```
# This Python 3 environment comes with many helpful analytics
libraries installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save &
Run All"
# You can also write temporary files to /kaggle/temp/, but they won't
be saved outside of the current session

# Import necessary libraries
import pandas as pd
import numpy as np
from collections import Counter
import matplotlib.pyplot as plt
import seaborn as sns
from ast import literal_eval
from datasets import load_dataset
from collections import defaultdict
```

1. Exploratory Data Analysis

Dataset Overview The dataset used for this task is derived from the FIQA 2018 dataset, which contains financial domain sentences annotated with aspects and sentiment scores. The primary objective is to predict:

The main aspect of a given sentence. The sentiment score associated with it, along with a sentiment label.

```
# Load the data
ds = load_dataset("pauri32/fiqa-2018")
```

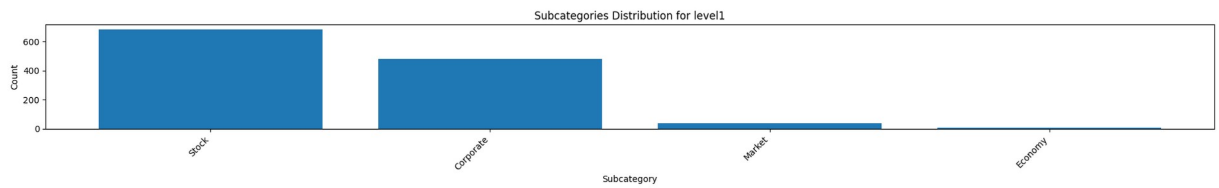
```python
# Convert each split to DataFrame
train_df = pd.DataFrame(ds['train'])
test_df = pd.DataFrame(ds['test'])
validate_df = pd.DataFrame(ds['validation'])

# Function to check valid 'aspects' strings
def is_valid_list_string(s):
    try:
        literal_eval(s)
        return True
    except (SyntaxError, ValueError):
        return False

# Filter out rows with invalid 'aspects' strings for all splits
train_df = train_df[train_df['aspects'].apply(is_valid_list_string)]

# Convert string representations of lists to actual lists
train_df.loc[:, 'aspects'] = train_df['aspects'].apply(literal_eval)

# Extract main aspects (taking the first level of hierarchy)
train_df['main_aspect'] = train_df['aspects'].apply(lambda x:
x[0].split('/')[0])

# Filter out rows with invalid 'aspects' strings for all splits
test_df = test_df[test_df['aspects'].apply(is_valid_list_string)]

# Convert string representations of lists to actual lists
test_df.loc[:, 'aspects'] = test_df['aspects'].apply(literal_eval)

# Extract main aspects (taking the first level of hierarchy)
test_df['main_aspect'] = test_df['aspects'].apply(lambda x:
x[0].split('/')[0])

# Filter out rows with invalid 'aspects' strings for all splits
validate_df =
validate_df[validate_df['aspects'].apply(is_valid_list_string)]

# Convert string representations of lists to actual lists
validate_df.loc[:, 'aspects'] =
validate_df['aspects'].apply(literal_eval)

# Extract main aspects (taking the first level of hierarchy)
validate_df['main_aspect'] = validate_df['aspects'].apply(lambda x:
x[0].split('/')[0])

# Concatenate for consistent label encoding
combined_df = pd.concat([train_df, test_df, validate_df])

# Create label encoder for aspects
aspect_encoder = LabelEncoder()
```

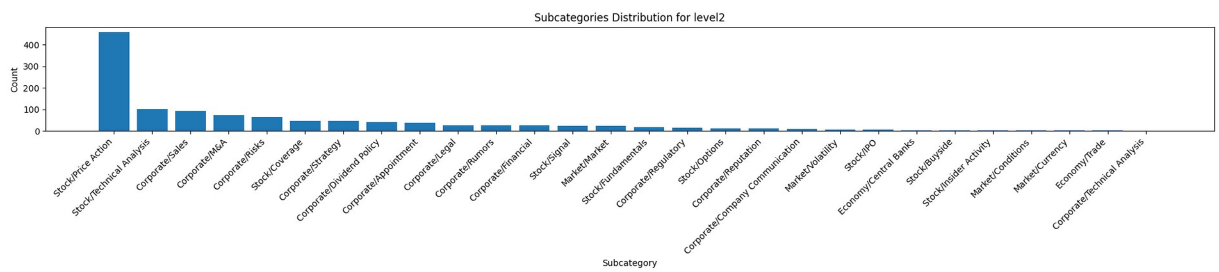
```python
combined_df['aspect_encoded'] =
aspect_encoder.fit_transform(combined_df['main_aspect'])

# Update the splits with the encoded aspects
train_df['aspect_encoded'] =
aspect_encoder.transform(train_df['main_aspect'])
test_df['aspect_encoded'] =
aspect_encoder.transform(test_df['main_aspect'])
validate_df['aspect_encoded'] =
aspect_encoder.transform(validate_df['main_aspect'])

# df = pd.DataFrame(ds['train'])
combined_df.head()
```

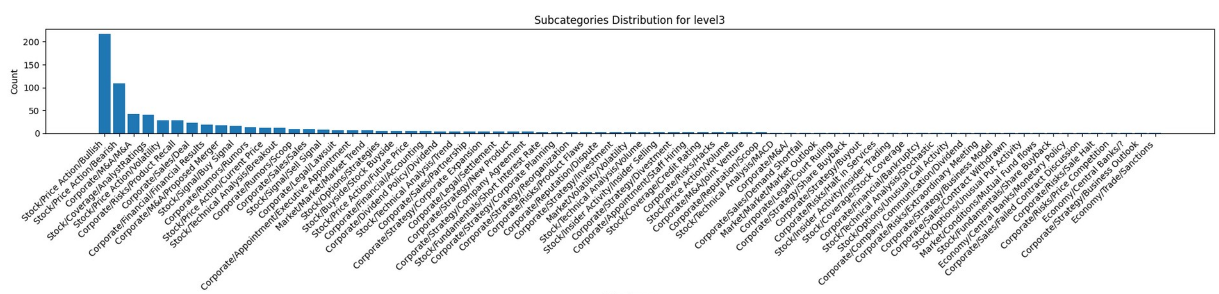
```
                                            sentence  \
0  Still short $LNG from $11.70 area...next stop ...
1                                    $PLUG bear raid
2  How Kraft-Heinz Merger Came Together in Speedy...
3     Slump in Weir leads FTSE down from record high
4                 $AAPL bounces off support, it seems

                                            snippets target
sentiment_score  \
0  ['Still short $LNG from $11.70 area...next sto...    LNG
-0.543
1                                      ['bear raid']   PLUG
-0.480
2        ['Merger Came Together in Speedy 10 Weeks']  Kraft
0.214
3                           ['down from record high']   Weir
-0.827
4                             ['bounces off support']   AAPL
0.443

                                         aspects    format  label
main_aspect  \
0  [Stock/Price Action/Volatility/Short Selling]      post      2
Stock
1                    [Stock/Price Action/Bearish]      post      2
Stock
2                            [Corporate/M&A/M&A]  headline      0
Corporate
3                 [Market/Volatility/Volatility]  headline      2
Market
4  [Stock/Price Action/Bullish/Bullish Behavior]      post      0
Stock

   aspect_encoded
0               3
1               3
```

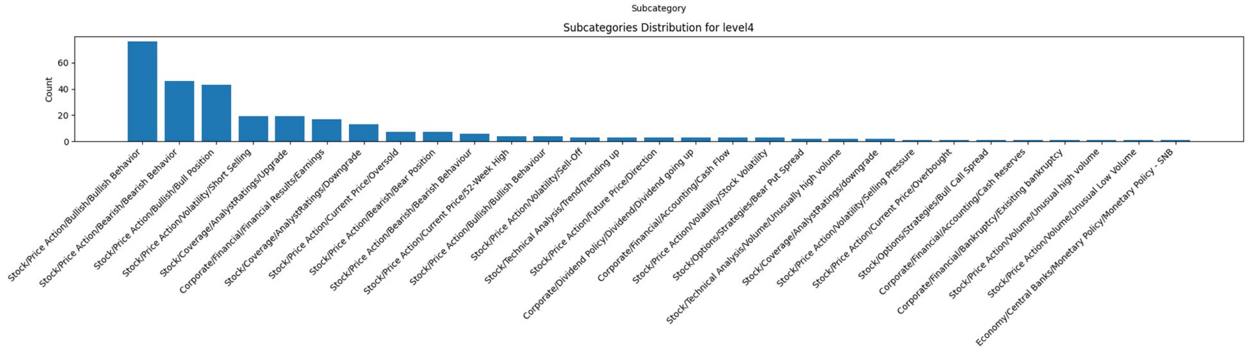
```
2                0
3                2
4                3

def count_aspects(aspect_lists):
    # Create nested defaultdict to store hierarchical counts
    hierarchy = defaultdict(lambda: defaultdict(lambda:
defaultdict(lambda: defaultdict(int))))

    # Count for all levels
    level_counts = {
        'level1': defaultdict(int),  # e.g., Stock
        'level2': defaultdict(int),  # e.g., Stock/Price Action
        'level3': defaultdict(int),  # e.g., Stock/Price
Action/Bullish
        'level4': defaultdict(int)   # e.g., Stock/Price
Action/Bullish/Bullish Behavior
    }

    for aspect_list in aspect_lists:
        for aspect in aspect_list:
            parts = aspect.split('/')

            # Count each level
            for i in range(len(parts)):
                current_path = '/'.join(parts[:i+1])
                if i == 0:
                    level_counts['level1'][parts[0]] += 1
                elif i == 1:
                    level_counts['level2'][current_path] += 1
                elif i == 2:
                    level_counts['level3'][current_path] += 1
                elif i == 3:
                    level_counts['level4'][current_path] += 1

    return level_counts

# Get hierarchical counts
aspect_hierarchy = count_aspects(combined_df['aspects'])

# Plot subcategories for each main category
plt.figure(figsize=(20, 5*len(aspect_hierarchy)))
plot_idx = 1

for main_category, subcategories in aspect_hierarchy.items():
    plt.subplot(len(aspect_hierarchy), 1, plot_idx)

    # Sort subcategories by count
    sorted_subcats = dict(sorted(subcategories.items(), key=lambda x:
x[1], reverse=True))
```

```
    plt.bar(sorted_subcats.keys(), sorted_subcats.values())
    plt.title(f'Subcategories Distribution for {main_category}')
    plt.xticks(rotation=45, ha='right')
    plt.xlabel('Subcategory')
    plt.ylabel('Count')
    plt.tight_layout()
    plot_idx += 1

plt.show()
```

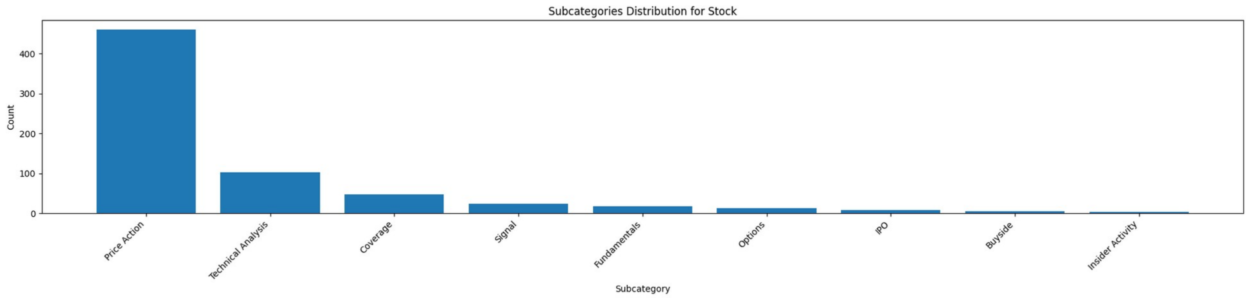


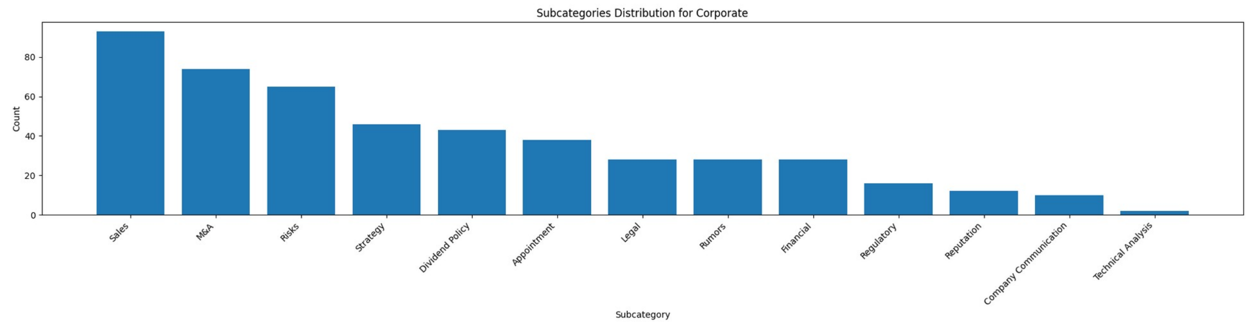


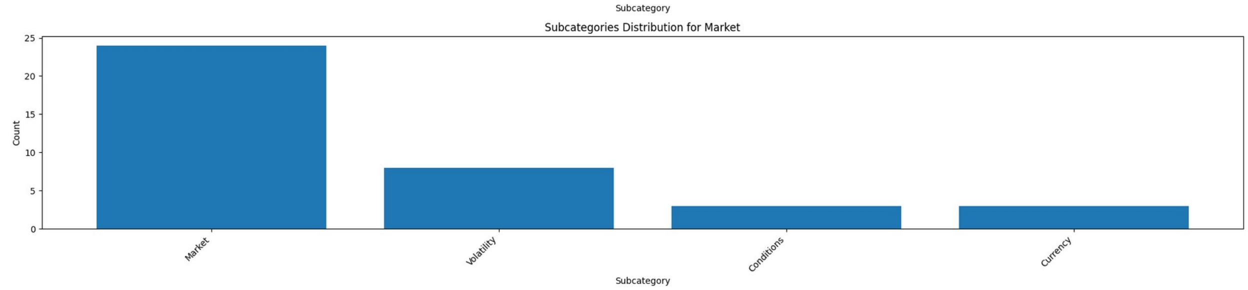


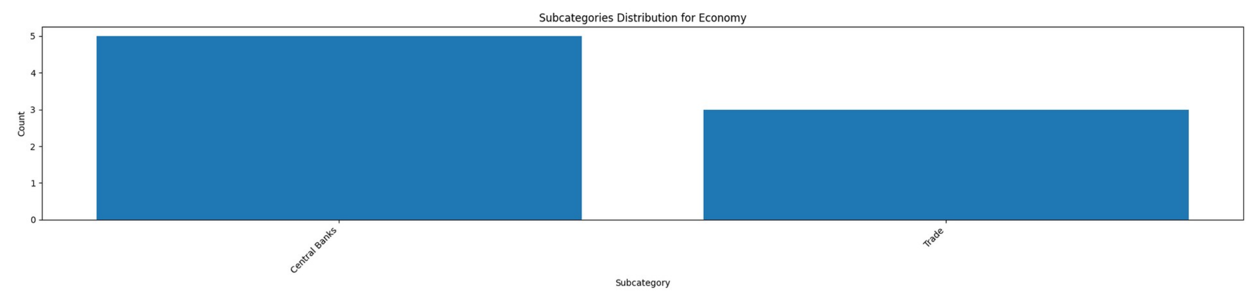


```python
# Function to count hierarchical aspects
def count_hierarchical_aspects(aspect_lists):
    hierarchy = defaultdict(lambda: defaultdict(int))

    for aspect_list in aspect_lists:
        for aspect in aspect_list:
            parts = aspect.split('/')
            main_category = parts[0]
            sub_category = parts[1] if len(parts) > 1 else 'Other'
            hierarchy[main_category][sub_category] += 1

    return hierarchy

# Get hierarchical counts
aspect_hierarchy = count_hierarchical_aspects(combined_df['aspects'])

# Plot subcategories for each main category
plt.figure(figsize=(20, 5*len(aspect_hierarchy)))
plot_idx = 1

for main_category, subcategories in aspect_hierarchy.items():
    plt.subplot(len(aspect_hierarchy), 1, plot_idx)

    # Sort subcategories by count
    sorted_subcats = dict(sorted(subcategories.items(), key=lambda x: x[1], reverse=True))

    plt.bar(sorted_subcats.keys(), sorted_subcats.values())
    plt.title(f'Subcategories Distribution for {main_category}')
    plt.xticks(rotation=45, ha='right')
    plt.xlabel('Subcategory')
    plt.ylabel('Count')
    plt.tight_layout()
    plot_idx += 1

plt.show()
```





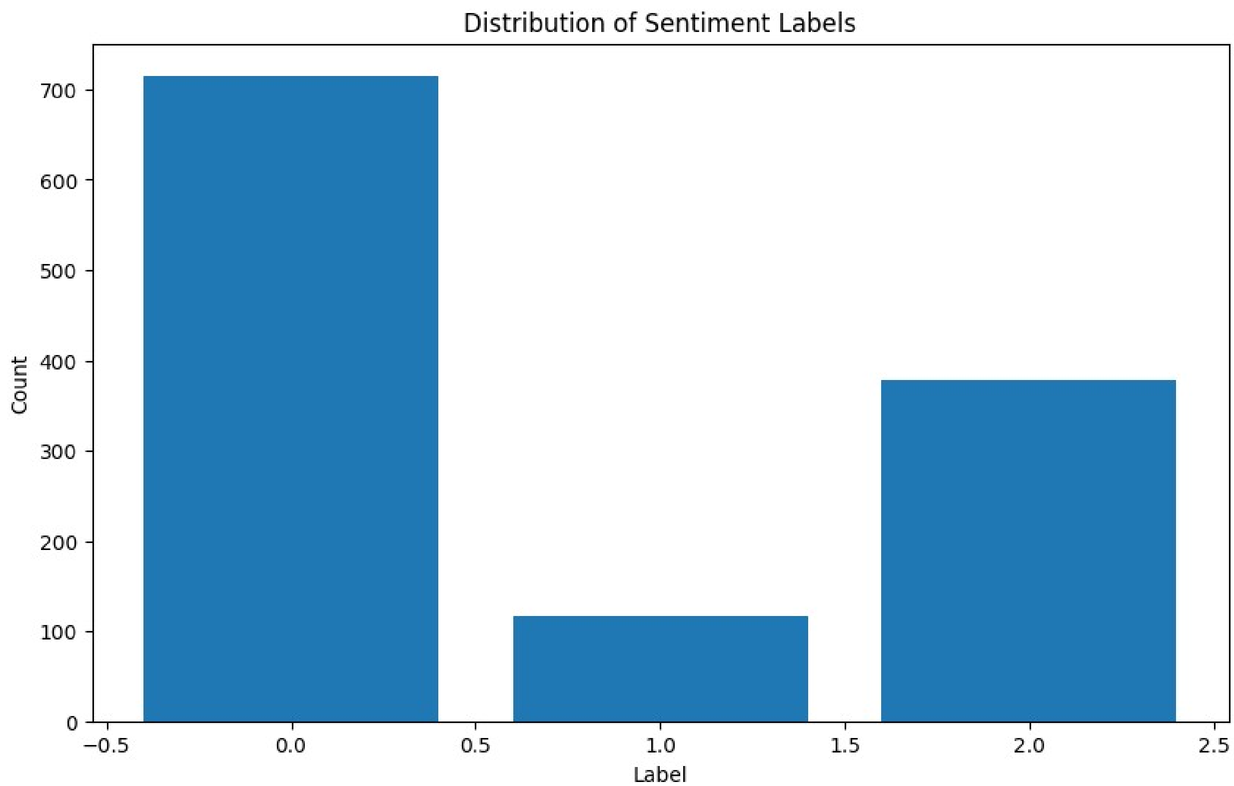




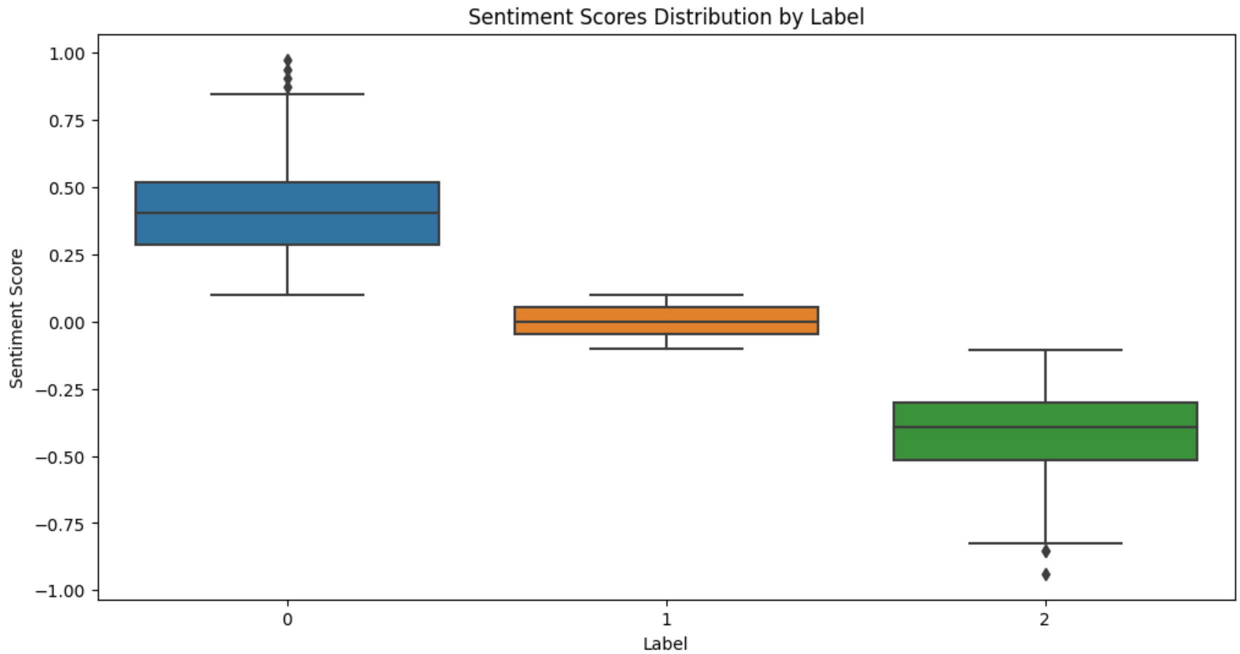

```python
# Analyze sentiment distribution
print("\nSentiment Score Statistics:")
print(combined_df['sentiment_score'].describe())
```

```
Sentiment Score Statistics:
count    1210.000000
mean        0.112594
std         0.399776
min        -0.938000
25%        -0.266000
50%         0.228000
```

```
75%          0.437000
max          0.975000
Name: sentiment_score, dtype: float64

# Analyze sentiment labels
plt.figure(figsize=(10, 6))
label_counts = combined_df['label'].value_counts().sort_index()
plt.bar(label_counts.index, label_counts.values)
plt.title('Distribution of Sentiment Labels')
plt.xlabel('Label')
plt.ylabel('Count')
plt.show()
```




```
# Create box plot to show sentiment score distribution for each label
plt.figure(figsize=(12, 6))
sns.boxplot(x='label', y='sentiment_score', data=combined_df)
plt.title('Sentiment Scores Distribution by Label')
plt.xlabel('Label')
plt.ylabel('Sentiment Score')
plt.show()
```

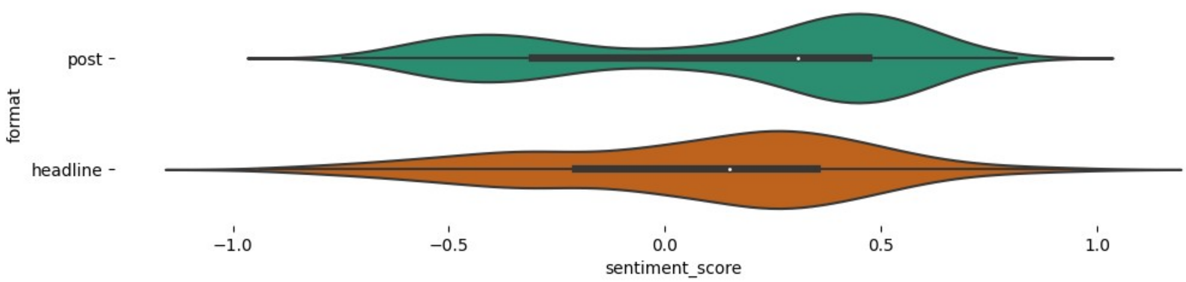



```
# Create violin plot for more detailed distribution
plt.figure(figsize=(12, 6))
sns.violinplot(x='label', y='sentiment_score', data=combined_df)
plt.title('Detailed Sentiment Scores Distribution by Label')
plt.xlabel('Label')
plt.ylabel('Sentiment Score')
plt.show()
```

```python
# Calculate summary statistics for each label
print("\nSentiment Score Statistics by Label:")
print("-" * 40)
for label in sorted(combined_df['label'].unique()):
    scores = combined_df[combined_df['label'] == label]
['sentiment_score']
    print(f"\nLabel {label}:")
    print(f"Min: {scores.min():.3f}")
    print(f"Max: {scores.max():.3f}")
    print(f"Mean: {scores.mean():.3f}")
    print(f"Median: {scores.median():.3f}")
    print(f"Std Dev: {scores.std():.3f}")
```

```
Sentiment Score Statistics by Label:
----------------------------------------

Label 0:
Min: 0.101
Max: 0.975
Mean: 0.406
Median: 0.406
Std Dev: 0.160

Label 1:
Min: -0.100
Max: 0.100
Mean: 0.003
Median: 0.000
Std Dev: 0.056

Label 2:
Min: -0.938
Max: -0.105
Mean: -0.408
Median: -0.392
Std Dev: 0.157
```

```python
# Analyze sentiment score distribution wrt format
figsize = (12, 1.2 * len(combined_df['format'].unique()))
plt.figure(figsize=figsize)
sns.violinplot(combined_df, x='sentiment_score', y='format',
inner='box', palette='Dark2')
sns.despine(top=True, right=True, bottom=True, left=True)
```



2. Model Training

```
import torch
from torch.utils.data import Dataset, DataLoader
from transformers import AutoTokenizer,
AutoModelForSequenceClassification, AdamW
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from tqdm import tqdm
from sklearn.preprocessing import LabelEncoder

# Custom dataset class
class FinancialDataset(Dataset):
    def __init__(self, texts, aspects, sentiments, tokenizer,
max_length=128):
        self.texts = texts
        self.aspects = aspects
        self.sentiments = sentiments
        self.tokenizer = tokenizer
        self.max_length = max_length

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

    def __getitem__(self, idx):
        text = str(self.texts[idx])

        encoding = self.tokenizer(
            text,
            add_special_tokens=True,
            max_length=self.max_length,
            padding='max_length',
            truncation=True,
            return_tensors='pt'
        )

        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
```

```
            'aspect_label': torch.tensor(self.aspects[idx],
dtype=torch.long),
            'sentiment_label': torch.tensor(self.sentiments[idx],
dtype=torch.long)
        }

# Initialize FinBERT tokenizer and model
tokenizer = AutoTokenizer.from_pretrained('ProsusAI/finbert')
model = AutoModelForSequenceClassification.from_pretrained(
    'ProsusAI/finbert',
    num_labels=num_aspects,  # Set to number of aspects
    problem_type="single_label_classification",
    ignore_mismatched_sizes=True
)
```

```
Some weights of BertForSequenceClassification were not initialized
from the model checkpoint at ProsusAI/finbert and are newly
initialized because the shapes did not match:
- classifier.weight: found shape torch.Size([3, 768]) in the
checkpoint and torch.Size([4, 768]) in the model instantiated
- classifier.bias: found shape torch.Size([3]) in the checkpoint and
torch.Size([4]) in the model instantiated
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
```

```
# Prepare datasets for training, validation, and testing
train_dataset = FinancialDataset(
    train_df['sentence'].values,
    train_df['aspect_encoded'].values,
    train_df['label'].values,
    tokenizer
)

validate_dataset = FinancialDataset(
    validate_df['sentence'].values,
    validate_df['aspect_encoded'].values,
    validate_df['label'].values,
    tokenizer
)

test_dataset = FinancialDataset(
    test_df['sentence'].values,
    test_df['aspect_encoded'].values,
    test_df['label'].values,
    tokenizer
)

# Create DataLoader instances
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
```

```
validate_loader = DataLoader(validate_dataset, batch_size=16)
test_loader = DataLoader(test_dataset, batch_size=16)
```

Aspect Prediction Model: Chose FinBERT, a transformer model pre-trained on financial text, as the base model for aspect classification. The model predicts the main aspect category for each sentence. The reason for using FinBERT is its domain-specific pretraining, which allows it to effectively understand financial context compared to a generic language model like BERT. Fine-tuned the model on our dataset to adapt it to the specific task.

```
# Training function
def train_model(model, train_loader, test_loader, epochs=5):
    device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
    print(f"Using device: {device}")

    model.to(device)

    optimizer = AdamW(model.parameters(), lr=2e-5)

    # Training loop
    for epoch in range(epochs):
        model.train()
        total_loss = 0
        progress_bar = tqdm(train_loader, desc=f'Epoch {epoch + 1}')

        for batch in progress_bar:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            aspect_labels = batch['aspect_label'].to(device)

            optimizer.zero_grad()

            outputs = model(
                input_ids=input_ids,
                attention_mask=attention_mask,
                labels=aspect_labels
            )

            loss = outputs.loss
            total_loss += loss.item()

            loss.backward()
            optimizer.step()

            progress_bar.set_postfix({'loss': total_loss /
len(train_loader)})

        print(f"\nEpoch {epoch + 1} average loss: {total_loss /
len(train_loader)}")
```

```python
        # Evaluation after each epoch
        evaluate_model(model, test_loader, device)

# Evaluation function
def evaluate_model(model, test_loader, device):
    model.eval()
    aspect_predictions = []
    aspect_true = []

    with torch.no_grad():
        for batch in test_loader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            aspect_labels = batch['aspect_label'].to(device)

            outputs = model(
                input_ids=input_ids,
                attention_mask=attention_mask
            )

            _, predicted = torch.max(outputs.logits, 1)

            aspect_predictions.extend(predicted.cpu().numpy())
            aspect_true.extend(aspect_labels.cpu().numpy())
    # Print classification report
    print("\nClassification Report:")
    print(classification_report(
        aspect_true,
        aspect_predictions,
        target_names=aspect_encoder.classes_
    ))

    # Plot confusion matrix
    plt.figure(figsize=(12, 8))
    cm = confusion_matrix(aspect_true, aspect_predictions)
    sns.heatmap(
        cm,
        annot=True,
        fmt='d',
        xticklabels=aspect_encoder.classes_,
        yticklabels=aspect_encoder.classes_
    )
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

```python
# Train the model
print("Starting training...")
train_model(model, train_loader, test_loader)
```

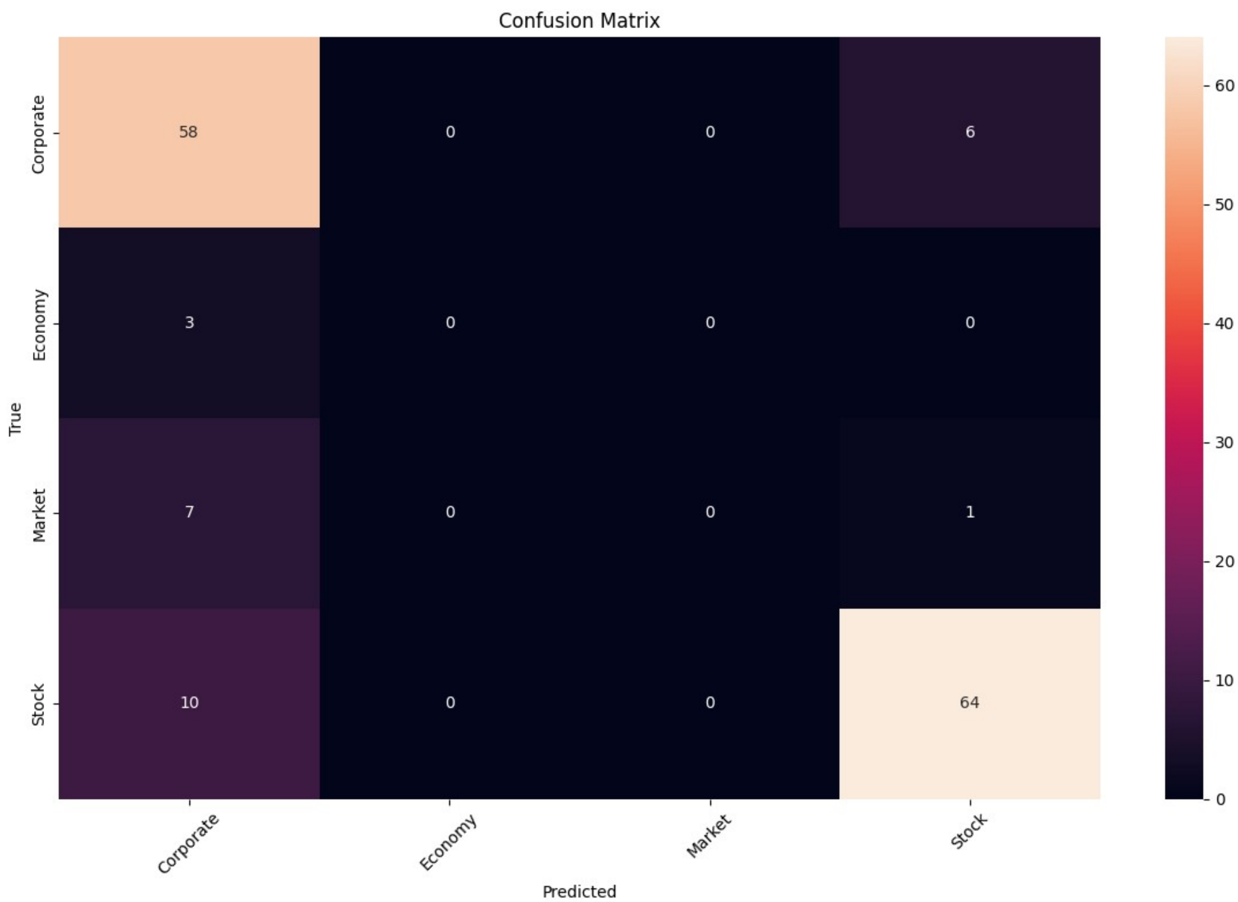
```
/opt/conda/lib/python3.10/site-packages/transformers/
optimization.py:591: FutureWarning: This implementation of AdamW is
deprecated and will be removed in a future version. Use the PyTorch
implementation torch.optim.AdamW instead, or set
`no_deprecation_warning=True` to disable this warning
  warnings.warn(
```

```
Starting training...
Using device: cuda
```

```
Epoch 1: 100%|██████████| 60/60 [00:20<00:00,  2.96it/s, loss=0.735]
```

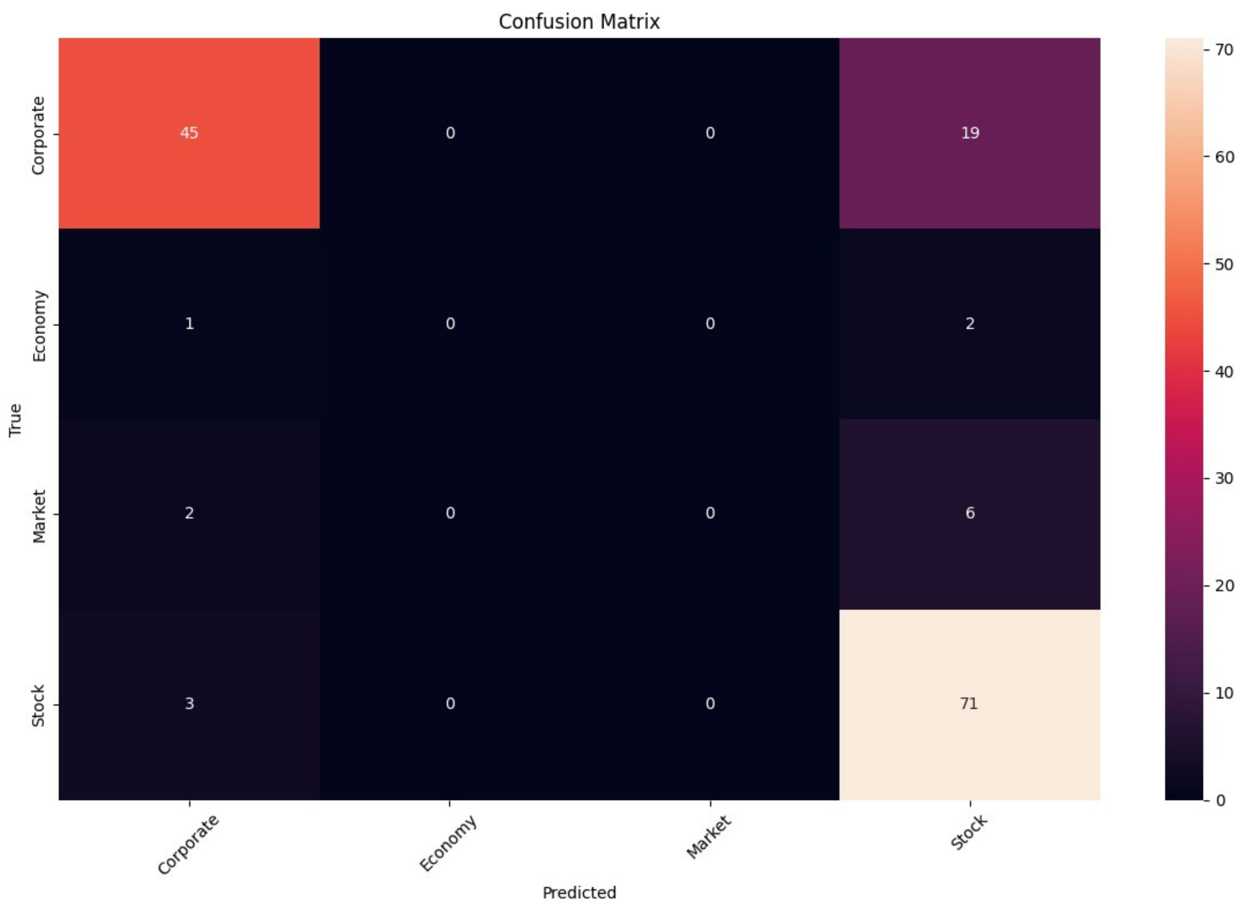
```
Epoch 1 average loss: 0.7345654100179673

Classification Report:
              precision    recall  f1-score   support

   Corporate       0.74      0.91      0.82        64
     Economy       0.00      0.00      0.00         3
      Market       0.00      0.00      0.00         8
       Stock       0.90      0.86      0.88        74

    accuracy                           0.82       149
   macro avg       0.41      0.44      0.42       149
weighted avg       0.77      0.82      0.79       149
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/
_classification.py:1344: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```



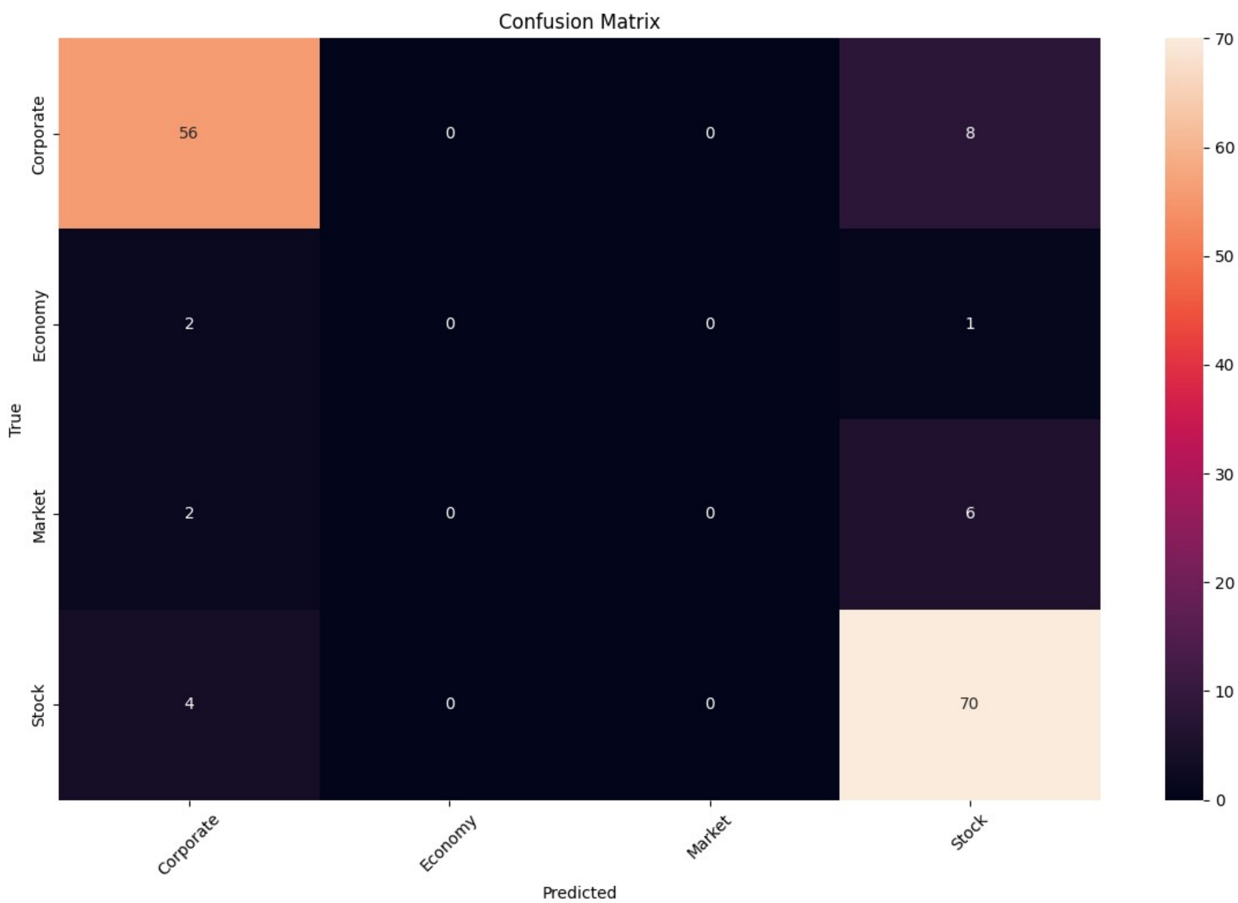

```
Epoch 2: 100%|██████████| 60/60 [00:21<00:00,  2.83it/s, loss=0.4]

Epoch 2 average loss: 0.40045148581266404

Classification Report:
              precision    recall  f1-score   support

   Corporate       0.88      0.70      0.78        64
     Economy       0.00      0.00      0.00         3
      Market       0.00      0.00      0.00         8
       Stock       0.72      0.96      0.83        74

    accuracy                           0.78       149
   macro avg       0.40      0.42      0.40       149
weighted avg       0.74      0.78      0.75       149


/opt/conda/lib/python3.10/site-packages/sklearn/metrics/
_classification.py:1344: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

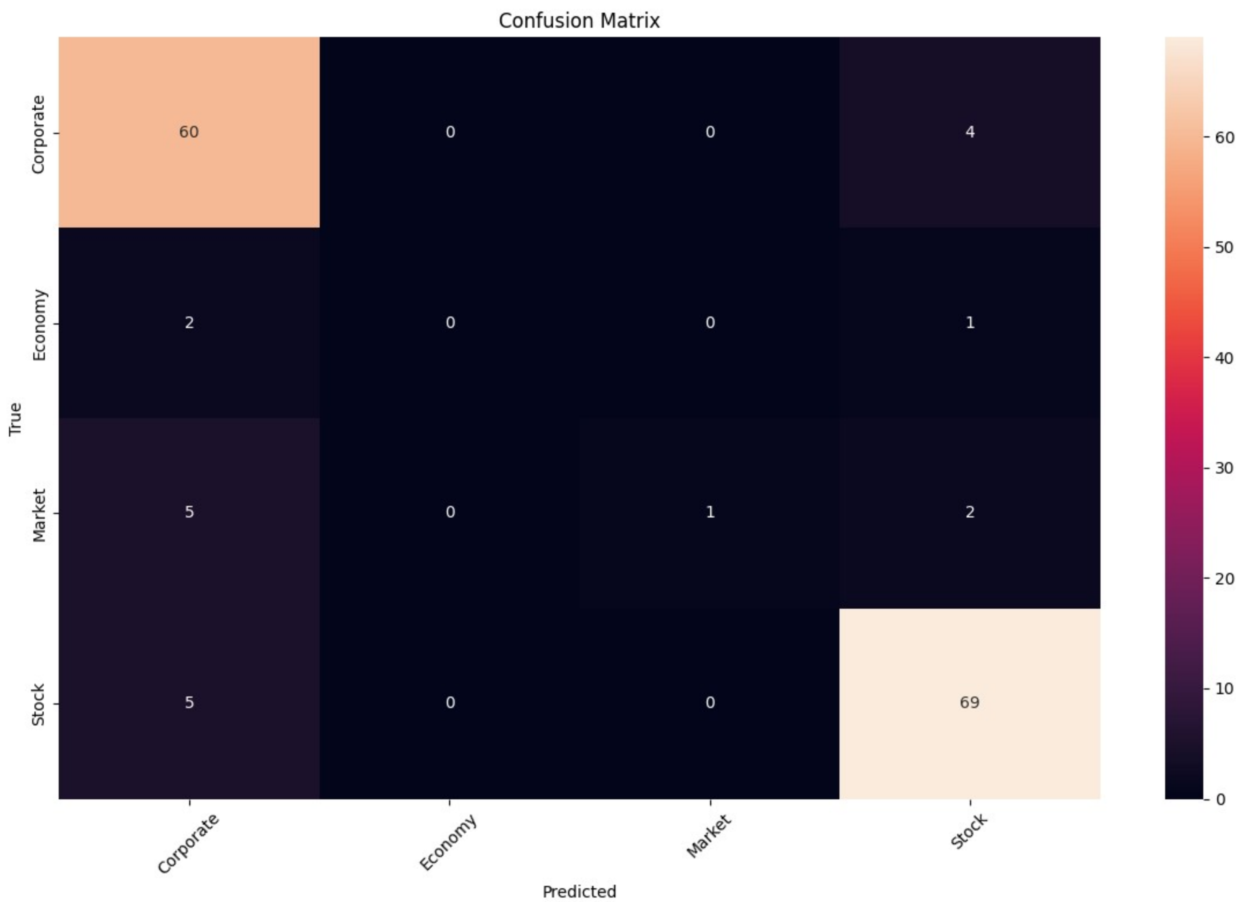



```
Epoch 3: 100%|██████████| 60/60 [00:21<00:00,  2.84it/s, loss=0.254]


Epoch 3 average loss: 0.2542765395094951

Classification Report:
              precision    recall  f1-score   support

   Corporate       0.88      0.88      0.88        64
     Economy       0.00      0.00      0.00         3
      Market       0.00      0.00      0.00         8
```

```
       Stock       0.82      0.95      0.88        74

    accuracy                           0.85       149
   macro avg       0.42      0.46      0.44       149
weighted avg       0.78      0.85      0.81       149


/opt/conda/lib/python3.10/site-packages/sklearn/metrics/
_classification.py:1344: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

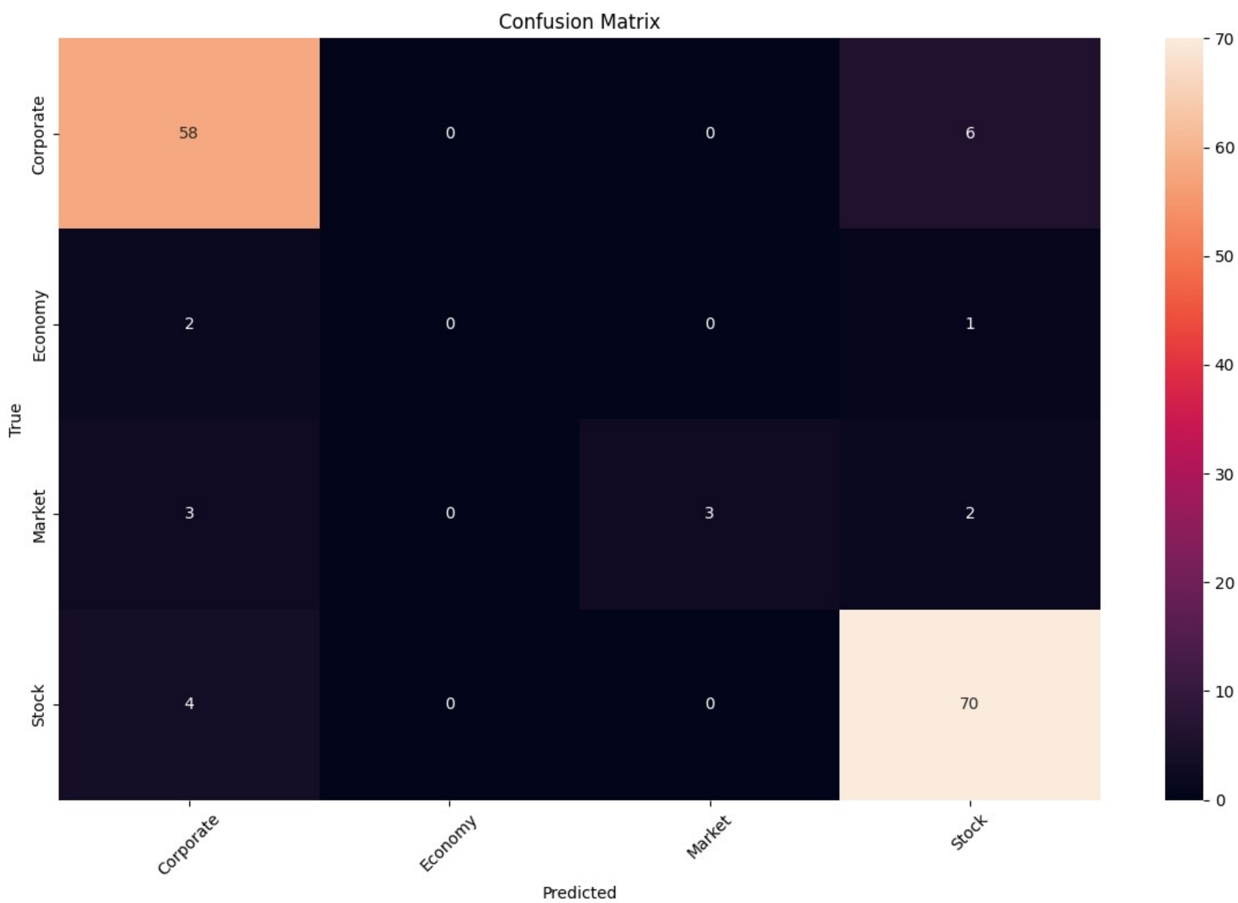


```
Epoch 4: 100%|██████████| 60/60 [00:20<00:00,  2.92it/s, loss=0.152]


Epoch 4 average loss: 0.15153204541032514

Classification Report:
              precision    recall  f1-score   support

   Corporate       0.83      0.94      0.88        64
     Economy       0.00      0.00      0.00         3
      Market       1.00      0.12      0.22         8
       Stock       0.91      0.93      0.92        74

    accuracy                           0.87       149
   macro avg       0.69      0.50      0.51       149
weighted avg       0.86      0.87      0.85       149


/opt/conda/lib/python3.10/site-packages/sklearn/metrics/
_classification.py:1344: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```




```
Epoch 5: 100%|██████████| 60/60 [00:20<00:00,  2.92it/s, loss=0.0883]


Epoch 5 average loss: 0.08828230975195765

Classification Report:
              precision    recall  f1-score   support

   Corporate       0.87      0.91      0.89        64
     Economy       0.00      0.00      0.00         3
      Market       1.00      0.38      0.55         8
       Stock       0.89      0.95      0.92        74

    accuracy                           0.88       149
   macro avg       0.69      0.56      0.59       149
weighted avg       0.87      0.88      0.86       149


/opt/conda/lib/python3.10/site-packages/sklearn/metrics/
_classification.py:1344: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```




```
# Function for making predictions on new text
def predict_aspect_and_sentiment(text, model, tokenizer):
    device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
    model.to(device)
    model.eval()

    # Tokenize input text
    encoding = tokenizer(
        text,
```

```
        add_special_tokens=True,
        max_length=128,
        padding='max_length',
        truncation=True,
        return_tensors='pt'
    )

    # Move inputs to device
    input_ids = encoding['input_ids'].to(device)
    attention_mask = encoding['attention_mask'].to(device)

    # Get predictions
    with torch.no_grad():
        outputs = model(input_ids=input_ids,
attention_mask=attention_mask)
        aspect_pred = torch.argmax(outputs.logits, dim=1)

    # Convert predictions to labels
    predicted_aspect =
aspect_encoder.inverse_transform(aspect_pred.cpu().numpy())[0]

    return predicted_aspect
```

Sentiment Prediction Model: Used the same tokenizer and pre-trained embeddings (from FinBERT) to initialize the sentiment model. Added additional layers for regression to predict the sentiment score and classification to predict sentiment labels. Leveraged the predicted aspect as an additional feature to guide the sentiment prediction.

```
class SentimentDataset(Dataset):
    def __init__(self, texts, aspects, sentiments, tokenizer,
max_length=128):
        self.texts = texts
        self.aspects = aspects
        self.sentiments = sentiments
        self.tokenizer = tokenizer
        self.max_length = max_length

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

    def __getitem__(self, idx):
        text = str(self.texts[idx])
        aspect = str(self.aspects[idx])

        # Combine text and aspect
        combined_input = f"{text} [ASPECT] {aspect}"

        encoding = self.tokenizer(
            combined_input,
```

```
            add_special_tokens=True,
            max_length=self.max_length,
            padding='max_length',
            truncation=True,
            return_tensors='pt'
        )

        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'sentiment_label': torch.tensor(self.sentiments[idx],
dtype=torch.float)
        }

# Create datasets for the sentiment model
sentiment_train_dataset = SentimentDataset(
    train_df['sentence'].values,
    train_df['aspect_encoded'].values,
    train_df['label'].values,
    tokenizer
)

sentiment_validate_dataset = SentimentDataset(
    validate_df['sentence'].values,
    validate_df['aspect_encoded'].values,
    validate_df['label'].values,
    tokenizer
)

sentiment_test_dataset = SentimentDataset(
    test_df['sentence'].values,
    test_df['aspect_encoded'].values,
    test_df['label'].values,
    tokenizer
)

# Create DataLoader instances
sentiment_train_loader = DataLoader(sentiment_train_dataset,
batch_size=16, shuffle=True)
sentiment_validate_loader = DataLoader(sentiment_validate_dataset,
batch_size=16)
sentiment_test_loader = DataLoader(sentiment_test_dataset,
batch_size=16)

# Define the sentiment model
class SentimentModel(nn.Module):
    def __init__(self, pretrained_model):
        super(SentimentModel, self).__init__()
        self.bert = AutoModel.from_pretrained(pretrained_model)
        self.regressor = nn.Linear(self.bert.config.hidden_size, 1)
```

```
    def forward(self, input_ids, attention_mask):
        outputs = self.bert(input_ids=input_ids,
attention_mask=attention_mask)
        pooled_output = outputs.pooler_output
        sentiment_score = self.regressor(pooled_output)
        return sentiment_score

sentiment_model = SentimentModel('ProsusAI/finbert')

# Define training function for sentiment model
def train_sentiment_model(model, train_loader, test_loader, epochs=3):
    device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
    model.to(device)
    optimizer = AdamW(model.parameters(), lr=2e-5)
    criterion = nn.MSELoss()  # Using Mean Squared Error for
regression

    for epoch in range(epochs):
        model.train()
        total_loss = 0
        progress_bar = tqdm(train_loader, desc=f"Epoch {epoch + 1}")

        for batch in progress_bar:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            sentiment_labels = batch['sentiment_label'].to(device)

            optimizer.zero_grad()
            outputs = model(input_ids, attention_mask)
            loss = criterion(outputs.squeeze(), sentiment_labels)
            total_loss += loss.item()

            loss.backward()
            optimizer.step()
            progress_bar.set_postfix({'loss': total_loss /
len(train_loader)})

        print(f"Epoch {epoch + 1} average loss: {total_loss /
len(train_loader)}")

        # Evaluation
        evaluate_sentiment_model(model, test_loader, device)

# Evaluate function for sentiment model
def evaluate_sentiment_model(model, test_loader, device):
    model.eval()
    predictions = []
    true_labels = []
```

```
    with torch.no_grad():
        for batch in test_loader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            sentiment_labels = batch['sentiment_label'].to(device)

            outputs = model(input_ids, attention_mask)
            predictions.extend(outputs.squeeze().cpu().numpy())
            true_labels.extend(sentiment_labels.cpu().numpy())

    mse = mean_squared_error(true_labels, predictions)
    print(f"Mean Squared Error: {mse:.4f}")

train_sentiment_model(sentiment_model, sentiment_train_loader,
sentiment_test_loader, epochs=5)

/opt/conda/lib/python3.10/site-packages/transformers/
optimization.py:591: FutureWarning: This implementation of AdamW is
deprecated and will be removed in a future version. Use the PyTorch
implementation torch.optim.AdamW instead, or set
`no_deprecation_warning=True` to disable this warning
  warnings.warn(
Epoch 1: 100%|██████████| 60/60 [00:20<00:00,  2.87it/s, loss=0.55]

Epoch 1 average loss: 0.5497872099280358
Mean Squared Error: 0.4691

Epoch 2: 100%|██████████| 60/60 [00:20<00:00,  2.87it/s, loss=0.298]

Epoch 2 average loss: 0.2975017582376798
Mean Squared Error: 0.4178

Epoch 3: 100%|██████████| 60/60 [00:20<00:00,  2.90it/s, loss=0.159]

Epoch 3 average loss: 0.15939903970186908
Mean Squared Error: 0.5082

Epoch 4: 100%|██████████| 60/60 [00:20<00:00,  2.91it/s, loss=0.0807]

Epoch 4 average loss: 0.08070989610472074
Mean Squared Error: 0.4534

Epoch 5: 100%|██████████| 60/60 [00:20<00:00,  2.88it/s, loss=0.0591]

Epoch 5 average loss: 0.05910083274357021
Mean Squared Error: 0.4968
```

3. Model Testing and Inference

Pipeline Integration: Combined the aspect prediction model and sentiment prediction model into a single pipeline. First, the aspect prediction model identifies the main aspect for a sentence. The output is passed to the sentiment model to predict both the sentiment score and sentiment label.

```
def predict_pipeline_with_label(text, aspect_model, sentiment_model, 
tokenizer):
    """
    Pipeline to predict aspect, sentiment score, and label.
    Label:
      - 1 if -0.1 <= sentiment_score <= 0.1 (neutral)
      - 0 if sentiment_score < -0.1 (negative)
      - 2 if sentiment_score > 0.1 (positive)
    """
    # Step 1: Predict the aspect
    device = torch.device('cuda' if torch.cuda.is_available() else 
'cpu')
    aspect_model.to(device)
    sentiment_model.to(device)

    aspect_model.eval()
    sentiment_model.eval()

    # Tokenize input for aspect prediction
    encoding = tokenizer(
        text,
        add_special_tokens=True,
        max_length=128,
        padding='max_length',
        truncation=True,
        return_tensors='pt'
    )

    input_ids = encoding['input_ids'].to(device)
    attention_mask = encoding['attention_mask'].to(device)

    with torch.no_grad():
        aspect_outputs = aspect_model(input_ids=input_ids, 
attention_mask=attention_mask)
        predicted_aspect_id = torch.argmax(aspect_outputs.logits, 
dim=1).item()
        predicted_aspect = 
aspect_encoder.inverse_transform([predicted_aspect_id])[0]

    # Step 2: Predict the sentiment score using the aspect
    combined_input = f"{text} [ASPECT] {predicted_aspect}"
    sentiment_encoding = tokenizer(
```

```
        combined_input,
        add_special_tokens=True,
        max_length=128,
        padding='max_length',
        truncation=True,
        return_tensors='pt'
    )

    sentiment_input_ids = sentiment_encoding['input_ids'].to(device)
    sentiment_attention_mask =
sentiment_encoding['attention_mask'].to(device)

    with torch.no_grad():
        sentiment_score = sentiment_model(sentiment_input_ids,
sentiment_attention_mask).item()

    # Step 3: Determine the label based on the sentiment score
    if -0.1 <= sentiment_score <= 0.1:
        label = 1  # Neutral
    elif sentiment_score < -0.1:
        label = 0  # Negative
    else:
        label = 2  # Positive

    return predicted_aspect, sentiment_score, label

sample_texts = [
    "Tesla reports record quarterly deliveries",
    "Barclays Stock shoots up after me joining the company",
    "Amazon announces new acquisition deal"
]


print("\nPipeline Predictions:")
for text in sample_texts:
    aspect, sentiment, label = predict_pipeline_with_label(text,
model, sentiment_model, tokenizer)
    print(f"\nText: {text}")
    print(f"Predicted Aspect: {aspect}")
    print(f"Predicted Sentiment Score: {sentiment:.2f}")
    print(f"Predicted Label: {label} (0=Negative, 1=Neutral,
2=Positive)")
```

```
Pipeline Predictions:

Text: Tesla reports record quarterly deliveries
Predicted Aspect: Corporate
Predicted Sentiment Score: 0.02
Predicted Label: 1 (0=Negative, 1=Neutral, 2=Positive)
```

```
Text: Barclays Stock shoots up after me joining the company
Predicted Aspect: Stock
Predicted Sentiment Score: -0.04
Predicted Label: 1 (0=Negative, 1=Neutral, 2=Positive)

Text: Amazon announces new acquisition deal
Predicted Aspect: Corporate
Predicted Sentiment Score: 0.03
Predicted Label: 1 (0=Negative, 1=Neutral, 2=Positive)
```

```
# # %% [code] {"execution":{"iopub.status.busy":"2024-11-
25T03:42:07.467770Z","iopub.execute_input":"2024-11-
25T03:42:07.468058Z","iopub.status.idle":"2024-11-
25T03:42:07.965836Z","shell.execute_reply.started":"2024-11-
25T03:42:07.468029Z","shell.execute_reply":"2024-11-
25T03:42:07.964896Z"}}
# from sklearn.metrics import mean_squared_error,r2_score

# class MultitaskFinancialModel(nn.Module):
#     def __init__(self, pretrained_model='ProsusAI/finbert',
num_aspects=4):
#         super(MultitaskFinancialModel, self).__init__()
#         self.bert = AutoModel.from_pretrained(pretrained_model)
#         hidden_size = self.bert.config.hidden_size

#         # Aspect classification head
#         self.aspect_classifier = nn.Sequential(
#             nn.Dropout(0.1),
#             nn.Linear(hidden_size, hidden_size),
#             nn.ReLU(),
#             nn.Linear(hidden_size, num_aspects)
#         )

#         # Sentiment regression head
#         self.sentiment_regressor = nn.Sequential(
#             nn.Dropout(0.1),
#             nn.Linear(hidden_size, hidden_size),
#             nn.ReLU(),
#             nn.Linear(hidden_size, 1),
#             nn.Tanh()  # Output between -1 and 1
#         )

#     def forward(self, input_ids, attention_mask):
#         outputs = self.bert(input_ids=input_ids,
attention_mask=attention_mask)
#         pooled_output = outputs.last_hidden_state[:, 0, :]  # Use
[CLS] token

#         aspect_logits = self.aspect_classifier(pooled_output)
#         sentiment_score = self.sentiment_regressor(pooled_output)
```

```
#         return aspect_logits, sentiment_score

# class FinancialDataset(Dataset):
#     def __init__(self, texts, aspects, sentiments, tokenizer,
max_length=128):
#         self.texts = texts
#         self.aspects = aspects
#         self.sentiments = sentiments
#         self.tokenizer = tokenizer
#         self.max_length = max_length

#     def __len__(self):
#         return len(self.texts)

#     def __getitem__(self, idx):
#         text = str(self.texts[idx])

#         encoding = self.tokenizer(
#             text,
#             add_special_tokens=True,
#             max_length=self.max_length,
#             padding='max_length',
#             truncation=True,
#             return_tensors='pt'
#         )

#         return {
#             'input_ids': encoding['input_ids'].flatten(),
#             'attention_mask': encoding['attention_mask'].flatten(),
#             'aspect_label': torch.tensor(self.aspects[idx],
dtype=torch.long),
#             'sentiment_score': torch.tensor(self.sentiments[idx],
dtype=torch.float)
#         }

# def train_multitask_model(model, train_loader, test_loader,
epochs=10):
#     device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
#     print(f"Using device: {device}")

#     model.to(device)
#     optimizer = AdamW(model.parameters(), lr=2e-5)

#     # Loss functions
#     aspect_criterion = nn.CrossEntropyLoss()
#     sentiment_criterion = nn.MSELoss()

#     for epoch in range(epochs):
```

```
#         model.train()
#         total_loss = 0
#         progress_bar = tqdm(train_loader, desc=f'Epoch {epoch + 1}')

#         for batch in progress_bar:
#             input_ids = batch['input_ids'].to(device)
#             attention_mask = batch['attention_mask'].to(device)
#             aspect_labels = batch['aspect_label'].to(device)
#             sentiment_scores = batch['sentiment_score'].to(device)

#             optimizer.zero_grad()

#             # Forward pass
#             aspect_logits, predicted_sentiment = model(input_ids,
attention_mask)

#             # Calculate losses
#             aspect_loss = aspect_criterion(aspect_logits,
aspect_labels)
#             sentiment_loss =
sentiment_criterion(predicted_sentiment.squeeze(), sentiment_scores)

#             # Combined loss (you can adjust the weights)
#             loss = aspect_loss + sentiment_loss

#             loss.backward()
#             optimizer.step()

#             total_loss += loss.item()
#             progress_bar.set_postfix({'loss': total_loss /
len(train_loader)})

#         print(f"\nEpoch {epoch + 1} average loss: {total_loss /
len(train_loader)}")
#         evaluate_multitask_model(model, test_loader, device)

# def evaluate_multitask_model(model, test_loader, device):
#     model.eval()
#     aspect_predictions = []
#     sentiment_predictions = []
#     aspect_true = []
#     sentiment_true = []

#     with torch.no_grad():
#         for batch in test_loader:
#             input_ids = batch['input_ids'].to(device)
#             attention_mask = batch['attention_mask'].to(device)

#             aspect_logits, predicted_sentiment = model(input_ids,
```

```
attention_mask)

#             # Get predictions
#             _, aspect_pred = torch.max(aspect_logits, 1)

#             aspect_predictions.extend(aspect_pred.cpu().numpy())
#
sentiment_predictions.extend(predicted_sentiment.cpu().squeeze().numpy
())
#             aspect_true.extend(batch['aspect_label'].cpu().numpy())
#
sentiment_true.extend(batch['sentiment_score'].cpu().numpy())

#     # Print metrics
#     print("\nAspect Classification Report:")
#     print(classification_report(aspect_true, aspect_predictions))

#     print("\nSentiment Regression Metrics:")
#     mse = mean_squared_error(sentiment_true, sentiment_predictions)
#     # mse=((sentiment_true-sentiment_predictions)**2).mean(axis)
#     r2 = r2_score(sentiment_true, sentiment_predictions)
#     print(f"Mean Squared Error: {mse:.4f}")
#     print(f"R² Score: {r2:.4f}")

# def predict_aspect_and_sentiment(text, model, tokenizer,
aspect_encoder):
#     device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
#     model.to(device)
#     model.eval()

#     encoding = tokenizer(
#         text,
#         add_special_tokens=True,
#         max_length=128,
#         padding='max_length',
#         truncation=True,
#         return_tensors='pt'
#     )

#     input_ids = encoding['input_ids'].to(device)
#     attention_mask = encoding['attention_mask'].to(device)

#     with torch.no_grad():
#         aspect_logits, sentiment_score = model(input_ids,
attention_mask)
#         aspect_pred = torch.argmax(aspect_logits, dim=1)

#     predicted_aspect =
```

```
aspect_encoder.inverse_transform(aspect_pred.cpu().numpy())[0]
#     predicted_sentiment = sentiment_score.cpu().numpy()[0][0]

#     return predicted_aspect, predicted_sentiment

# # Initialize and train the model
# tokenizer = AutoTokenizer.from_pretrained('ProsusAI/finbert')
# model =
MultitaskFinancialModel(num_aspects=len(aspect_encoder.classes_))

# # Create datasets with sentiment scores
# train_dataset = FinancialDataset(
#     X_train.values,
#     y_aspect_train.values,
#     y_sentiment_train.values,
#     tokenizer
# )

# test_dataset = FinancialDataset(
#     X_test.values,
#     y_aspect_test.values,
#     y_sentiment_test.values,
#     tokenizer
# )

# # Create data loaders
# train_loader = DataLoader(train_dataset, batch_size=16,
shuffle=True)
# test_loader = DataLoader(test_dataset, batch_size=16)

# # %% [code] {"execution":{"iopub.status.busy":"2024-11-
25T03:42:07.967395Z","iopub.execute_input":"2024-11-
25T03:42:07.967656Z","iopub.status.idle":"2024-11-
25T03:45:11.151725Z","shell.execute_reply.started":"2024-11-
25T03:42:07.967631Z","shell.execute_reply":"2024-11-
25T03:45:11.150815Z"}}
# # Train the model
# train_multitask_model(model, train_loader, test_loader)

# # Make predictions
# text = "Tesla reports record quarterly deliveries"
# aspect, sentiment = predict_aspect_and_sentiment(text, model,
tokenizer, aspect_encoder)
# print(f"Text: {text}")
# print(f"Predicted aspect: {aspect}")
# print(f"Predicted sentiment score: {sentiment:.2f}")
```

Attempted Multitask Model for Aspect and Sentiment Prediction

Model Overview Objective: Develop a multitask model capable of: Predicting the main aspect of a financial text. Regressing the sentiment score to quantify sentiment intensity. Architecture: Pretrained Base Model: ProsusAI/finbert (fine-tuned for financial domain text). Aspect Classification Head: A fully connected neural network with ReLU activation for multi-class classification. Sentiment Regression Head: A fully connected neural network with Tanh activation to predict sentiment scores between -1 and 1. Dataset: Input: Textual data from the financial dataset. Outputs: Aspect labels: Encoded as integers for classification. Sentiment scores: Real values for regression. Training Process: Used separate CrossEntropyLoss for aspect classification and MSELoss for sentiment regression. Combined loss function for joint optimization. Optimized with AdamW optimizer and a learning rate of 2e-5. Results and Challenges Performance Metrics: Aspect Classification: Metrics: Precision, Recall, F1-Score (via classification report). Sentiment Regression: Metrics: Mean Squared Error (MSE) and R^2 score. Observed Issues: The multitask nature of the model resulted in high error rates: Sentiment Regression: Poor R^2 scores, indicating the model failed to capture the relationship between inputs and sentiment scores. Aspect Classification: Subpar precision and recall due to overfitting or lack of sufficient signal for aspect differentiation. Combined training might have led to conflicting gradients, degrading the performance of both tasks.