```
!pip install pandas numpy scikit-learn nltk textblob gensim spacy transformers matplotlib
!python -m spacy download en_core_web_sm
!python -m nltk.downloader punkt wordnet omw-1.4 stopwords
```

Show hidden output

```




import pandas as pd
import numpy as np
import re
import nltk
import spacy
from textblob import TextBlob
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Deep Learning
import torch
from torch import nn
from torch.utils.data import DataLoader, TensorDataset
from transformers import BertTokenizer, BertForSequenceClassification, pipeline
from gensim.models import Word2Vec, FastText
from gensim.utils import simple_preprocess
import warnings
warnings.filterwarnings("ignore")

# NLTK & SpaCy
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('wordnet')
nlp = spacy.load("en_core_web_sm")

# Set style
plt.style.use('seaborn-v0_8')
sns.set_palette("husl")
```

Show hidden output

1.Text Pre-processing on file.txt

```
# a. Import necessary libraries
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from textblob import TextBlob
import spacy

# Auto-create file.txt with improper text
file_content = """
Natural language processing (NLP) is a sub-field of artificial intelligence, concerned wit

NLP is used in many applications such as: machine translation, sentiment analysis, speech

Some common challenges include: ambiguity in language, sarcasm, slang, and misspellings. F

Here are some sample sentences with errors:
- "Ths sentnce has speling erors."
- "I luv programing in pyhton!!!"
- "Artifical inteligence is the future..."

NLP techniques help clean and normalize such texts. Tokenization breaks text into words, s

Named entity recognition (NER) identifies names of people, organizations, locations. For e

Finally, sentence boundary detection splits text into sentences. This is important for sum
"""

with open("file.txt", "w", encoding="utf-8") as f:
    f.write(file_content.strip())

print("file.txt created successfully!")
nltk.download('punkt_tab')
nltk.download('averaged_perceptron_tagger_eng')
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')

nlp = spacy.load("en_core_web_sm")

# b. Load the text corpus
with open("file.txt", "r", encoding="utf-8") as f:
    text = f.read()

print("Original text length:", len(text))
print(text[:500], "\n")

# c. Tokenization
tokens = word_tokenize(text.lower())
print("First 30 tokens:", tokens[:30])

# d. Spelling correction
corrected_tokens = []
for token in tokens:
    corrected = str(TextBlob(token).correct())
    corrected_tokens.append(corrected)

corrected_text = " ".join(corrected_tokens)
print("\nFirst 10 corrected tokens:", corrected_tokens[:10])
print("\nCorrected text (first 200 chars):", corrected_text[:200])

# e. POS tagging on corrected tokens
pos_tags = nltk.pos_tag(corrected_tokens)
print("\nPOS tags (first 10):", pos_tags[:10])

# f. Remove stop words
stop_words = set(stopwords.words("english"))
filtered_tokens = [t for t in corrected_tokens if t not in stop_words and t.isalpha()]
print("\nFirst 20 tokens after stop-word removal:", filtered_tokens[:20])

# g. Stemming & Lemmatization
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

stemmed = [stemmer.stem(t) for t in filtered_tokens]
lemmatized = [lemmatizer.lemmatize(t) for t in filtered_tokens]

print("\nStemmed (first 20):", stemmed[:20])
print("Lemmatized (first 20):", lemmatized[:20])

# h. Sentence boundaries
sentences = sent_tokenize(text)
print(f"\nTotal number of sentences: {len(sentences)}")
```

```
file.txt created successfully!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

```
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
Original text length: 1253
Natural language processing (NLP) is a sub-field of artificial intelligence, concerned with the interactions between compute

NLP is used in many applications such as: machine translation, sentiment analysis, speech recognition, chatbots and informat

Some common challenges include: ambiguity in language, sarcasm, slang, and misspellin

First 30 tokens: ['natural', 'language', 'processing', '(', 'nlp', ')', 'is', 'a', 'sub-field', 'of', 'artificial', 'intelli

First 10 corrected tokens: ['natural', 'language', 'processing', '(', 'nap', ')', 'is', 'a', 'sub-field', 'of']

Corrected text (first 200 chars): natural language processing ( nap ) is a sub-field of artificial intelligence , concerned

POS tags (first 10): [('natural', 'JJ'), ('language', 'NN'), ('processing', 'NN'), ('(', '('), ('nap', 'JJ'), (')', ')'), ('

First 20 tokens after stop-word removal: ['natural', 'language', 'processing', 'nap', 'artificial', 'intelligence', 'concern

Stemmed (first 20): ['natur', 'languag', 'process', 'nap', 'artifici', 'intellig', 'concern', 'interact', 'comput', 'human',
Lemmatized (first 20): ['natural', 'language', 'processing', 'nap', 'artificial', 'intelligence', 'concerned', 'interaction'

Total number of sentences: 14
```

2.Feature Extraction on 20newsgroups

```python
# a. Import packages
from sklearn.datasets import fetch_20newsgroups
import pandas as pd
import re
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

# b. Fetch dataset
data = fetch_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'))
df = pd.DataFrame({'text': data.data, 'target': data.target})

# c. Clean data
def clean_text(txt):
    txt = txt.lower()
    txt = re.sub(r'[^a-z\s]', '', txt)
    txt = re.sub(r'\s+', ' ', txt).strip()
    return txt

df['cleaned'] = df['text'].apply(clean_text)

# d. BoW model
bow_vectorizer = CountVectorizer(max_features=5000, stop_words='english')
bow_matrix = bow_vectorizer.fit_transform(df['cleaned'])
bow_df = pd.DataFrame(bow_matrix.toarray(), columns=bow_vectorizer.get_feature_names_out())

# e. TF-IDF model
tfidf_vectorizer = TfidfVectorizer(max_features=5000, stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(df['cleaned'])
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf_vectorizer.get_feature_names_

# f. Compare top 20 frequent words
bow_sums = bow_df.sum().sort_values(ascending=False).head(20)
tfidf_sums = tfidf_df.sum().sort_values(ascending=False).head(20)

print("BoW Top 20 words:\n", bow_sums)
print("\nTF-IDF Top 20 words:\n", tfidf_sums)
```

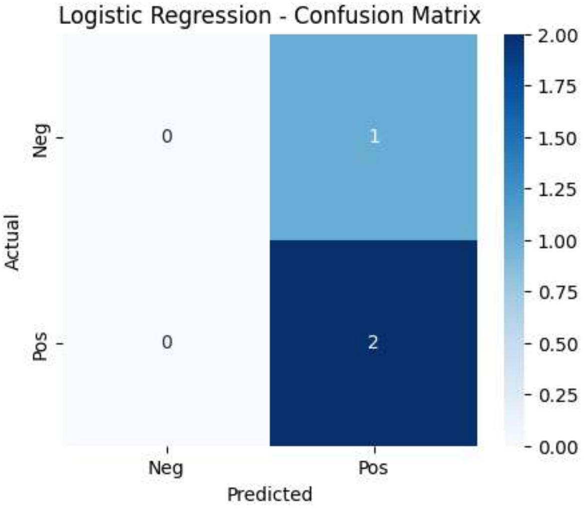
```
BoW Top 20 words:
 dont                            6433
like                            6391
```

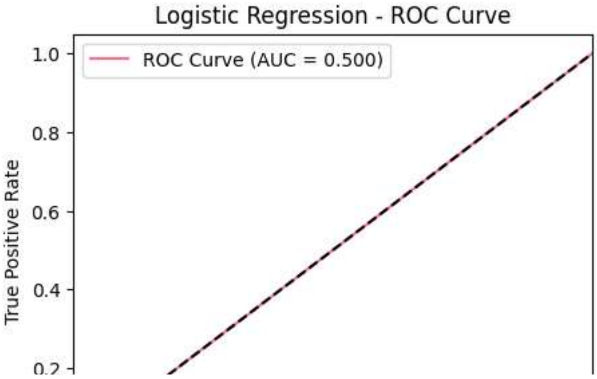
```
people                           6294
just                             6155
know                             5752
use                              4989
think                            4983
time                             4601
does                             4489
new                              4034
im                               4004
good                             3878
make                             3334
way                              3328
maxaxaxaxaxaxaxaxaxaxaxaxaxaxax    3317
used                             3028
say                              3007
did                              2989
god                              2949
right                            2944
dtype: int64

TF-IDF Top 20 words:
 like      324.598339
just      322.250787
know      318.619716
dont      317.050850
think     271.152683
does      267.344562
people    265.967689
im        265.655044
use       235.307722
thanks    232.253079
good      223.114082
time      215.373138
new       200.757875
make      177.541428
need      175.007184
way       174.105098
want      171.583419
did       171.285917
right     169.222452
ive       168.423608
dtype: float64
```

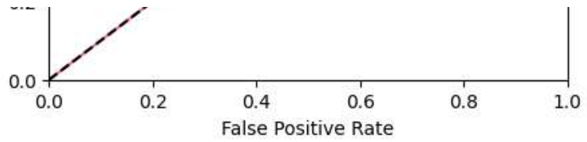
3.Amazon Musical Instruments Reviews

```python

# Import necessary libraries
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import (
    classification_report, confusion_matrix, accuracy_score,
    roc_curve, auc, mean_squared_error
)

# Classifiers
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
import xgboost as xgb

# Regressors
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
import xgboost as xgb

# Set style
sns.set_palette("husl")

# ==============================================
# a. Data Input
# ============================================

data = {
    "reviewText": [
        "Not much to write about here, but it does exac...",
        "The product does exactly as it should and is q...",
        "The primary job of this device is to block the...",
        "Nice windscreen protects my MXL mic and preven...",
        "This pop filter is great. It looks and perform...",
        "So good that I bought another one. Love the h...",
        "I have used monster cables for years, and with...",
        "I now use this cable to run from the output of...",
        "Perfect for my Epiphone Sheraton II. Monster ...",
        "Monster makes the best cables and a lifetime w..."
    ],
    "Overall": [5, 4, 3, 5, 4, 4, 5, 5, 3, 4]
}

df = pd.DataFrame(data)
print("Original Data:")
print(df)
print("\n" + "="*60 + "\n")

# ============================================
# b. Clean reviewText
# ============================================

def clean_review(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)  # Remove punctuation & numbers
    text = re.sub(r'\s+', ' ', text).strip()  # Normalize whitespace
    return text

df['cleaned_review'] = df['reviewText'].apply(clean_review)

print("Cleaned Reviews (first 3):")
for i in range(3):
    print(f"{i+1}. {df['cleaned_review'].iloc[i]}")
print("\n" + "="*60 + "\n")

# ============================================
# c. TF-IDF Feature Extraction
# ============================================

vectorizer = TfidfVectorizer(
    max_features=100,
    stop_words='english',
    ngram_range=(1,2),  # Include bigrams for richer features
    min_df=1
)

X = vectorizer.fit_transform(df['cleaned_review'])
feature_names = vectorizer.get_feature_names_out()

print(f"TF-IDF Matrix Shape: {X.shape}")
print(f"Features (sample): {feature_names[:10]}")
print("\n" + "="*60 + "\n")

# Target for classification: 1 = Positive (4–5), 0 = Negative (1–3)
y_class = (df['Overall'] >= 4).astype(int)  # Critical Fix: Use 0 and 1
y_reg = df['Overall']

# Train-test split
X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(
    X, y_class, test_size=0.3, random_state=42, stratify=y_class
)
X_train_r, X_test_r, y_train_r, y_test_r = train_test_split(
    X, y_reg, test_size=0.3, random_state=42
)

# ============================================
# d. Models Definition
# ============================================

classifiers = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Naive Bayes": MultinomialNB(),
    "KNN": KNeighborsClassifier(n_neighbors=3),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "GBM": GradientBoostingClassifier(random_state=42),
    "XGBoost": xgb.XGBClassifier(
        use_label_encoder=False,
        eval_metric='logloss',
        random_state=42
    )
}

regressors = {
    "Linear Regression": LinearRegression(),
    "Decision Tree Reg": DecisionTreeRegressor(random_state=42),
    "Random Forest Reg": RandomForestRegressor(n_estimators=100, random_state=42),
    "GBM Reg": GradientBoostingRegressor(random_state=42),
    "XGB Reg": xgb.XGBRegressor(random_state=42)
}

# ============================================
# e. Evaluation Function (Classification)
# ============================================

def evaluate_classification(model, X_test, y_test, name):
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1] if hasattr(model, "predict_proba") else No

    print(f"\n{'='*20} {name} {'='*20}")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.3f}")
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred, target_names=['Negative', 'Positive']))

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['Neg', 'Pos'], yticklabels=['Neg', 'Pos'])
    plt.title(f"{name} - Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

    # ROC Curve
    if y_prob is not None:
        fpr, tpr, _ = roc_curve(y_test, y_prob)
        roc_auc = auc(fpr, tpr)
        plt.figure(figsize=(5,4))
        plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.3f})')
        plt.plot([0,1], [0,1], 'k--')
        plt.xlim([0,1]); plt.ylim([0,1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title(f"{name} - ROC Curve")
        plt.legend()
        plt.show()

# ============================================
# CLASSIFICATION RESULTS
# ============================================

print("\nCLASSIFICATION RESULTS\n" + "-"*60)
for name, clf in classifiers.items():
    clf.fit(X_train_c, y_train_c)
    evaluate_classification(clf, X_test_c, y_test_c, name)

# ============================================
# REGRESSION RESULTS
# ============================================

print("\n\nREGRESSION RESULTS\n" + "-"*60)

for name, reg in regressors.items():
    reg.fit(X_train_r, y_train_r)
    y_pred = reg.predict(X_test_r)
    mse = mean_squared_error(y_test_r, y_pred)
    print(f"{name:25} → MSE: {mse:.3f} | RMSE: {np.sqrt(mse):.3f}")

# Optional: Predict on first review
print("\n" + "="*60)
print("SAMPLE PREDICTION (First Review):")
sample = vectorizer.transform([df['cleaned_review'].iloc[0]])
for name, clf in classifiers.items():
    pred = clf.predict(sample)[0]
    prob = clf.predict_proba(sample)[0]
    print(f"{name:25}: {'Positive' if pred==1 else 'Negative'} (Prob: {prob[pred]:.3f})")
```

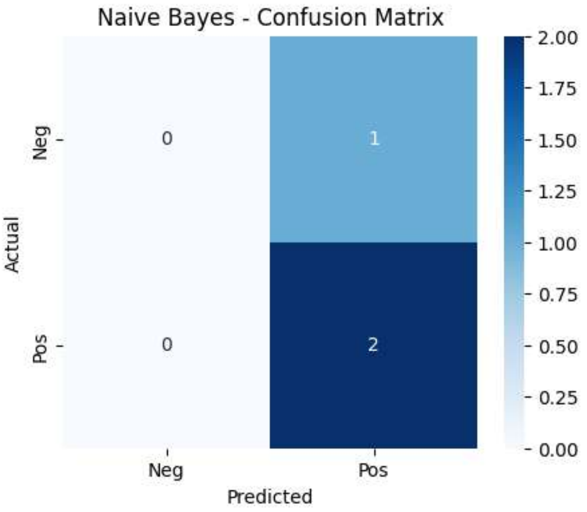
```
Original Data:
                                          reviewText  Overall
0  Not much to write about here, but it does exac...        5
1  The product does exactly as it should and is q...        4
2  The primary job of this device is to block the...        3
3  Nice windscreen protects my MXL mic and preven...        5
4  This pop filter is great. It looks and perform...        4
5   So good that I bought another one. Love the h...        4
6  I have used monster cables for years, and with...        5
7  I now use this cable to run from the output of...        5
8   Perfect for my Epiphone Sheraton II. Monster ...        3
9  Monster makes the best cables and a lifetime w...        4

============================================================

Cleaned Reviews (first 3):
1. not much to write about here but it does exac
2. the product does exactly as it should and is q
3. the primary job of this device is to block the

============================================================

TF-IDF Matrix Shape: (10, 70)
Features (sample): ['best' 'best cables' 'block' 'bought' 'bought love' 'cable' 'cable run'
 'cables' 'cables lifetime' 'cables years']

============================================================


CLASSIFICATION RESULTS
------------------------------------------------------------

==================== Logistic Regression ====================
Accuracy: 0.667

Classification Report:
              precision    recall  f1-score   support

    Negative       0.00      0.00      0.00         1
    Positive       0.67      1.00      0.80         2

    accuracy                           0.67         3
   macro avg       0.33      0.50      0.40         3
weighted avg       0.44      0.67      0.53         3

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

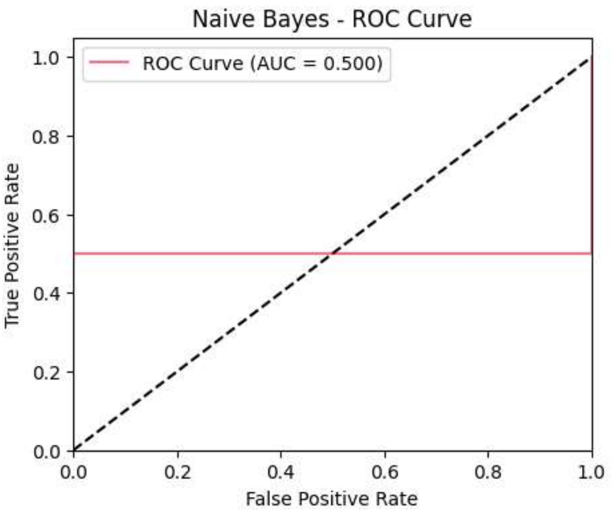






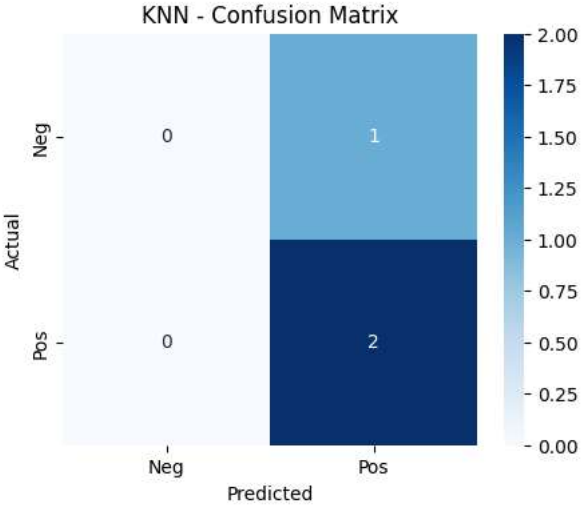
```
==================== Naive Bayes ====================
Accuracy: 0.667

Classification Report:
              precision    recall  f1-score   support

    Negative       0.00      0.00      0.00         1
    Positive       0.67      1.00      0.80         2

    accuracy                           0.67         3
   macro avg       0.33      0.50      0.40         3
weighted avg       0.44      0.67      0.53         3

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

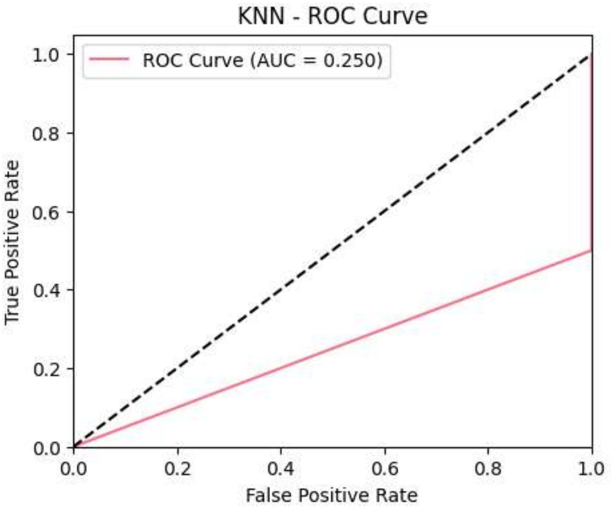




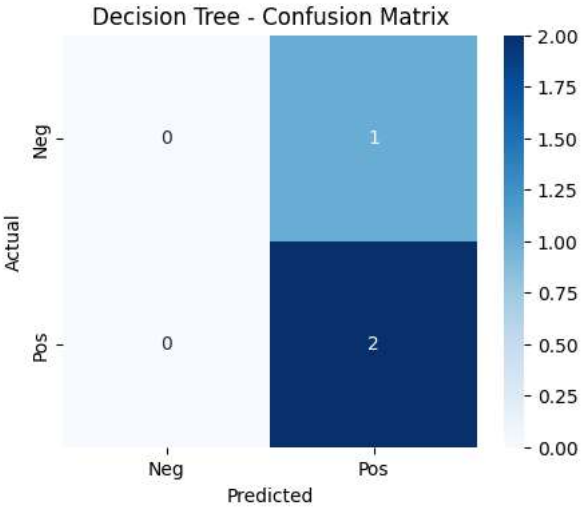
```
==================== KNN ====================
Accuracy: 0.667

Classification Report:
              precision    recall  f1-score   support

    Negative       0.00      0.00      0.00         1
    Positive       0.67      1.00      0.80         2

    accuracy                           0.67         3
   macro avg       0.33      0.50      0.40         3
weighted avg       0.44      0.67      0.53         3

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
```

```
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

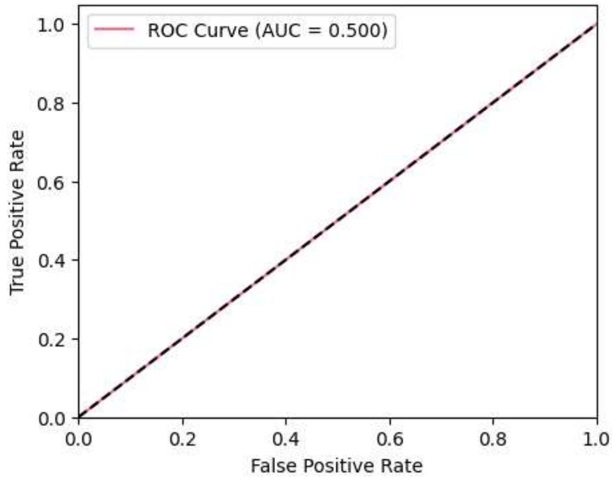




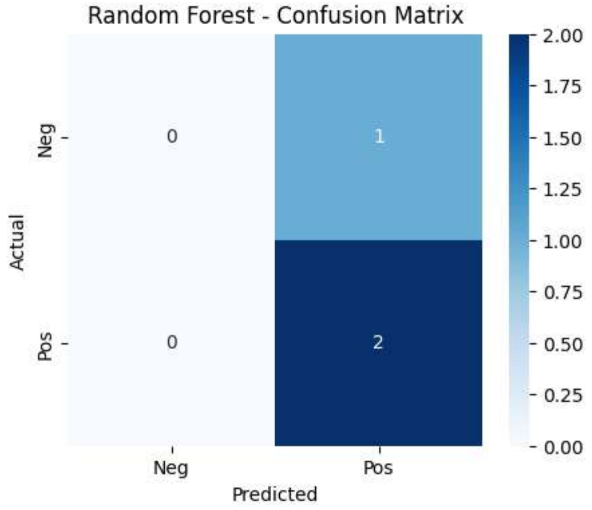
```
==================== Decision Tree ====================
Accuracy: 0.667

Classification Report:
              precision    recall  f1-score   support

    Negative       0.00      0.00      0.00         1
    Positive       0.67      1.00      0.80         2

    accuracy                           0.67         3
   macro avg       0.33      0.50      0.40         3
weighted avg       0.44      0.67      0.53         3

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

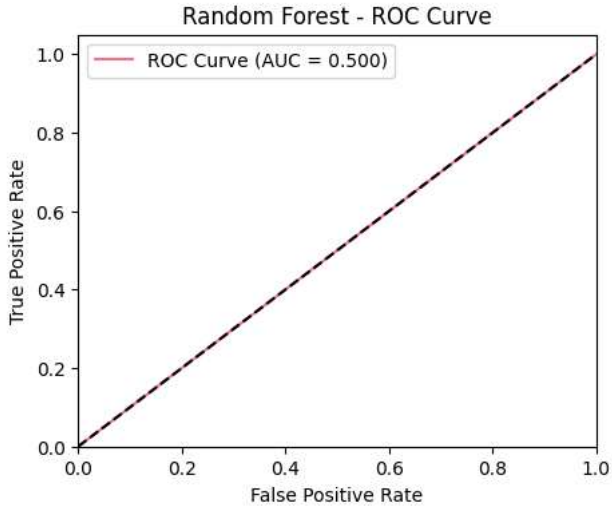


Decision Tree - ROC Curve



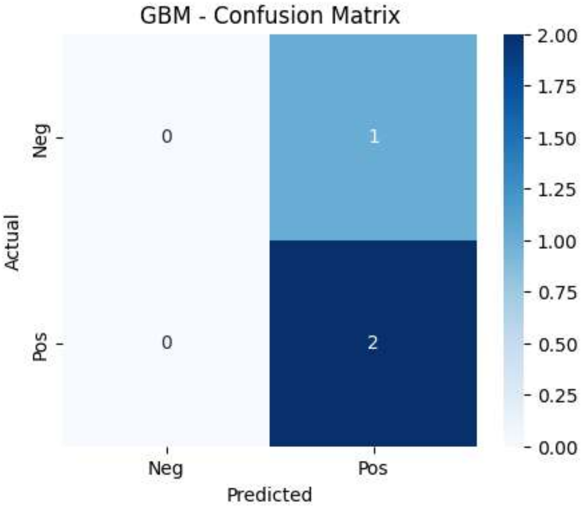
```
=================== Random Forest ===================
Accuracy: 0.667

Classification Report:
              precision    recall  f1-score   support

    Negative       0.00      0.00      0.00         1
    Positive       0.67      1.00      0.80         2

    accuracy                           0.67         3
   macro avg       0.33      0.50      0.40         3
weighted avg       0.44      0.67      0.53         3

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

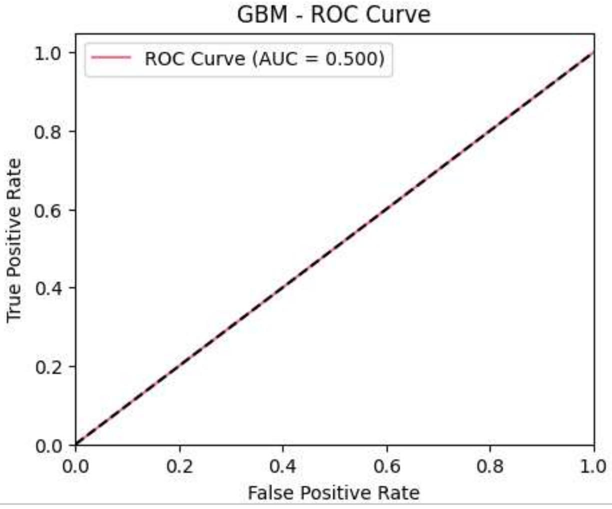




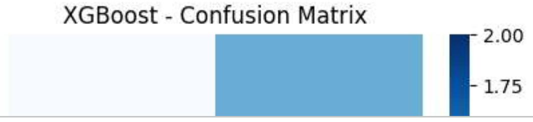
```
=================== GBM ===================
Accuracy: 0.667
```

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

    Negative       0.00      0.00      0.00         1
    Positive       0.67      1.00      0.80         2

    accuracy                           0.67         3
   macro avg       0.33      0.50      0.40         3
weighted avg       0.44      0.67      0.53         3

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```





```
==================== XGBoost ====================
Accuracy: 0.667

Classification Report:
              precision    recall  f1-score   support

    Negative       0.00      0.00      0.00         1
    Positive       0.67      1.00      0.80         2

    accuracy                           0.67         3
   macro avg       0.33      0.50      0.40         3
weighted avg       0.44      0.67      0.53         3

/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [15:17:15] WARNING: /workspace/src/learner.c
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



4.IMDB Sentiment Analysis (Kaggle)

```python
from google.colab import files
import os

if not os.path.exists("/root/.kaggle"):
    os.makedirs("/root/.kaggle", exist_ok=True)

print("Please upload your kaggle.json file:")
uploaded = files.upload()                     # <-- Upload kaggle.json
for fn in uploaded.keys():
    os.rename(fn, "/root/.kaggle/kaggle.json")
os.chmod("/root/.kaggle/kaggle.json", 600)

# ---------------------------------------------------------------
# 3. Download the exact dataset you mentioned
# ---------------------------------------------------------------
!kaggle datasets download -d columbine/imdb-dataset-sentiment-analysis-in-csv-format --un

# The dataset contains Train.csv and Test.csv
# We will use Test.csv as per your instruction
csv_path = "Test.csv"   # <-- This is the file we need
print(f"\nDataset ready: {csv_path}")

# ---------------------------------------------------------------
# 4. Import libraries
# ---------------------------------------------------------------
import pandas as pd
import re
from textblob import TextBlob
import warnings
warnings.filterwarnings("ignore")

# ---------------------------------------------------------------
# 5. Load the CSV file
# ---------------------------------------------------------------
df = pd.read_csv(csv_path)
print(f"Loaded {len(df)} rows from {csv_path}")
print(df.head())

# ---------------------------------------------------------------
# 6. Fetch the text column
# ---------------------------------------------------------------
# The column is named 'text' in this dataset
texts = df['text'].astype(str).copy()

# ---------------------------------------------------------------
# 7. Extract and remove @handles
# ---------------------------------------------------------------
def remove_handles(t):
    return re.sub(r'@\w+', '', t)

texts_clean = texts.apply(remove_handles)

# ---------------------------------------------------------------
# 8. Perform sentiment analysis using TextBlob
# ---------------------------------------------------------------
def get_sentiment(txt):
    blob = TextBlob(txt)
    polarity = blob.sentiment.polarity
    if polarity > 0.1:
        return 'positive'
    elif polarity < -0.1:
        return 'negative'
    else:
        return 'neutral'

```