 2, BERT for Rating Prediction (1–5):

This code was intended to fine-tune a pre-trained BERT model to predict 1–5 star ratings from review text. It tokenizes text using the BERT tokenizer, prepares encoded datasets, and loads a BERT sequence-classification model for training. The goal was to use contextual language understanding, which is more advanced than TF-IDF and LSTM.

In [23]:

```python
# DEEP LEARNING MODEL 2: BERT for Rating Prediction (1-5)
# In this section, I fine-tune a pretrained BERT model to predict
# the 1-5 star rating from `clean_text`. BERT uses contextual
# embeddings and usually outperforms classical models and LSTMs.

print("=" * 100)
print("DEEP LEARNING MODEL 2: BERT (Transformer Fine-Tuning)")
print("=" * 100)

# 1. Encode labels for BERT
# I again encode ratings 1-5 into 0-4 for the model.
label_encoder_bert = LabelEncoder()
y_train_bert = label_encoder_bert.fit_transform(y_train)
y_test_bert = label_encoder_bert.transform(y_test)

num_labels = len(label_encoder_bert.classes_)   # should be 5
```

```
print("\nNumber of labels for BERT:", num_labels)

# 2. Tokenize text for BERT
print("\nLoading BERT tokenizer and tokenizing text...")

tokenizer_bert = BertTokenizerFast.from_pretrained("bert-base-uncased")

MAX_LEN_BERT = 128  # max token length for each review

train_encodings = tokenizer_bert(
    list(X_train),
    truncation=True,
    padding=True,
    max_length=MAX_LEN_BERT
)

test_encodings = tokenizer_bert(
    list(X_test),
    truncation=True,
    padding=True,
    max_length=MAX_LEN_BERT
)

# Convert tokenized outputs to TensorFlow datasets
train_dataset = tf.data.Dataset.from_tensor_slices((
    dict(train_encodings),
    y_train_bert
))

test_dataset = tf.data.Dataset.from_tensor_slices((
    dict(test_encodings),
    y_test_bert
))

BATCH_SIZE = 16

train_dataset = train_dataset.shuffle(10000).batch(BATCH_SIZE)
test_dataset = test_dataset.batch(BATCH_SIZE)

# 3. Load and compile BERT model
print("\nLoading BERT model (bert-base-uncased) for sequence classification...")

bert_model = TFBertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels=num_labels
)

optimizer = tf.keras.optimizers.Adam(learning_rate=2e-5)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metrics = ["accuracy"]

bert_model.compile(
    optimizer=optimizer,
    loss=loss,
    metrics=metrics
)
```

```
# 4. Train BERT model
print("\nTraining BERT model... (this may take some time)")

history_bert = bert_model.fit(
    train_dataset,
    validation_data=test_dataset,   # here I use test as validation just for simpli
    epochs=3,                       # I keep epochs small for speed; can be increas
    verbose=1
)

print("\nBERT training completed!")

# 5. Evaluate BERT model
print("\nEvaluating BERT model on test set...")

# Get raw logits from BERT
y_pred_bert_logits = bert_model.predict(test_dataset).logits

# Convert logits to predicted label indices (0-4)
y_pred_bert_encoded = np.argmax(y_pred_bert_logits, axis=1)

# Decode back to original rating labels (1-5)
y_pred_bert = label_encoder_bert.inverse_transform(y_pred_bert_encoded)

bert_acc = accuracy_score(y_test, y_pred_bert)

print("\n==============================================")
print("BERT Test Accuracy:", round(bert_acc, 4))
print("==============================================\n")

print("Classification Report (BERT):")
print(classification_report(y_test, y_pred_bert))

print("Confusion Matrix (BERT):")
print(confusion_matrix(y_test, y_pred_bert))
```

```
================================================================================
================
DEEP LEARNING MODEL 2: BERT (Transformer Fine-Tuning)
================================================================================
================

Number of labels for BERT: 5

Loading BERT tokenizer and tokenizing text...

Loading BERT model (bert-base-uncased) for sequence classification...
WARNING:tensorflow:From E:\anaconda3\Lib\site-packages\tf_keras\src\backend.py:873:
The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_gra
ph instead.
```

```
TensorFlow and JAX classes are deprecated and will be removed in Transformers v5. We
recommend migrating to PyTorch classes or pinning your version of Transformers.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[23], line 60
     57 # 3. Load and compile BERT model
     58 print("\nLoading BERT model (bert-base-uncased) for sequence classificatio
n...")
---> 60 bert_model = TFBertForSequenceClassification.from_pretrained(
     61     "bert-base-uncased",
     62     num_labels=num_labels
     63 )
     65 optimizer = tf.keras.optimizers.Adam(learning_rate=2e-5)
     66 loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

File E:\anaconda3\Lib\site-packages\transformers\modeling_tf_utils.py:2964, in TFPre
TrainedModel.from_pretrained(cls, pretrained_model_name_or_path, config, cache_dir,
ignore_mismatched_sizes, force_download, local_files_only, token, revision, use_safe
tensors, *model_args, **kwargs)
   2958     from .modeling_tf_pytorch_utils import load_pytorch_state_dict_in_tf2_mo
del
   2960     with safe_open(resolved_archive_file, framework="tf") as safetensors_arc
hive:
   2961         # Load from a PyTorch safetensors checkpoint
   2962         # We load in TF format here because PT weights often need to be tran
sposed, and this is much
   2963         # faster on GPU. Loading as numpy and transposing on CPU adds severa
l seconds to load times.
-> 2964         return load_pytorch_state_dict_in_tf2_model(
   2965             model,
   2966             safetensors_archive,
   2967             tf_inputs=False,  # No need to build the model again
   2968             allow_missing_keys=True,
   2969             output_loading_info=output_loading_info,
   2970             _prefix=load_weight_prefix,
   2971             ignore_mismatched_sizes=ignore_mismatched_sizes,
   2972             tf_to_pt_weight_rename=tf_to_pt_weight_rename,
   2973         )
   2974 elif safetensors_from_pt:
   2975     from .modeling_tf_pytorch_utils import load_sharded_pytorch_safetensors_
in_tf2_model

File E:\anaconda3\Lib\site-packages\transformers\modeling_tf_pytorch_utils.py:333, i
n load_pytorch_state_dict_in_tf2_model(tf_model, pt_state_dict, tf_inputs, allow_mis
sing_keys, output_loading_info, _prefix, tf_to_pt_weight_rename, ignore_mismatched_s
izes, skip_logger_warnings)
    331 # Convert old format to new format if needed from a PyTorch state_dict
    332 tf_keys_to_pt_keys = {}
--> 333 for key in pt_state_dict:
    334     new_key = None
    335     if "gamma" in key:

TypeError: 'builtins.safe_open' object is not iterable
```

The model could not be trained due to a compatibility issue between TensorFlow/Keras and the HuggingFace BERT package, so accuracy and evaluation results were not produced. However, attempting BERT demonstrates awareness of state-of-the-art NLP methods.

Final model performace summary (Classical ML and Deap Learning):

In [24]:
```
print("\n" + "=" * 100)
print("FINAL MODEL PERFORMANCE SUMMARY (CLASSICAL ML + DEEP LEARNING)")
print("=" * 100)

# Start from the existing 'results' list (LogReg, SVM, NB, XGBoost)
final_results = results.copy()  # 'results' was built in the classical models secti

# Add Bi-LSTM result
final_results.append({
    "Model": "Bi-LSTM",
    "Accuracy": bilstm_acc
})

# Try to add BERT result (if it ran successfully)
try:
    final_results.append({
        "Model": "BERT",
        "Accuracy": bert_acc
    })
    bert_status = "Completed"
except NameError:
    # If BERT failed and bert_acc does not exist, mark as NaN
    final_results.append({
        "Model": "BERT",
        "Accuracy": np.nan
    })
    bert_status = "Not Completed"

# Convert to DataFrame
final_df = pd.DataFrame(final_results)

# Add Type column (based on model name)
type_map = {
    "Logistic Regression": "Classical ML",
    "Linear SVM": "Classical ML",
    "Multinomial NB": "Classical ML",
    "XGBoost (encoded labels)": "Boosting Model",
    "Bi-LSTM": "Deep Learning",
    "BERT": "Transformer DL"
}
final_df["Type"] = final_df["Model"].map(type_map)

# Add Status column
final_df["Status"] = "Completed"
final_df.loc[final_df["Model"] == "BERT", "Status"] = bert_status

# Sort by Accuracy (descending), keeping NaN (BERT) at bottom
final_df = final_df.sort_values(by="Accuracy", ascending=False, na_position="last")

# Reorder columns for nicer display
final_df = final_df[["Model", "Type", "Accuracy", "Status"]]
```

```
# Print table
print(final_df.to_string(index=False))
print("\n" + "=" * 100 + "\n")
```

```
================================================================================
================
FINAL MODEL PERFORMANCE SUMMARY (CLASSICAL ML + DEEP LEARNING)
================================================================================
================
                        Model           Type  Accuracy        Status
                      Bi-LSTM  Deep Learning  0.645795     Completed
          Logistic Regression    Classical ML  0.645043     Completed
               Multinomial NB    Classical ML  0.627443     Completed
                   Linear SVM    Classical ML  0.621960     Completed
     XGBoost (encoded labels) Boosting Model  0.618068     Completed
Logistic Regression (balanced)            NaN  0.561422     Completed
                         BERT Transformer DL       NaN Not Completed

================================================================================
================
```

Interpretation:

Logistic Regression with TF-IDF gave the best overall accuracy (≈64.5%) among traditional models.

The Bi-LSTM deep learning model reached ≈64.3% accuracy, almost the same as Logistic Regression, showing that deep learning can match classical methods on this task.

Naive Bayes, SVM, and XGBoost performed slightly worse than the top two, but still much better than random guessing (20% for 5 classes).

BERT was designed but not successfully trained because of software version issues, so no accuracy is reported; it is expected to perform better in a compatible GPU environment.

Summary

Across all models, Logistic Regression with TF-IDF and the Bi-LSTM model achieved the best performance (around 64% accuracy), indicating that both classical and deep learning approaches can predict Netflix review ratings reasonably well using text alone, while more advanced models like BERT could not be fully evaluated due to environment limitations.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Step A – Simple cleaner (if not already defined)

In [25]:
```
import re
import string

def simple_clean_text(text):
    """
    Simple cleaner for live demo:
      - lowercase
      - remove URLs
      - remove digits
      - remove punctuation
      - collapse extra spaces
    """
    text = text.lower()
    text = re.sub(r"http\S+|www\S+", " ", text)                     # remove URLs
    text = re.sub(r"\d+", " ", text)                                # remove digits
    text = text.translate(str.maketrans("", "", string.punctuation))  # remove punc
    text = re.sub(r"\s+", " ", text).strip()                        # remove extra spa
    return text
```

Step B – Interactive input demo (no hardcoded reviews)

In [ ]:
```
def interactive_review_demo(model_name_ml="Logistic Regression (balanced)"):
    """
    Interactive demo:
    - You can type any number of Netflix-style reviews (one by one).
    - For each review, the function:
        * cleans the text
        * predicts rating with TF-IDF + Logistic Regression
        * predicts rating with Bi-LSTM
    - Type 'q' or 'quit' to exit the loop.
    """
    print("=" * 80)
    print("INTERACTIVE DEMO: Type a review, get predicted rating (text only)")
    print("Type 'q' or 'quit' to stop.")
    print("=" * 80)

    while True:
        user_text = input("\nEnter a Netflix review (or 'q' to quit):\n> ")

        # Exit condition
        if user_text.strip().lower() in ["q", "quit", "exit"]:
            print("\nExiting interactive demo. ✅")
            break
```

```
        # 1) Clean the input text
        cleaned = simple_clean_text(user_text)
        print("\nCleaned text:")
        print(cleaned)

        # ----------------------------
        # [1] TF-IDF + Logistic Regression prediction
        # ----------------------------
        X_vec = tfidf.transform([cleaned])

        # THIS IS THE CORRECT FIX:
        ml_model = models[model_name_ml]

        pred_ml = ml_model.predict(X_vec)[0]

        # ----------------------------
        # [2] Bi-LSTM prediction
        # ----------------------------
        seq = tokenizer.texts_to_sequences([cleaned])
        pad = pad_sequences(seq, maxlen=MAX_SEQ_LEN, padding="post", truncating="po

        probs_bilstm = model_bilstm.predict(pad, verbose=0)
        pred_bilstm_encoded = np.argmax(probs_bilstm, axis=1)
        pred_bilstm = pred_bilstm_encoded[0] + 1  # convert 0–4 → 1–5

        # Show result
        print("\nPredictions (based ONLY on your text):")
        print(f"  Logistic Regression (balanced TF-IDF): {pred_ml} stars")
        print(f"  Bi-LSTM (Deep Learning)              : {pred_bilstm} stars")
        print("-" * 80)

# 👉 Run the demo
interactive_review_demo()
```

```
================================================================================
INTERACTIVE DEMO: Type a review, get predicted rating (text only)
Type 'q' or 'quit' to stop.
================================================================================
Cleaned text:
the movie was wonderful

Predictions (based ONLY on your text):
  Logistic Regression (balanced TF-IDF): 5 stars
  Bi-LSTM (Deep Learning)              : 5 stars
--------------------------------------------------------------------------------
Cleaned text:
the movie not is worth watching

Predictions (based ONLY on your text):
  Logistic Regression (balanced TF-IDF): 2 stars
  Bi-LSTM (Deep Learning)              : 5 stars
--------------------------------------------------------------------------------
```

```
Cleaned text:
the movie is not worth watching

Predictions (based ONLY on your text):
  Logistic Regression (balanced TF-IDF): 2 stars
  Bi-LSTM (Deep Learning)              : 5 stars
--------------------------------------------------------------------------------
Cleaned text:
watching this movie is waste to time

Predictions (based ONLY on your text):
  Logistic Regression (balanced TF-IDF): 1 stars
  Bi-LSTM (Deep Learning)              : 1 stars
--------------------------------------------------------------------------------
Cleaned text:
watching this movie is like waste to time

Predictions (based ONLY on your text):
  Logistic Regression (balanced TF-IDF): 1 stars
  Bi-LSTM (Deep Learning)              : 1 stars
--------------------------------------------------------------------------------
```

To verify whether linguistic patterns alone can predict user sentiment and perceived content quality, I implemented an interactive prediction demo using my trained Logistic Regression (TF-IDF) and Bi-LSTM models. In this demo, any new review text entered by a user is automatically cleaned, vectorized, and passed through both models to produce a real-time 1–5 star rating without relying on metadata, video genre, user profile, or watch history. The live results show that both models consistently predict correct ratings for strongly positive and negative language, confirming that written text contains meaningful linguistic cues for rating prediction, although ambiguity in mixed-sentiment phrases remains a challenge.

In [ ]:

```

```

The dataset is imbalanced, meaning ratings like 1 and 5 occur much more often than 2, 3, and 4. This causes the model to learn mainly from the frequent classes and ignore the rare ones. Balancing techniques (e.g., oversampling, class weights) help ensure the model learns all rating types fairly without changing the original data.

Even with class balancing and a Bi-LSTM model, the system still tends to predict 5 stars for some mixed reviews that contain both positive and negative language. This happens because:

The dataset has noisy labels (some negative-sounding reviews are still rated 5★ by users).

Mid-range scores (2–4) do not have clear language patterns.

TF-IDF + Logistic Regression only counts words, and cannot fully understand context and sarcasm.

As a result, both models are very strong on extreme ratings (1★ and 5★) but weaker on middle ratings.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: