```python
print("Installing required packages...")
import sys
import subprocess

packages = ['datasets', 'scikit-learn', 'pandas', 'numpy']
for package in packages:
    try:
        __import__(package.replace('-', '_'))
    except ImportError:
        print(f"Installing {package}...")
        subprocess.check_call([sys.executable, "-m", "pip", "install", package, "-q"])

print("✓ All packages installed!")

```

```
Installing required packages...
Installing scikit-learn...
✓ All packages installed!
```

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, mean_absolute_error
import re
import warnings
warnings.filterwarnings('ignore')

print("✓ Libraries imported successfully")
```

```
✓ Libraries imported successfully
```

```python
print("=" * 70)
print("LOADING AMAZON ELECTRONICS REVIEWS DATASET")
print("=" * 70)
```

```
======================================================================
LOADING AMAZON ELECTRONICS REVIEWS DATASET
======================================================================
```

```python
df = None
try:
    from datasets import load_dataset

    print("\nLoading Electronics reviews from HuggingFace...")
    print("This may take 2-3 minutes on first download...\n")

    dataset = load_dataset(
        "McAuley-Lab/Amazon-Reviews-2023",
        "raw_review_Electronics",
        split="full[:10000]",
        trust_remote_code=False  # Updated parameter
    )

    df = pd.DataFrame(dataset)
    print(f"✅ Loaded {len(df)} reviews via HuggingFace")

except Exception as e:
    print(f"HuggingFace method failed: {e}")
    print("\nTrying alternative: Old Amazon Dataset (2014)...\n")

    # Method 2: Old UCSD Dataset (backup)
    try:
        import gzip
        import json
        import urllib.request

        url = "http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/reviews_Electronics_5.json.gz"

        print("Downloading from UCSD...")
        urllib.request.urlretrieve(url, 'electronics.json.gz')

        reviews = []
        with gzip.open('electronics.json.gz', 'rt', encoding='utf-8') as f:
            for i, line in enumerate(f):
                if i >= 10000:
                    break
                reviews.append(json.loads(line))

        df = pd.DataFrame(reviews)
        print(f"✅ Loaded {len(df)} reviews from 2014 dataset")

    except Exception as e2:
        print(f"❌ All methods failed: {e2}")
        raise

# Display dataset info
print(f"\nDataset Shape: {df.shape}")
print(f"Columns: {df.columns.tolist()}")
print("\nFirst 3 reviews:")
```

```
Loading Electronics reviews from HuggingFace...
This may take 2-3 minutes on first download...

README.md:        30.3k/?  [00:00<00:00,  1.33MB/s]

Amazon-Reviews-2023.py:        39.6k/?  [00:00<00:00,  1.57MB/s]
HuggingFace method failed: Dataset scripts are no longer supported, but found Amazon-Reviews-2023.py

Trying alternative: Old Amazon Dataset (2014)...

Downloading from UCSD...
✅ Loaded 10000 reviews from 2014 dataset

Dataset Shape: (10000, 9)
Columns: ['reviewerID', 'asin', 'reviewerName', 'helpful', 'reviewText', 'overall', 'summary', 'unixReviewTime', 'reviewTime']

First 3 reviews:
       reviewerID        asin      reviewerName   helpful  \
0   AO94DHGC771SJ  0528881469           amazdnu    [0, 0]
1   AMO214LNFCEI4  0528881469  Amazon Customer  [12, 15]
2  A3N7T0DY83Y4IG  0528881469    C. A. Freeman  [43, 45]

                                          reviewText  overall  \
0  We got this GPS for my husband who is an (OTR)...      5.0
1  I'm a professional OTR truck driver, and I bou...      1.0
2  Well, what can I say.  I've had this unit in m...      3.0

            summary  unixReviewTime   reviewTime
0    Gotta have GPS!      1370131200   06 2, 2013
1  Very Disappointed      1290643200  11 25, 2010
2     1st impression      1283990400   09 9, 2010
```

```
print("\n" + "=" * 70)
print("DATA PREPROCESSING")
print("=" * 70)

# Identify column names
text_col = None
rating_col = None

for col in ['text', 'reviewText', 'review_text']:
    if col in df.columns:
        text_col = col
        break

for col in ['rating', 'overall', 'stars']:
    if col in df.columns:
        rating_col = col
        break

print(f"\n✓ Text column: '{text_col}'")
print(f"✓ Rating column: '{rating_col}'")

# Clean data
df_clean = df[[text_col, rating_col]].dropna()
print(f"✓ Removed {len(df) - len(df_clean)} rows with missing values")

# Convert ratings to integers
```

```python
df_clean[rating_col] = pd.to_numeric(df_clean[rating_col], errors='coerce')
df_clean = df_clean.dropna()
df_clean[rating_col] = df_clean[rating_col].astype(int)
df_clean = df_clean[df_clean[rating_col].isin([1, 2, 3, 4, 5])]

print(f"✓ Final dataset: {len(df_clean)} reviews")

# Show class distribution
print("\n📊 Class Distribution:")
dist = df_clean[rating_col].value_counts().sort_index()
for rating, count in dist.items():
    pct = (count / len(df_clean)) * 100
    print(f"   {rating}-star: {count:5d} ({pct:5.1f}%)")
```

```

======================================================================
DATA PREPROCESSING
======================================================================

✓ Text column: 'reviewText'
✓ Rating column: 'overall'
✓ Removed 0 rows with missing values
✓ Final dataset: 10000 reviews

📊 Class Distribution:
   1-star:   572 (  5.7%)
   2-star:   450 (  4.5%)
   3-star:   822 (  8.2%)
   4-star:  2095 ( 20.9%)
   5-star:  6061 ( 60.6%)
```

```python
# %% Cell 5: Text Cleaning
print("\n" + "=" * 70)
print("TEXT PREPROCESSING")
print("=" * 70)

def clean_text(text):
    """Clean and normalize text"""
    if not isinstance(text, str):
        return ""
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
    text = ' '.join(text.split())
    return text

print("\nCleaning text...")
df_clean['cleaned_text'] = df_clean[text_col].apply(clean_text)

# Remove very short reviews
df_clean = df_clean[df_clean['cleaned_text'].str.len() >= 10]
print(f"✓ Text cleaned. Final reviews: {len(df_clean)}")

print("\nExample cleaned review:")
print(f"Original: {df_clean[text_col].iloc[0][:100]}...")
print(f"Cleaned:  {df_clean['cleaned_text'].iloc[0][:100]}...")

```

```

======================================================================
TEXT PREPROCESSING
======================================================================

Cleaning text...
✓ Text cleaned. Final reviews: 9991

Example cleaned review:
Original: We got this GPS for my husband who is an (OTR) over the road trucker.  Very Impressed with the shipp...
Cleaned:  we got this gps for my husband who is an otr over the road trucker very impressed with the shipping ...
```

```python
print("\n" + "=" * 70)
print("FEATURE EXTRACTION")
print("=" * 70)

X = df_clean['cleaned_text']
y = df_clean[rating_col]

# Train-test split
print("\nSplitting data (80% train, 20% test)...")
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# TF-IDF Vectorization
print("Extracting TF-IDF features...")
vectorizer = TfidfVectorizer(max_features=1000, min_df=5, max_df=0.8)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

print(f"✓ Training set: {X_train_tfidf.shape[0]} samples")
print(f"✓ Test set: {X_test_tfidf.shape[0]} samples")
print(f"✓ Features: {X_train_tfidf.shape[1]} TF-IDF terms")

print("\nTop 10 features by TF-IDF score:")
feature names = vectorizer.get feature names out
```

```
======================================================================
FEATURE EXTRACTION
======================================================================

Splitting data (80% train, 20% test)...
Extracting TF-IDF features...
✓ Training set: 7992 samples
✓ Test set: 1999 samples
✓ Features: 1000 TF-IDF terms

Top 10 features by TF-IDF score:
```

```
print("\n" + "=" * 70)
print("TRAINING MODELS")
print("=" * 70)

# Model 1: Multinomial Naive Bayes
print("\n[1/2] Training Naive Bayes...")
nb_model = MultinomialNB()
nb_model.fit(X_train_tfidf, y_train)
nb_pred = nb_model.predict(X_test_tfidf)
print("✓ Naive Bayes trained")

# Model 2: Logistic Regression
print("\n[2/2] Training Logistic Regression...")
lr_model = LogisticRegression(max_iter=500, random_state=42, multi_class='multinomial')
lr_model.fit(X_train_tfidf, y_train)
lr_pred = lr_model.predict(X_test_tfidf)
print("✓ Logistic Regression trained")
```

```
======================================================================
TRAINING MODELS
======================================================================

[1/2] Training Naive Bayes...
✓ Naive Bayes trained

[2/2] Training Logistic Regression...
✓ Logistic Regression trained
```

```
# %% Cell 8: Evaluate Results
print("\n" + "=" * 70)
print("EVALUATION RESULTS")
print("=" * 70)

# Calculate metrics
nb_acc = accuracy_score(y_test, nb_pred)
nb_mae = mean_absolute_error(y_test, nb_pred)
nb_cm = confusion_matrix(y_test, nb_pred)

lr_acc = accuracy_score(y_test, lr_pred)
lr_mae = mean_absolute_error(y_test, lr_pred)
lr_cm = confusion_matrix(y_test, lr_pred)

print(f"\n📊 MULTINOMIAL NAIVE BAYES:")
print(f"   Accuracy: {nb_acc*100:.1f}% ({nb_acc:.4f})")
print(f"   MAE: {nb_mae:.2f}")

print(f"\n📊 LOGISTIC REGRESSION (Nominal):")
print(f"   Accuracy: {lr_acc*100:.1f}% ({lr_acc:.4f})")
print(f"   MAE: {lr_mae:.2f}")

# Confusion matrices
print("\n📈 Confusion Matrix - Naive Bayes:")
print(nb_cm)

print("\n📈 Confusion Matrix - Logistic Regression:")
print(lr_cm)
```

```
======================================================================
EVALUATION RESULTS
======================================================================

📊 MULTINOMIAL NAIVE BAYES:
   Accuracy: 60.5% (0.6053)
   MAE: 0.74

📊 LOGISTIC REGRESSION (Nominal):
   Accuracy: 62.6% (0.6258)
   MAE: 0.62

📈 Confusion Matrix - Naive Bayes:
[[   0    0    0    0  114]
 [   0    0    0    1   89]
 [   0    0    0    1  164]
 [   0    0    0    0  419]
 [   0    0    0    1 1210]]

📈 Confusion Matrix - Logistic Regression:
[[  23    2    5    6   78]
 [   9    3    6   20   52]
 [   6    1    6   47  105]
 [   2    1    3   92  321]
 [   2    0    4   78 1127]]
```

```
print("\n" + "=" * 70)
print("DETAILED ANALYSIS")
print("=" * 70)

# Adjacent rating error analysis
def calc_adjacent_errors(cm):
    """Calculate % of errors that are adjacent ratings"""
    total_errors = np.sum(cm) - np.trace(cm)
    if total_errors == 0:
        return 0
    adjacent_errors = 0
    for i in range(len(cm)):
        for j in range(len(cm)):
            if abs(i - j) == 1:
                adjacent_errors += cm[i][j]
    return (adjacent_errors / total_errors) * 100

nb_adj = calc_adjacent_errors(nb_cm)
lr_adj = calc_adjacent_errors(lr_cm)

print(f"\n📊 Adjacent Rating Errors:")
print(f"   Naive Bayes: {nb_adj:.1f}% of errors are adjacent (e.g., 4↔5)")
print(f"   Logistic Regression: {lr_adj:.1f}% of errors are adjacent")

# Per-class performance
print("\n📊 Per-Class F1-Scores:")
nb_report = classification_report(y_test, nb_pred, output_dict=True, zero_division=0)
lr_report = classification_report(y_test, lr_pred, output_dict=True, zero_division=0)

print("\nNaive Bayes:")
for rating in [1, 2, 3, 4, 5]:
    if str(rating) in nb_report:
        f1 = nb_report[str(rating)]['f1-score']
        support = nb_report[str(rating)]['support']
        print(f"   {rating}-star: F1 = {f1:.3f} (n={int(support)})")

print("\nLogistic Regression:")
for rating in [1, 2, 3, 4, 5]:
    if str(rating) in lr_report:
        f1 = lr_report[str(rating)]['f1-score']
        support = lr_report[str(rating)]['support']
```

```
======================================================================
DETAILED ANALYSIS
======================================================================

📊 Adjacent Rating Errors:
   Naive Bayes: 53.4% of errors are adjacent (e.g., 4↔5)
   Logistic Regression: 62.4% of errors are adjacent

📊 Per-Class F1-Scores:

Naive Bayes:
   1-star: F1 = 0.000 (n=114)
   2-star: F1 = 0.000 (n=90)
   3-star: F1 = 0.000 (n=165)
   4-star: F1 = 0.000 (n=419)
   5-star: F1 = 0.755 (n=1211)

Logistic Regression:
   1-star: F1 = 0.295 (n=114)
   2-star: F1 = 0.062 (n=90)
   3-star: F1 = 0.063 (n=165)
   4-star: F1 = 0.278 (n=419)
   5-star: F1 = 0.779 (n=1211)
```

```python
# Get key metrics
f1_5star = lr_report.get('5', {}).get('f1-score', 0)
f1_3star = lr_report.get('3', {}).get('f1-score', 0)

paragraph = f"""Initial experiments on {len(df_clean):,} Electronics reviews show promising
directions. Using TF-IDF features (max 1,000 features), Multinomial Naive Bayes
achieved {nb_acc*100:.1f}% accuracy with MAE of {nb_mae:.2f}, while Logistic Regression
(nominal treatment) achieved {lr_acc*100:.1f}% accuracy with MAE of {lr_mae:.2f}.
Confusion matrix analysis reveals that {lr_adj:.0f}% of misclassifications occur
between adjacent ratings (particularly 4↔5 stars), supporting our hypothesis that
ordinal treatment may improve performance. The class imbalance is evident—models
achieve {f1_5star*100:.0f}% F1-score for 5-star reviews but only {f1_3star*100:.0f}%
for 3-star reviews. These preliminary findings motivate our investigation into
whether ordinal methods can reduce MAE by better modeling rating structure while
addressing the adjacent-rating confusion problem."""

print("\n" + paragraph)
```

```
Initial experiments on 9,991 Electronics reviews show promising
directions. Using TF-IDF features (max 1,000 features), Multinomial Naive Bayes
achieved 60.5% accuracy with MAE of 0.74, while Logistic Regression
(nominal treatment) achieved 62.6% accuracy with MAE of 0.62.
Confusion matrix analysis reveals that 62% of misclassifications occur
between adjacent ratings (particularly 4↔5 stars), supporting our hypothesis that
ordinal treatment may improve performance. The class imbalance is evident—models
achieve 78% F1-score for 5-star reviews but only 6%
for 3-star reviews. These preliminary findings motivate our investigation into
whether ordinal methods can reduce MAE by better modeling rating structure while
addressing the adjacent-rating confusion problem.
```