a) Q1. The A share company list contains 4,654 different companies while the "News" data include massive 103,7035 news. 'Company name' is a much better measure during the noise removal than the 'company full name'. The 'company full name' itself has contained company name, industry, and "股份有限公司". That could lead to noise to the analysis and likely to leave irrelevant news as the full name has provided more irrelevant information.

1. Similarity Approach:

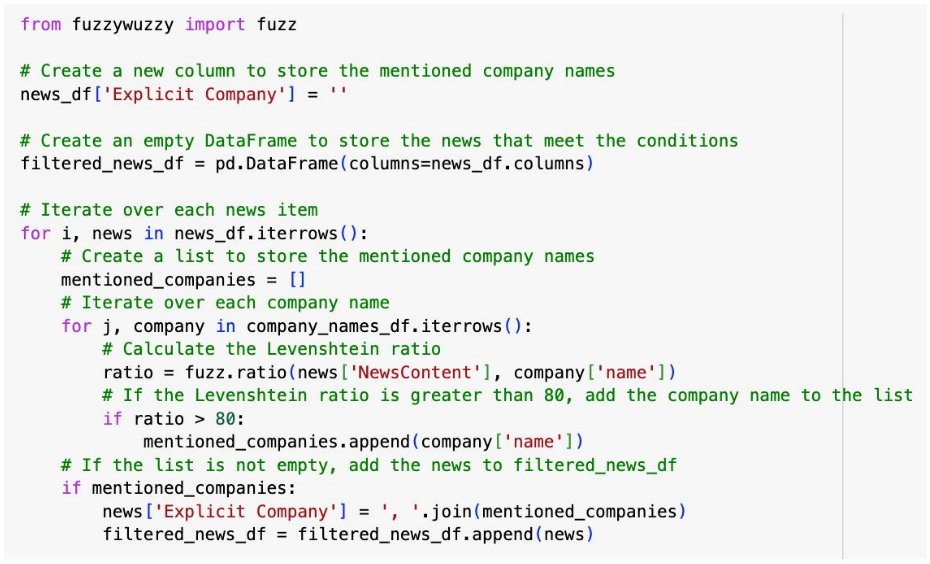
If the company name does not appear exactly as it is in the list in the news, or if there are spelling errors, using similarity matching could be a good approach. In this case, we can use a technique called "fuzzy matching," such as Levenshtein distance.

The Levenshtein distance between two strings a, b (of length $|a|$ and $|b|$ respectively) is given by $\operatorname{lev}(a,b)$ where

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}\big(\operatorname{tail}(a), \operatorname{tail}(b)\big) & \text{if } \operatorname{head}(a) = \operatorname{head}(b), \\ 1 + \min \begin{cases} \operatorname{lev}\big(\operatorname{tail}(a), b\big) \\ \operatorname{lev}\big(a, \operatorname{tail}(b)\big) \\ \operatorname{lev}\big(\operatorname{tail}(a), \operatorname{tail}(b)\big) \end{cases} & \text{otherwise} \end{cases}$$

In the code segment splits the news content into words and then calculates the Levenshtein ratio between each news content and the company name. If the ratio is greater than 0.8, the word is considered sufficiently similar to the company name, and the company name is added to the mentioned company list.

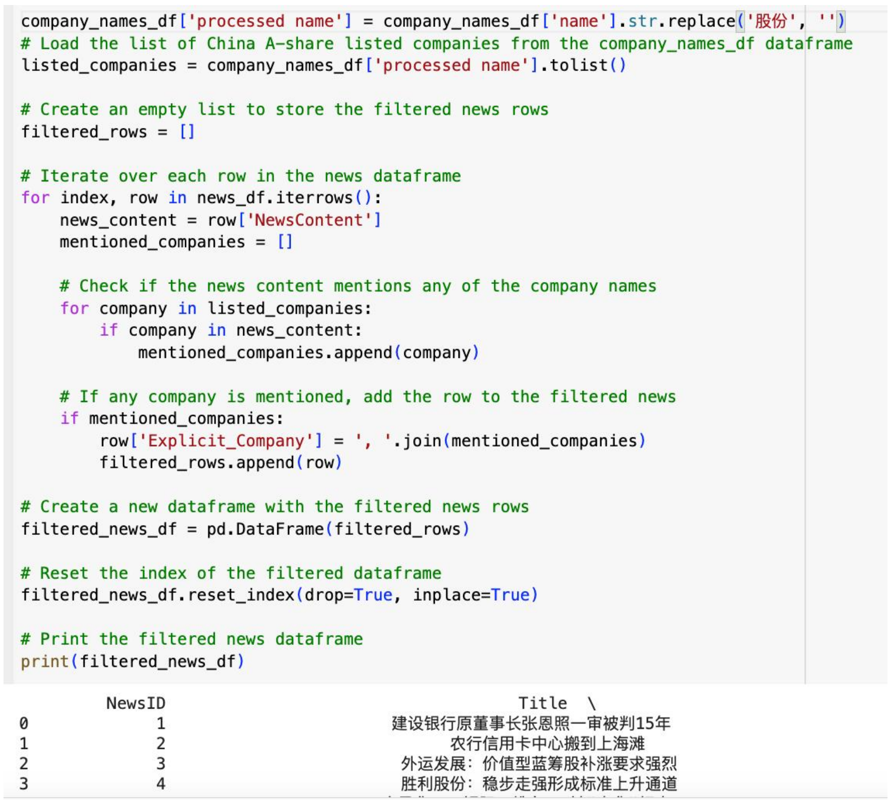
However, this method has a high computational complexity and has taken a long time for 103,7035 rows. And these method could also wrongly consider the news content contain the relevant information about listed companies but actually not.



```python
from fuzzywuzzy import fuzz

# Create a new column to store the mentioned company names
news_df['Explicit Company'] = ''

# Create an empty DataFrame to store the news that meet the conditions
filtered_news_df = pd.DataFrame(columns=news_df.columns)

# Iterate over each news item
for i, news in news_df.iterrows():
    # Create a list to store the mentioned company names
    mentioned_companies = []
    # Iterate over each company name
    for j, company in company_names_df.iterrows():
        # Calculate the Levenshtein ratio
        ratio = fuzz.ratio(news['NewsContent'], company['name'])
        # If the Levenshtein ratio is greater than 80, add the company name to the list
        if ratio > 80:
            mentioned_companies.append(company['name'])
    # If the list is not empty, add the news to filtered_news_df
    if mentioned_companies:
        news['Explicit Company'] = ', '.join(mentioned_companies)
        filtered_news_df = filtered_news_df.append(news)
```

2. Name Matching Approach

Similarity approach does not quite deliver our goal, as many irrelevant information fails to be removed. We change into name matching approach using “company name” rather than “full name”. The accuracy is largely improved as we know the filtered news will only contain listed A share company names. It is much faster and accurate approach even though it seems to be the “dummy” approach. Company name typically include industry information and structure of the company such as “科技, 食品，生态，股份”. “股份” indicates the structure so is not what we expect the find the exact matching. So we removed “股份” from the company name before starting the exact matching.



```
company_names_df['processed name'] = company_names_df['name'].str.replace('股份', '')
# Load the list of China A-share listed companies from the company_names_df dataframe
listed_companies = company_names_df['processed name'].tolist()

# Create an empty list to store the filtered news rows
filtered_rows = []

# Iterate over each row in the news dataframe
for index, row in news_df.iterrows():
    news_content = row['NewsContent']
    mentioned_companies = []

    # Check if the news content mentions any of the company names
    for company in listed_companies:
        if company in news_content:
            mentioned_companies.append(company)

    # If any company is mentioned, add the row to the filtered news
    if mentioned_companies:
        row['Explicit_Company'] = ', '.join(mentioned_companies)
        filtered_rows.append(row)

# Create a new dataframe with the filtered news rows
filtered_news_df = pd.DataFrame(filtered_rows)

# Reset the index of the filtered dataframe
filtered_news_df.reset_index(drop=True, inplace=True)

# Print the filtered news dataframe
print(filtered_news_df)
```

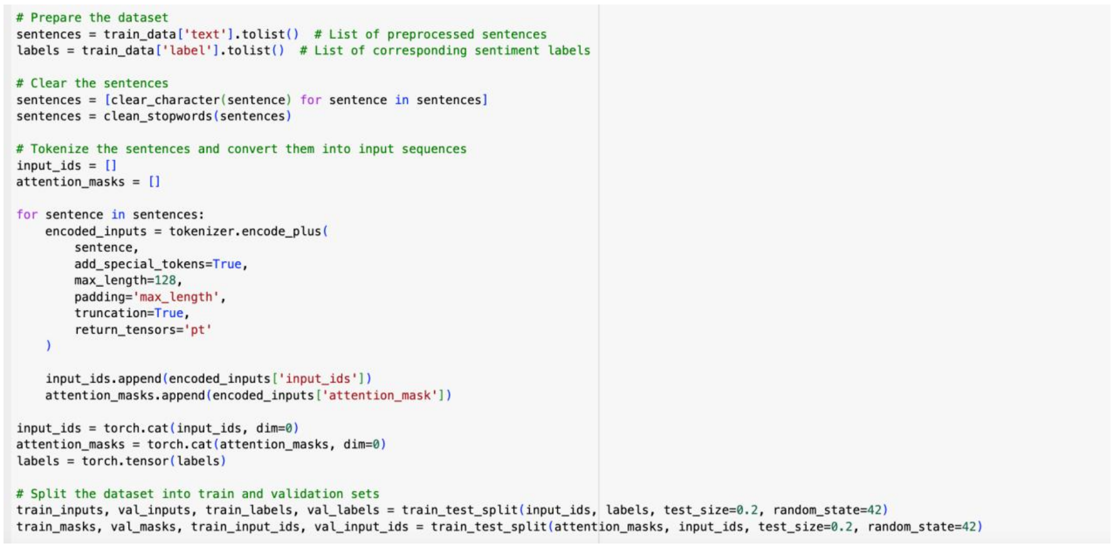
```
        NewsID                                  Title  \
0            1                     建设银行原董事长张恩照一审被判15年
1            2                         农行信用卡中心搬到上海滩
2            3                     外运发展：价值型蓝筹股补涨要求强烈
3            4                     胜利股份：稳步走强形成标准上升通道
```

b) filter rate = # filtered news/# total news = 462,464/103,7035 = 44.5948%

c) Q2. This is a sentiment analysis problem for financial news in Chinese.

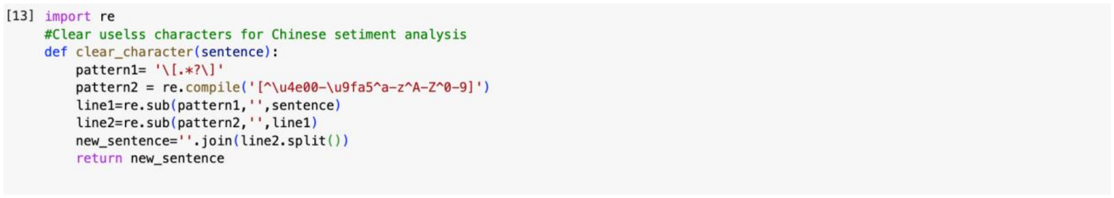
1) To tackle sentiment analysis in Chinese financial news, we initially started with a pre-trained BERT model specifically designed for the Chinese language (https://huggingface.co/bert-base-chinese). However, the results obtained from this model were not entirely ideal, as the sentiment labels often missed the mark when we performed manual checks.

2) So we obtained a well labeled Chinese financial news data set including 5,000 news and trained the Pre-trained BERT Chinese model.

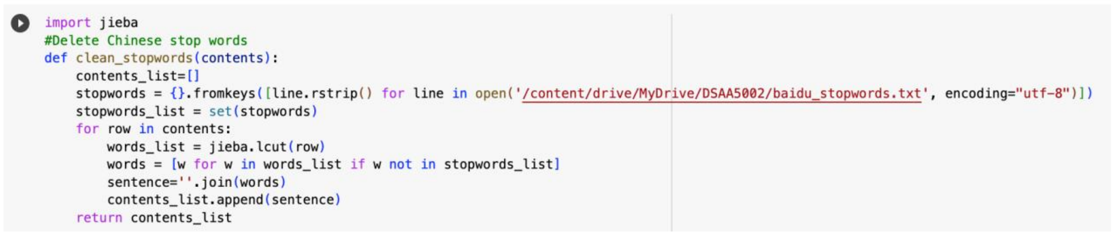


```python
# Prepare the dataset
sentences = train_data['text'].tolist()  # List of preprocessed sentences
labels = train_data['label'].tolist()  # List of corresponding sentiment labels

# Clear the sentences
sentences = [clear_character(sentence) for sentence in sentences]
sentences = clean_stopwords(sentences)

# Tokenize the sentences and convert them into input sequences
input_ids = []
attention_masks = []

for sentence in sentences:
    encoded_inputs = tokenizer.encode_plus(
        sentence,
        add_special_tokens=True,
        max_length=128,
        padding='max_length',
        truncation=True,
        return_tensors='pt'
    )

    input_ids.append(encoded_inputs['input_ids'])
    attention_masks.append(encoded_inputs['attention_mask'])

input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)

# Split the dataset into train and validation sets
train_inputs, val_inputs, train_labels, val_labels = train_test_split(input_ids, labels, test_size=0.2, random_state=42)
train_masks, val_masks, train_input_ids, val_input_ids = train_test_split(attention_masks, input_ids, test_size=0.2, random_state=42)
```

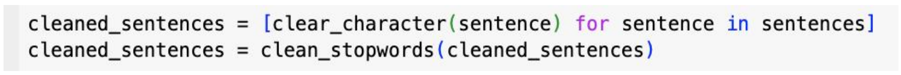
At the same time, I have also used Jieba to perform text preprocessing to remove irrelevant notation and delete Chinese stop words according to ‘baidu’ stop word list.



```python
[13] import re
     #Clear uselss characters for Chinese setiment analysis
     def clear_character(sentence):
         pattern1= '\[.*?\]'
         pattern2 = re.compile('[^\u4e00-\u9fa5^a-z^A-Z^0-9]')
         line1=re.sub(pattern1,'',sentence)
         line2=re.sub(pattern2,'',line1)
         new_sentence=''.join(line2.split())
         return new_sentence
```



```python
import jieba
#Delete Chinese stop words
def clean_stopwords(contents):
    contents_list=[]
    stopwords = {}.fromkeys([line.rstrip() for line in open('/content/drive/MyDrive/DSAA5002/baidu_stopwords.txt', encoding="utf-8")])
    stopwords_list = set(stopwords)
    for row in contents:
        words_list = jieba.lcut(row)
        words = [w for w in words_list if w not in stopwords_list]
        sentence=''.join(words)
        contents_list.append(sentence)
    return contents_list
```

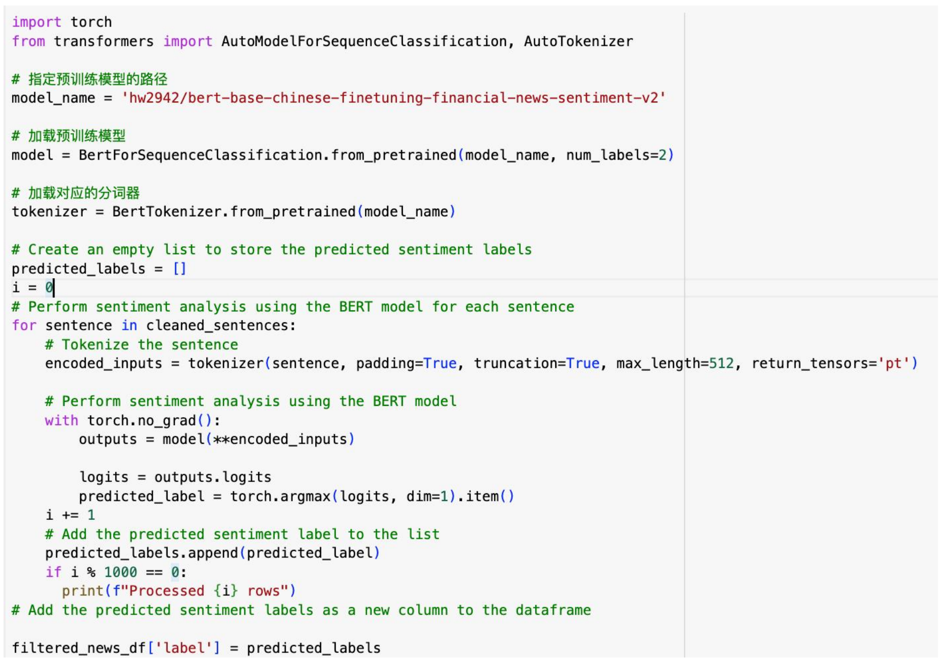


```python
cleaned_sentences = [clear_character(sentence) for sentence in sentences]
cleaned_sentences = clean_stopwords(cleaned_sentences)
```

However, even with trained model, I found that the label accuracy is still not very accurate when I checked the results. To address this, I decided to provide more labeled data to train the model. I collected and trained the model with an additional 5,000 well-labeled financial news data. One challenge I encountered is that the training data set of 5,000 samples is relatively small compared to the total of 462,464 news articles that need to be labeled. This limitation makes it difficult to significantly improve the labeling accuracy on such a large dataset. Additionally, training the model with more data has increased its size, which is already causing slow processing times with the current 5,000 trained samples. As a result, the accuracy of the model's predictions is not as great as desired, and the processing time has become excessively long due to the increased size of the model.

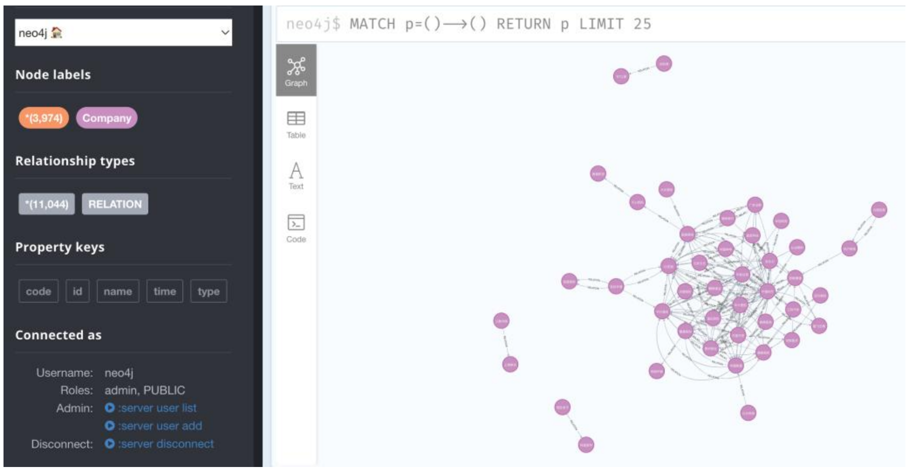
3) So I started to look for a fine-tuned model rather than training the model and applied bert-base-chinese-finetuning-financial-news-sentiment-v2 from hugging face(https://huggingface.co/hw2942/bert-base-chinese-finetuning-financial-news-sentiment-v2). Since this model produced 3 types of classification, negative, neutral, and

positive. Neutral news are also considered to be negative for its company stock in this case. This is an fine-tuned model and has proved to deliver relatively accurate label. One advantage of using a fine-tuned model is that it has already been trained on a large amount of data and fine-tuned for the specific task of sentiment analysis in financial news. This means that the model has learned patterns and features that are relevant to this domain, leading to relatively accurate labeling. By utilizing this fine-tuned model, you can benefit from its accuracy and leverage its ability to classify news sentiment into multiple categories. This approach can save you time and effort compared to training your own model from scratch, especially considering the large amount of data that needs to be labeled.



```python
import torch
from transformers import AutoModelForSequenceClassification, AutoTokenizer

# 指定预训练模型的路径
model_name = 'hw2942/bert-base-chinese-finetuning-financial-news-sentiment-v2'

# 加载预训练模型
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=2)

# 加载对应的分词器
tokenizer = BertTokenizer.from_pretrained(model_name)

# Create an empty list to store the predicted sentiment labels
predicted_labels = []
i = 0
# Perform sentiment analysis using the BERT model for each sentence
for sentence in cleaned_sentences:
    # Tokenize the sentence
    encoded_inputs = tokenizer(sentence, padding=True, truncation=True, max_length=512, return_tensors='pt')

    # Perform sentiment analysis using the BERT model
    with torch.no_grad():
        outputs = model(**encoded_inputs)

        logits = outputs.logits
        predicted_label = torch.argmax(logits, dim=1).item()
    i += 1
    # Add the predicted sentiment label to the list
    predicted_labels.append(predicted_label)
    if i % 1000 == 0:
      print(f"Processed {i} rows")
# Add the predicted sentiment labels as a new column to the dataframe

filtered_news_df['label'] = predicted_labels
```

a) Q3. Knowledge Graph includes 3974 company nodes and 11,044 relationships with types of compete, cooperate, dispute, invest, same-industry, supply.



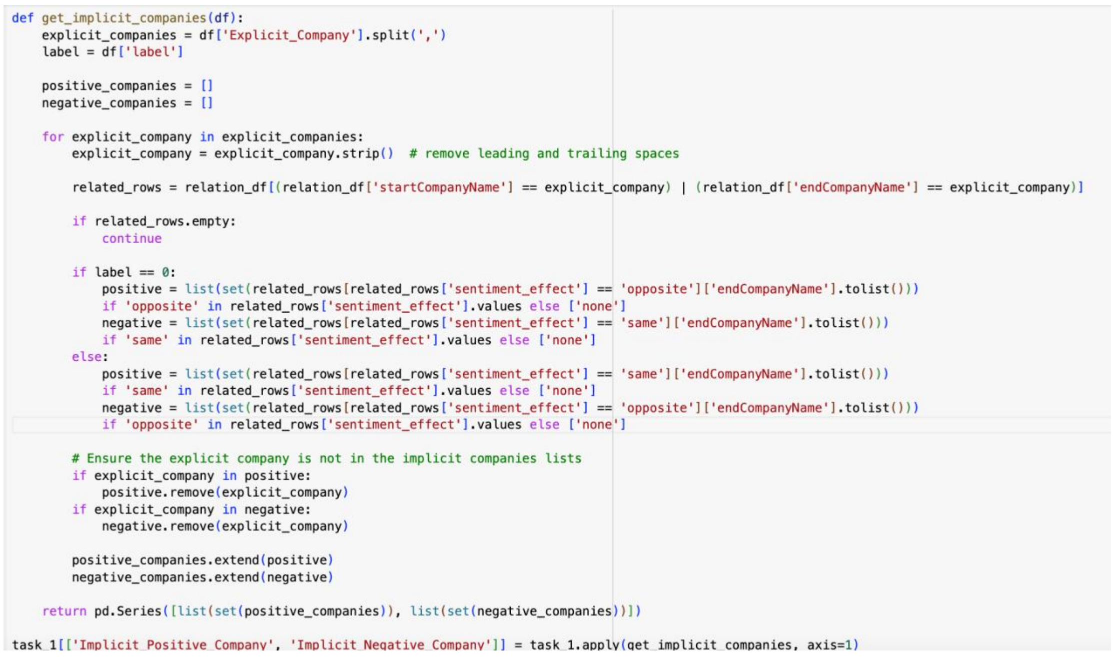
```
neo4j$ MATCH p=()⟶() RETURN p LIMIT 25
```

Code: nowledgeGraph.ipyn

Q4.
All types of relationships are indicated by either ‘opposite’ or the ‘same’. We apply a if else statement to illustrated the implicit positive/negative relationship as explained in below logic chart. Explicit company already has its own column. So they should not be reflected in implicit company. On the other hand, explicit company could be a list of companies and we have ensured the designed function can process it.

The function is designed to handle explicit companies that can be a list of companies by splitting them on commas. The function ensures that this list of explicit companies is processed correctly.

label	Explicit Company	Relationship	Implicit Company
positive(1)	positive(1)	opposite(1)	negative(0)
negative(0)	negative(0)	opposite(1)	positive(1)
positive(1)	positive(1)	same(0)	positive(1)
negative(0)	negative(0)	same(0)	negative(0)



```python
def get_implicit_companies(df):
    explicit_companies = df['Explicit_Company'].split(',')
    label = df['label']

    positive_companies = []
    negative_companies = []

    for explicit_company in explicit_companies:
        explicit_company = explicit_company.strip()  # remove leading and trailing spaces

        related_rows = relation_df[(relation_df['startCompanyName'] == explicit_company) | (relation_df['endCompanyName'] == explicit_company)]

        if related_rows.empty:
            continue

        if label == 0:
            positive = list(set(related_rows[related_rows['sentiment_effect'] == 'opposite']['endCompanyName'].tolist()))
            if 'opposite' in related_rows['sentiment_effect'].values else ['none']
            negative = list(set(related_rows[related_rows['sentiment_effect'] == 'same']['endCompanyName'].tolist()))
            if 'same' in related_rows['sentiment_effect'].values else ['none']
        else:
            positive = list(set(related_rows[related_rows['sentiment_effect'] == 'same']['endCompanyName'].tolist()))
            if 'same' in related_rows['sentiment_effect'].values else ['none']
            negative = list(set(related_rows[related_rows['sentiment_effect'] == 'opposite']['endCompanyName'].tolist()))
            if 'opposite' in related_rows['sentiment_effect'].values else ['none']

        # Ensure the explicit company is not in the implicit companies lists
        if explicit_company in positive:
            positive.remove(explicit_company)
        if explicit_company in negative:
            negative.remove(explicit_company)

        positive_companies.extend(positive)
        negative_companies.extend(negative)

    return pd.Series([list(set(positive_companies)), list(set(negative_companies))])

task_1[['Implicit_Positive_Company', 'Implicit_Negative_Company']] = task_1.apply(get_implicit_companies, axis=1)
```