amazon_review

February 23, 2025

1 AMAZON REVIEW ANALYSIS PROJECT

1.1 Data Inspection

let's do some inspection on the data, to see the *structure of the data* and check for *misssing values*

[24]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('amazon_reviews.csv')

print("Basic Dataset Info:")
print(df.info())

print("\nMissing Values Count:")
print(df.isnull().sum())

print("Row with missing reviewerName:")
print("\nFull details of the row:")
print(df[df['reviewerName'].isnull()][['reviewerName', 'overall', 'reviewText',␣
↪'reviewTime', 'helpful_yes', 'helpful_no']])

print("\nRow with missing reviewText:")
print("\nFull details of the row:")
print(df[df['reviewText'].isnull()][['reviewerName', 'overall', 'reviewText',␣
↪'reviewTime', 'helpful_yes', 'helpful_no']])
```

```
Basic Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4915 entries, 0 to 4914
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Unnamed: 0            4915 non-null   int64
 1   reviewerName          4914 non-null   object
 2   overall               4915 non-null   float64
 3   reviewText            4914 non-null   object
```

```
 4   reviewTime            4915 non-null   object
 5   day_diff              4915 non-null   int64
 6   helpful_yes           4915 non-null   int64
 7   helpful_no            4915 non-null   int64
 8   total_vote            4915 non-null   int64
 9   score_pos_neg_diff    4915 non-null   int64
 10  score_average_rating  4915 non-null   float64
 11  wilson_lower_bound    4915 non-null   float64
dtypes: float64(3), int64(6), object(3)
memory usage: 460.9+ KB
None

Missing Values Count:
Unnamed: 0              0
reviewerName            1
overall                 0
reviewText              1
reviewTime              0
day_diff                0
helpful_yes             0
helpful_no              0
total_vote              0
score_pos_neg_diff      0
score_average_rating    0
wilson_lower_bound      0
dtype: int64
Row with missing reviewerName:

Full details of the row:
  reviewerName  overall  reviewText  reviewTime  helpful_yes  helpful_no
0          NaN      4.0  No issues.  2014-07-23            0           0

Row with missing reviewText:

Full details of the row:
          reviewerName  overall reviewText  reviewTime  helpful_yes  \
125  Alexander Stevens      5.0        NaN  2012-08-21            2

     helpful_no
125           1
```

As we can see from here, we have missing values in the *reviewerName* and *reviewText* columns,

And we find them out of the dataset, and we can see they are two different rows,

As the reviewerName is inrelevant to our analysis, we can fill the missing values with *'Unknown'*,

And for the reviewText, we can simply *drop* it. (Cause the reviewText is the main part of the analysis)

[26]:
```python
df['reviewerName'] = df['reviewerName'].fillna('Unknown')
df = df.dropna(subset=['reviewText'])

print("Verification after processing:")
print("\nMissing Values Count:")
print(df.isnull().sum())

print("\nNew Dataset Info:")
print(df.info())
```

```
Verification after processing:

Missing Values Count:
Unnamed: 0              0
reviewerName            0
overall                 0
reviewText              0
reviewTime              0
day_diff                0
helpful_yes             0
helpful_no              0
total_vote              0
score_pos_neg_diff      0
score_average_rating    0
wilson_lower_bound      0
dtype: int64

New Dataset Info:
<class 'pandas.core.frame.DataFrame'>
Index: 4914 entries, 0 to 4914
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Unnamed: 0            4914 non-null   int64
 1   reviewerName          4914 non-null   object
 2   overall               4914 non-null   float64
 3   reviewText            4914 non-null   object
 4   reviewTime            4914 non-null   object
 5   day_diff              4914 non-null   int64
 6   helpful_yes           4914 non-null   int64
 7   helpful_no            4914 non-null   int64
 8   total_vote            4914 non-null   int64
 9   score_pos_neg_diff    4914 non-null   int64
 10  score_average_rating  4914 non-null   float64
 11  wilson_lower_bound    4914 non-null   float64
dtypes: float64(3), int64(6), object(3)
memory usage: 499.1+ KB
None
```

1.2 Data Preprocessing

Let's use nltk to clean the data, including removing stopwords, stemming, lemmatization, etc.

[27]:
```
import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')

def preprocess_text(text):
    # lowercasing
    # remove punctuation
    # remove extra whitespace
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()

    tokens = text.split()

    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]

    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token, pos='v') for token in tokens]
    tokens = [lemmatizer.lemmatize(token, pos='n') for token in tokens]
    tokens = [lemmatizer.lemmatize(token, pos='a') for token in tokens]

    return ' '.join(tokens)

df['cleaned_text'] = df['reviewText'].apply(preprocess_text)

print("Text Preprocessing Examples:")
for i in range(3):
    print(f"\nOriginal Text {i+1}:")
    print(df['reviewText'].iloc[i])
    print(f"\nProcessed Text {i+1}:")
    print(df['cleaned_text'].iloc[i])
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\MarkXu\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\MarkXu\AppData\Roaming\nltk_data…
```

```
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\MarkXu\AppData\Roaming\nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\MarkXu\AppData\Roaming\nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

```
Text Preprocessing Examples:

Original Text 1:
No issues.

Processed Text 1:
issue

Original Text 2:
Purchased this for my device, it worked as advertised. You can never have too
much phone memory, since I download a lot of stuff this was a no brainer for me.

Processed Text 2:
purchase device work advertise never much phone memory since download lot stuff
brainer

Original Text 3:
it works as expected. I should have sprung for the higher capacity.  I think its
made a bit cheesier than the earlier versions; the paint looks not as clean as
before

Processed Text 3:
work expect spring high capacity think make bite cheesy early version paint look
clean
```

As we can see, the text is cleaned and lemmatized. But in the first example, the text is not what it supposed to be meaning, for *no* is a stopword, so we need to keep those stopwords.

[28]:
```python
def preprocess_text(text):
    text = str(text).lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()

    tokens = text.split()

    stop_words = set(stopwords.words('english')) - {'no', 'not', 'nor', 'never'}
    tokens = [token for token in tokens if token not in stop_words]

    lemmatizer = WordNetLemmatizer()
```

```python
    tokens = [lemmatizer.lemmatize(token, pos='v') for token in tokens]
    tokens = [lemmatizer.lemmatize(token, pos='n') for token in tokens]
    tokens = [lemmatizer.lemmatize(token, pos='a') for token in tokens]

    return ' '.join(tokens)

df['cleaned_text'] = df['reviewText'].apply(preprocess_text)

print("Example results:")
for i in range(3):
    print(f"\nOriginal Text {i+1}:")
    print(df['reviewText'].iloc[i])
    print(f"Processed Text {i+1}:")
    print(df['cleaned_text'].iloc[i])
```

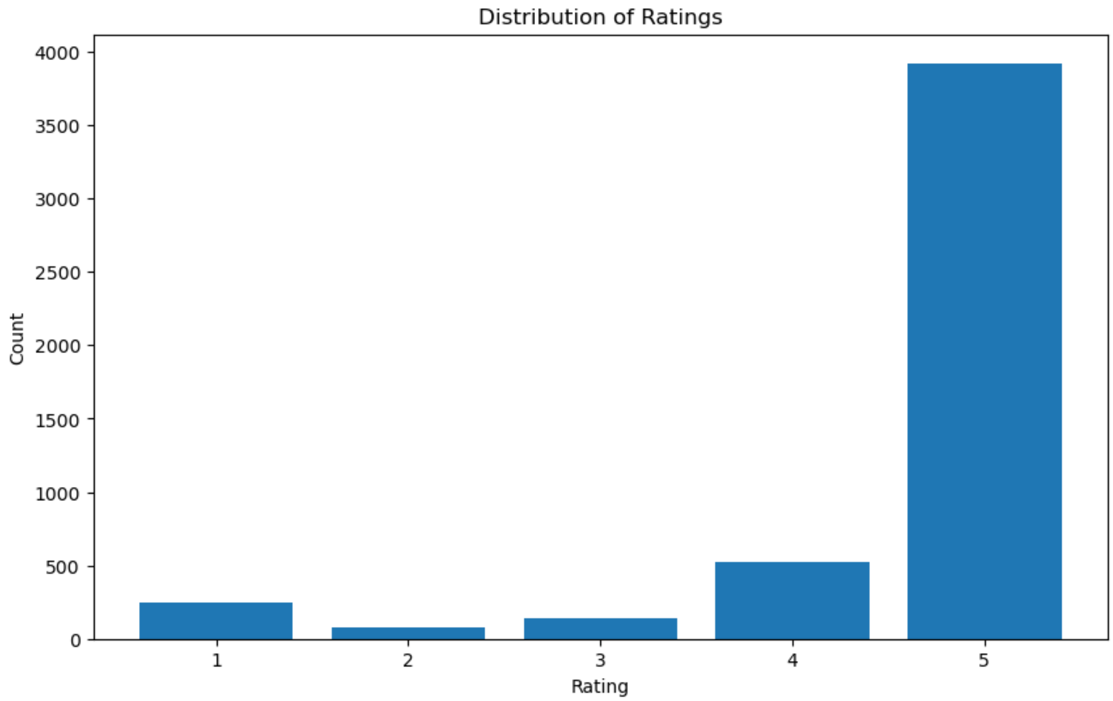
```
Example results:

Original Text 1:
No issues.
Processed Text 1:
no issue

Original Text 2:
Purchased this for my device, it worked as advertised. You can never have too
much phone memory, since I download a lot of stuff this was a no brainer for me.
Processed Text 2:
purchase device work advertise never much phone memory since download lot stuff
no brainer

Original Text 3:
it works as expected. I should have sprung for the higher capacity.  I think its
made a bit cheesier than the earlier versions; the paint looks not as clean as
before
Processed Text 3:
work expect spring high capacity think make bite cheesy early version paint look
not clean
```

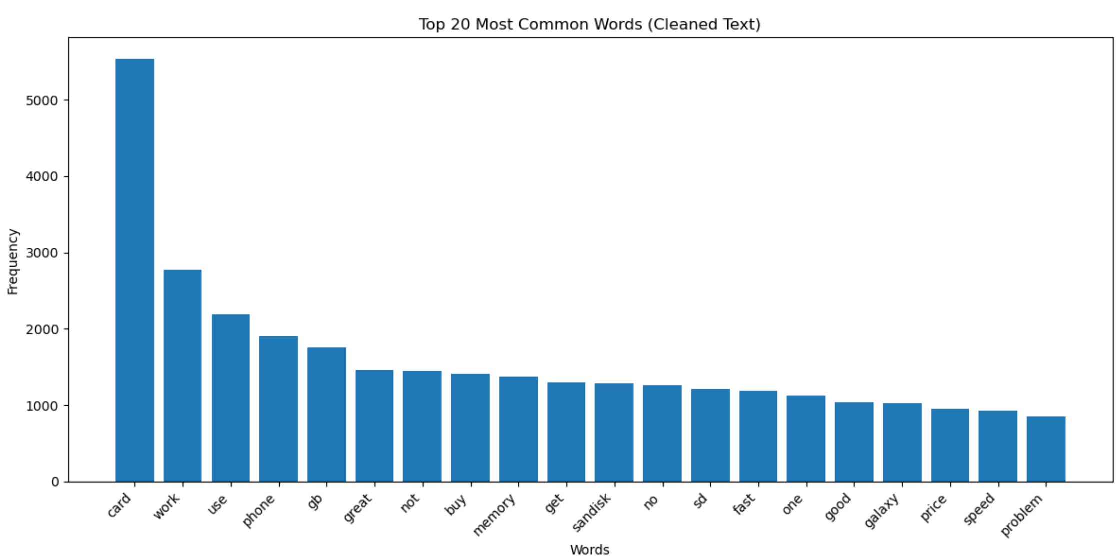
2 Exploratory Data Analysis(EDA)

[29]:
```python
from wordcloud import WordCloud
from collections import Counter
import numpy as np

plt.figure(figsize=(10, 6))
plt.hist(df['overall'],
         bins=np.arange(0.5, 5.6, 1.0),
         align='mid',
```

```
        rwidth=0.8)
plt.title('Distribution of Ratings')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.xticks(range(1, 6))
plt.show()

df['text_length'] = df['reviewText'].str.len()
df['word_count'] = df['reviewText'].str.split().str.len()

def get_top_words(texts, n=20):
    words = ' '.join(texts).split()
    return Counter(words).most_common(n)

top_words = get_top_words(df['cleaned_text'], n=20)
words, counts = zip(*top_words)

plt.figure(figsize=(12, 6))
plt.bar(words, counts)
plt.xticks(rotation=45, ha='right')
plt.title('Top 20 Most Common Words (Cleaned Text)')
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()

text = ' '.join(df['cleaned_text'])
wordcloud = WordCloud(width=800, height=400, background_color='white').
 ↪generate(text)

plt.figure(figsize=(15, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Reviews (Cleaned Text)')
plt.show()

print("\nBasic Statistics:")
print("\nReview Length Statistics:")
print(df['text_length'].describe())
print("\nWord Count Statistics:")
print(df['word_count'].describe())

print("\nRating Distribution:")
print(df['overall'].value_counts().sort_index())
print("\nRating Statistics:")
print(df['overall'].describe())
```

```
plt.figure(figsize=(10, 6))
plt.boxplot([df[df['overall'] == rating]['word_count'] for rating in␣
 ↪sorted(df['overall'].unique())],
            labels=sorted(df['overall'].unique()))
plt.title('Review Length by Rating (Word Count)')
plt.xlabel('Rating')
plt.ylabel('Number of Words')
plt.show()
```

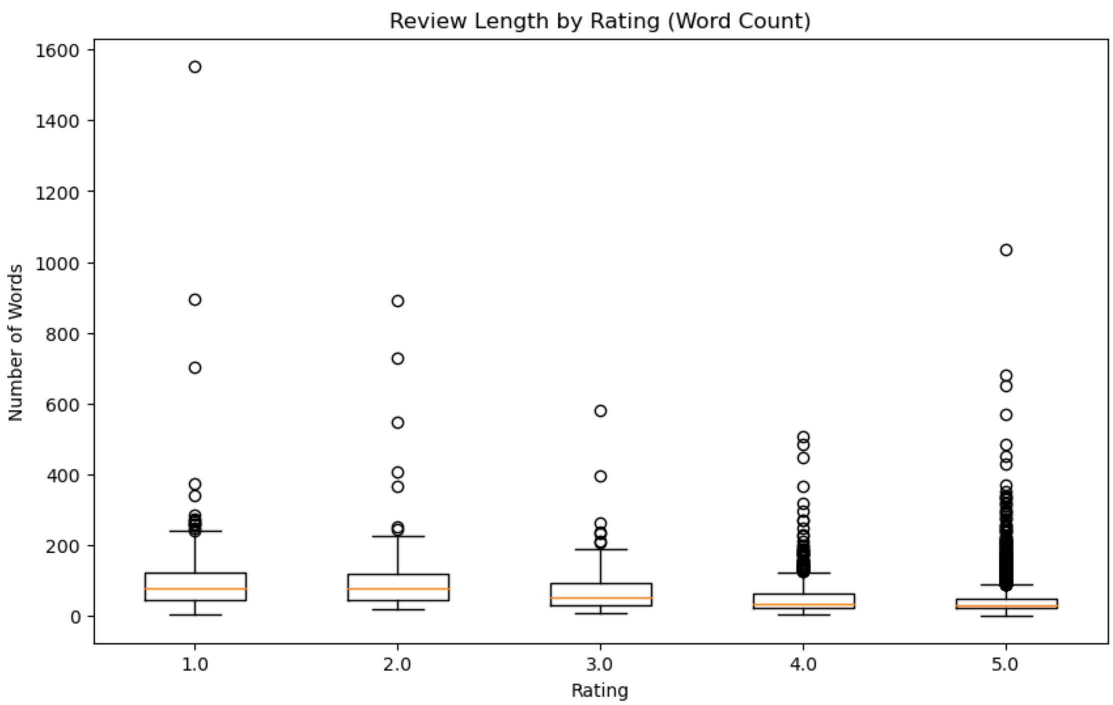




Word Cloud of Reviews (Cleaned Text)


```
Basic Statistics:

Review Length Statistics:
count    4914.000000
mean      267.747456
std       328.864594
min         3.000000
25%       123.000000
50%       172.000000
75%       289.000000
max      8638.000000
Name: text_length, dtype: float64

Word Count Statistics:
count    4914.000000
mean       50.452584
std        59.116494
min         1.000000
25%        23.000000
50%        33.000000
75%        55.000000
max      1554.000000
Name: word_count, dtype: float64

Rating Distribution:
```

```
overall
1.0     244
2.0      80
3.0     142
4.0     527
5.0    3921
Name: count, dtype: int64

Rating Statistics:
count    4914.000000
mean        4.587505
std         0.996929
min         1.000000
25%         5.000000
50%         5.000000
75%         5.000000
max         5.000000
Name: overall, dtype: float64
```



3 Feature Extraction

I am using TF-IDF for Feature extraction

[30]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer

print("\nExtracting TF-IDF features...")
tfidf = TfidfVectorizer(
    max_features=1000,
    min_df=5,
    max_df=0.95,
    stop_words='english'
)
tfidf_features = tfidf.fit_transform(df['cleaned_text'])
feature_names = tfidf.get_feature_names_out()

print(f"\nNumber of TF-IDF features: {len(feature_names)}")
print(f"Shape of TF-IDF matrix: {tfidf_features.shape}")

print("\nTop 20 most important words by average TF-IDF score:")
mean_tfidf = np.array(tfidf_features.mean(axis=0)).flatten()
top_word_indices = mean_tfidf.argsort()[-20:][::-1]
for idx in top_word_indices:
    if idx < len(feature_names):
        print(f"{feature_names[idx]}: {mean_tfidf[idx]:.4f}")
```

```
Extracting TF-IDF features…

Number of TF-IDF features: 1000
Shape of TF-IDF matrix: (4914, 1000)

Top 20 most important words by average TF-IDF score:
card: 0.0804
work: 0.0616
use: 0.0486
great: 0.0463
phone: 0.0446
memory: 0.0393
buy: 0.0387
gb: 0.0384
fast: 0.0354
good: 0.0348
price: 0.0321
galaxy: 0.0317
sd: 0.0315
sandisk: 0.0314
problem: 0.0278
samsung: 0.0267
speed: 0.0259
storage: 0.0255
tablet: 0.0253
```

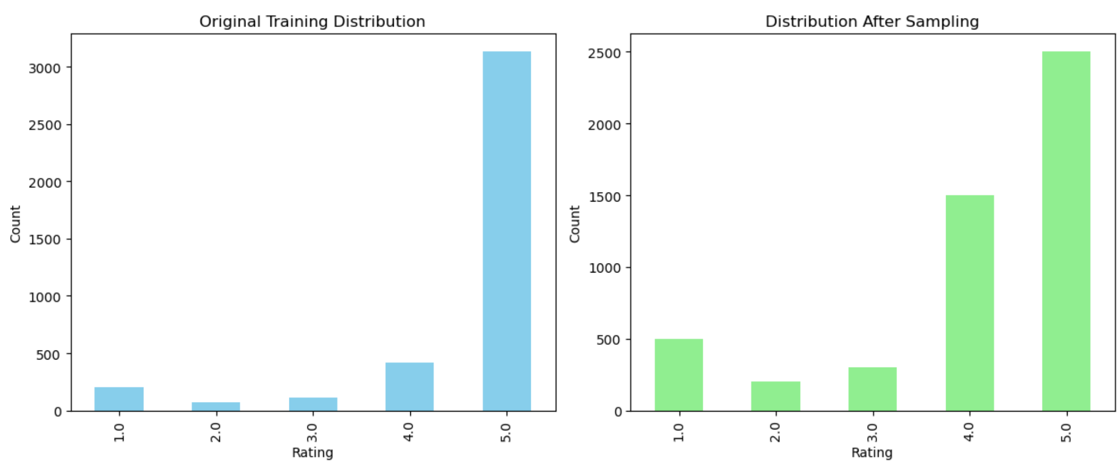
```
product: 0.0246
```

Let's deal with the imbalance of our dataset now.

[33]:
```python
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    tfidf_features,
    df['overall'],
    test_size=0.2,
    random_state=42
)
# Define sampling strategies
over_strategy = {
1: 500, # Oversample 1-star reviews to 500
2: 200, # Oversample 2-star reviews to 200
3: 300, # Oversample 3-star reviews to 300
4: 1500 # Oversample 4-star reviews to 1500
}
under_strategy = {5: 2500} # Undersample 5-star reviews to 2500
# Build sampling pipeline
sampler = Pipeline([
('over', SMOTE(
sampling_strategy=over_strategy,
k_neighbors=3, # Reduce number of neighbors
random_state=42
)),
('under', RandomUnderSampler(
sampling_strategy=under_strategy,
random_state=42
))
])
# Apply sampling
X_res, y_res = sampler.fit_resample(X_train, y_train)
# Verify distribution
print("Distribution after sampling:")
print(pd.Series(y_res).value_counts().sort_index())
# Visualize comparison
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
pd.Series(y_train).value_counts().sort_index().plot(kind='bar', color='skyblue')
plt.title('Original Training Distribution')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.subplot(1,2,2)
```

```
pd.Series(y_res).value_counts().sort_index().plot(kind='bar',color='lightgreen')
plt.title('Distribution After Sampling')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
# Validate 5-star samples
print("\n5-star samples validation:")
print(f"Original 5-star samples: {sum(y_train == 5)}")
print(f"5-star samples after undersampling: {sum(y_res == 5)}")
print("Random check of 5 undersampled 5-star reviews:")
for text in df.loc[y_res[y_res == 5].index[:5], 'cleaned_text']:
    print(f"- {text[:80]}...")
```

```
Distribution after sampling:
overall
1.0     500
2.0     200
3.0     300
4.0    1500
5.0    2500
Name: count, dtype: int64
```




```
5-star samples validation:
Original 5-star samples: 3131
5-star samples after undersampling: 2500
Random check of 5 undersampled 5-star reviews:
- buy memory card use dash cam im happy turn outit suppose thats no complaint…
- great work fast come full size sd adapterit clearly pay buy fast version sd…
- sd card work fine pretty quick not much else say product…
- buy gopro hero happy storage capacity format give hour minute p video fpsits
```

```
hig…
- reason memory card thousand star rating good price work well…
```

4 Prediction

since it is used to predict 0-5 ratings, we can use Random forest SVM and XGBoost for the classification to predict the value.

SVM Random Forest XGBoost

[38]:
```python
# Traditional Models Training
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report

traditional_results = {}
traditional_predictions = {}

models = {
    "SVM": SVC(kernel='linear', C=0.1, class_weight='balanced',␣
 ↪random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=200, max_depth=15,␣
 ↪class_weight='balanced', n_jobs=-1, random_state=42),
    "XGBoost": XGBClassifier(
        objective='multi:softmax',
        num_class=5,
        learning_rate=0.05,
        max_depth=7,
        min_child_weight=3,
        subsample=0.8,
        colsample_bytree=0.8,
        n_estimators=200,
        eval_metric='mlogloss',
        random_state=42
    )
}

for name, model in models.items():
    print(f"\nTraining {name}...")
    if name == "XGBoost":
        le = LabelEncoder()
        y_train_xgb = le.fit_transform(y_train)
        y_test_xgb = le.transform(y_test)
        model.fit(X_train_selected, y_train_xgb)
```

```python
        y_pred = le.inverse_transform(model.predict(X_test_selected))
    else:
        model.fit(X_train_selected, y_train)
        y_pred = model.predict(X_test_selected)

    traditional_results[name] = classification_report(y_test, y_pred,␣
↪zero_division=0, output_dict=True)
    traditional_predictions[name] = y_pred
    print(f"\n{name} Classification Report:")
    print(classification_report(y_test, y_pred, zero_division=0))

    print(f"\n{name} Prediction Distribution:")
    unique, counts = np.unique(y_pred, return_counts=True)
    print(dict(zip(unique, counts)))
```

```
Training SVM…

SVM Classification Report:
              precision    recall  f1-score   support

         1.0       0.38      0.57      0.46        44
         2.0       0.06      0.17      0.09        12
         3.0       0.08      0.13      0.10        30
         4.0       0.16      0.47      0.23       107
         5.0       0.90      0.59      0.72       790

    accuracy                           0.56       983
   macro avg       0.32      0.39      0.32       983
weighted avg       0.76      0.56      0.62       983


SVM Prediction Distribution:
{1.0: 65, 2.0: 32, 3.0: 48, 4.0: 319, 5.0: 519}

Training Random Forest…

Random Forest Classification Report:
              precision    recall  f1-score   support

         1.0       0.36      0.61      0.45        44
         2.0       0.00      0.00      0.00        12
         3.0       0.00      0.00      0.00        30
         4.0       0.18      0.12      0.14       107
         5.0       0.85      0.89      0.87       790

    accuracy                           0.76       983
   macro avg       0.28      0.33      0.29       983
```

```
weighted avg       0.72      0.76      0.73       983


Random Forest Prediction Distribution:
{1.0: 75, 4.0: 74, 5.0: 834}

Training XGBoost…

XGBoost Classification Report:
              precision    recall  f1-score   support

         1.0       0.59      0.39      0.47        44
         2.0       0.33      0.08      0.13        12
         3.0       0.00      0.00      0.00        30
         4.0       0.18      0.02      0.03       107
         5.0       0.83      0.99      0.90       790

    accuracy                           0.82       983
   macro avg       0.39      0.30      0.31       983
weighted avg       0.72      0.82      0.75       983


XGBoost Prediction Distribution:
{1.0: 29, 2.0: 3, 3.0: 1, 4.0: 11, 5.0: 939}
```

BERT

[49]:
```python
import os
os.environ['HTTP_PROXY'] = ''
os.environ['HTTPS_PROXY'] = ''
os.environ['NO_PROXY'] = '*'

from transformers import AutoTokenizer, AutoModelForSequenceClassification
from torch.utils.data import Dataset, DataLoader
import torch
from sklearn.metrics import classification_report
from tqdm import tqdm
from sklearn.model_selection import train_test_split

class ReviewDataset(Dataset):
    def __init__(self, reviews, ratings, tokenizer, max_length=512):
        self.encodings = tokenizer(
            list(map(str, reviews)),
            max_length=max_length,
            padding='max_length',
            truncation=True,
            return_tensors='pt'
        )
```

```
        self.labels = ratings

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        return {
            'input_ids': self.encodings['input_ids'][idx],
            'attention_mask': self.encodings['attention_mask'][idx],
            'labels': torch.tensor(self.labels[idx] - 1, dtype=torch.long)
        }

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model_name = 'bert-base-uncased'
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name,␣
 ↪num_labels=5).to(device)

train_texts, val_texts, train_labels, val_labels = train_test_split(
    df['reviewText'].values,
    df['overall'].values,
    test_size=0.2,
    random_state=42
)

train_dataset = ReviewDataset(train_texts, train_labels, tokenizer)
val_dataset = ReviewDataset(val_texts, val_labels, tokenizer)
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=16)

bert_results = {}
bert_predictions = {}
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5)
best_accuracy = 0
best_model_state = None

for epoch in range(3):
    model.train()
    for batch in train_loader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        optimizer.zero_grad()
        outputs = model(input_ids=input_ids, attention_mask=attention_mask,␣
 ↪labels=labels)
        outputs.loss.backward()
        optimizer.step()
```

```
    model.eval()
    predictions, true_labels = [], []
    with torch.no_grad():
        for batch in val_loader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)
            outputs = model(input_ids=input_ids, attention_mask=attention_mask)
            preds = torch.argmax(outputs.logits, dim=1)
            predictions.extend(preds.cpu().numpy())
            true_labels.extend(labels.cpu().numpy())

    accuracy = np.mean(np.array(predictions) == np.array(true_labels))
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_model_state = model.state_dict().copy()
        bert_predictions["BERT"] = predictions
        bert_results["BERT"] = classification_report(true_labels, predictions,
 ↪zero_division=0, output_dict=True)
    print(f'Epoch {epoch+1}: Accuracy: {accuracy:.4f}')

torch.save(best_model_state, 'best_model.pth')
```

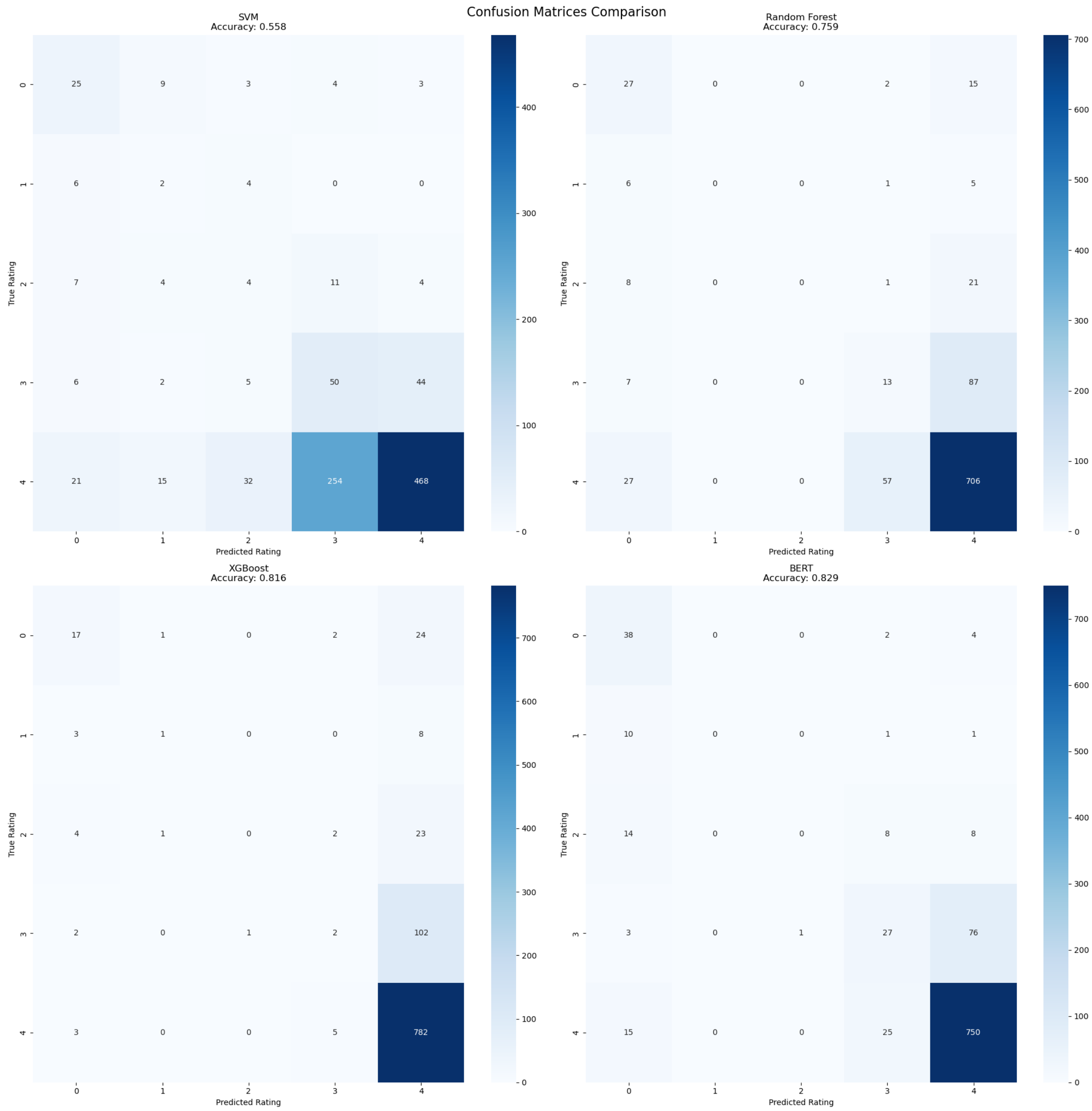
```
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
C:\Users\MarkXu\AppData\Local\Temp\ipykernel_24776\1178866174.py:31:
DeprecationWarning: an integer is required (got type numpy.float64).  Implicit
conversion to integers using __int__ is deprecated, and may be removed in a
future version of Python.
  'labels': torch.tensor(self.labels[idx] - 1, dtype=torch.long)

Epoch 1: Accuracy: 0.8230

C:\Users\MarkXu\AppData\Local\Temp\ipykernel_24776\1178866174.py:31:
DeprecationWarning: an integer is required (got type numpy.float64).  Implicit
conversion to integers using __int__ is deprecated, and may be removed in a
future version of Python.
  'labels': torch.tensor(self.labels[idx] - 1, dtype=torch.long)

Epoch 2: Accuracy: 0.8291

C:\Users\MarkXu\AppData\Local\Temp\ipykernel_24776\1178866174.py:31:
DeprecationWarning: an integer is required (got type numpy.float64).  Implicit
conversion to integers using __int__ is deprecated, and may be removed in a
future version of Python.
  'labels': torch.tensor(self.labels[idx] - 1, dtype=torch.long)
```

```
Epoch 3: Accuracy: 0.8220
```

[53]:
```
all_results = {**traditional_results, **bert_results}
all_predictions = {**traditional_predictions, **bert_predictions}

fig, axes = plt.subplots(2, 2, figsize=(20, 20))
fig.suptitle('Confusion Matrices Comparison', fontsize=16)

for idx, (name, y_pred) in enumerate(all_predictions.items()):
    row = idx // 2
    col = idx % 2
    y_true = y_test if name != "BERT" else true_labels
    cm = confusion_matrix(y_true, y_pred)
    accuracy = np.sum(np.diag(cm)) / np.sum(cm)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axes[row, col])
    axes[row, col].set_title(f'{name}\nAccuracy: {accuracy:.3f}')
    axes[row, col].set_ylabel('True Rating')
    axes[row, col].set_xlabel('Predicted Rating')
plt.tight_layout()
plt.show()

metrics = ['precision', 'recall', 'f1-score']
model_scores = {name: [] for name in all_predictions.keys()}
for name in all_predictions.keys():
    for metric in metrics:
        score = all_results[name]['weighted avg'][metric]
        model_scores[name].append(score)

plt.figure(figsize=(12, 6))
x = np.arange(len(metrics))
width = 0.2
colors = ['#FF9999', '#66B2FF', '#99FF99', '#FFCC99']

for i, (name, scores) in enumerate(model_scores.items()):
    bars = plt.bar(x + i*width, scores, width,
                  label=name,
                  color=colors[i],
                  alpha=0.8)
    for bar in bars:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2., height,
                f'{height:.2f}',
                ha='center', va='bottom')

plt.ylabel('Score')
plt.title('Model Performance Comparison')
plt.xticks(x + width*1.5, metrics)
```

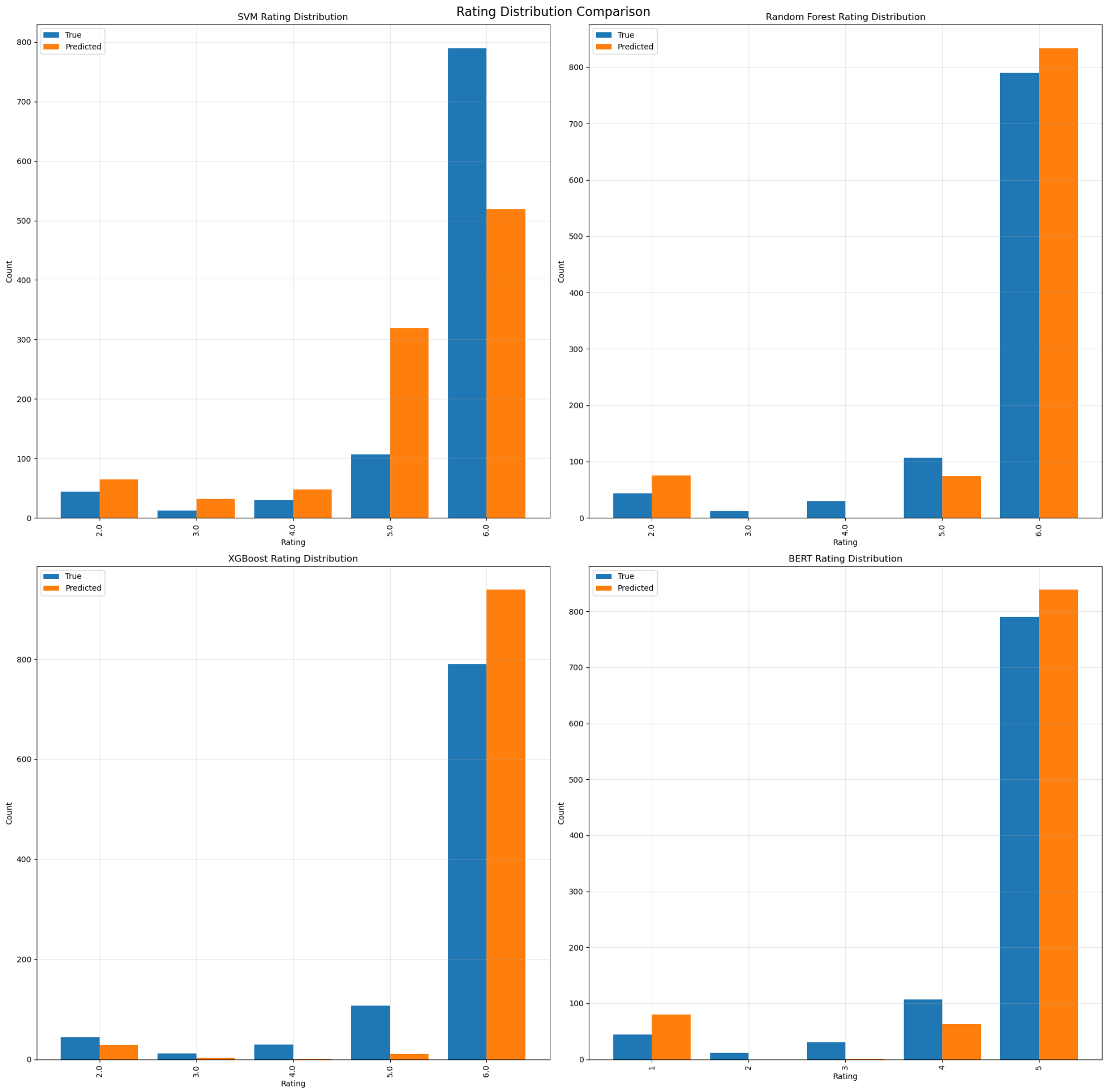
```
plt.legend(loc='upper right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

fig, axes = plt.subplots(2, 2, figsize=(20, 20))
fig.suptitle('Rating Distribution Comparison', fontsize=16)

for idx, (name, y_pred) in enumerate(all_predictions.items()):
    row = idx // 2
    col = idx % 2
    y_true = y_test if name != "BERT" else true_labels

    df_compare = pd.DataFrame({
        'True Ratings': np.array(y_true) + 1,
        'Predicted Ratings': np.array(y_pred) + 1
    })

    df_counts = pd.DataFrame({
        'True': df_compare['True Ratings'].value_counts().sort_index(),
        'Predicted': df_compare['Predicted Ratings'].value_counts().sort_index()
    })

    df_counts.plot(kind='bar', ax=axes[row, col], width=0.8)
    axes[row, col].set_title(f'{name} Rating Distribution')
    axes[row, col].set_xlabel('Rating')
    axes[row, col].set_ylabel('Count')
    axes[row, col].legend(['True', 'Predicted'])
    axes[row, col].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
def create_performance_table(results):
    table_data = []
    for model_name, result in results.items():
        for rating in sorted(result.keys()):
            if rating.isdigit():
                metrics = result[rating]
                table_data.append({
                    'Model': model_name,
                    'Rating': int(rating),
                    'Precision': round(metrics['precision'], 3),
                    'Recall': round(metrics['recall'], 3),
                    'F1-Score': round(metrics['f1-score'], 3),
                    'Support': metrics['support']
                })
        avg_metrics = result['weighted avg']
        table_data.append({
```

```
            'Model': f"{model_name} (Weighted Avg)",
            'Rating': 'All',
            'Precision': round(avg_metrics['precision'], 3),
            'Recall': round(avg_metrics['recall'], 3),
            'F1-Score': round(avg_metrics['f1-score'], 3),
            'Support': avg_metrics['support']
        })

    df_performance = pd.DataFrame(table_data)
    styled_df = df_performance.style.set_properties(**{
        'text-align': 'center',
        'padding': '8px'
    }).set_table_styles([
        {'selector': 'th',
         'props': [('text-align', 'center'),
                  ('background-color', '#f2f2f2'),
                  ('font-weight', 'bold')]},
        {'selector': 'td',
         'props': [('text-align', 'center')]},
        {'selector': 'tr:nth-of-type(even)',
         'props': [('background-color', '#f9f9f9')]},
        {'selector': 'tr:contains("Weighted Avg")',
         'props': [('background-color', '#e6f3ff'),
                  ('font-weight', 'bold')]}
    ]).format({
        'Precision': '{:.3f}',
        'Recall': '{:.3f}',
        'F1-Score': '{:.3f}',
        'Support': '{:,.0f}'
    })
    return styled_df

print("\nDetailed Performance Comparison Table:")
performance_table = create_performance_table(all_results)
display(performance_table)
```



Confusion Matrices Comparison
SVM
Accuracy: 0.558
Random Forest
Accuracy: 0.759
XGBoost
Accuracy: 0.816
BERT
Accuracy: 0.829
True Rating
Predicted Rating


Model Performance Comparison
SVM
Random Forest
XGBoost
BERT
Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.76
0.72
0.72
0.79
0.56
0.76
0.82
0.83
0.62
0.73
0.75
0.80
precision
recall
f1-score



Detailed Performance Comparison Table:

<pandas.io.formats.style.Styler at 0x26ce2334ca0>