In [5]:
```
#トピックモデリングによるクラスタリング

#労働災害のデータを読み込み，「災害状況」欄に記載されている発生状況の概要をpandasのDa
import pandas as pd

file_path = "C:¥¥Users¥¥syuhe¥¥Downloads¥¥sisyou_db_h29_01.csv"
data = pd.read_csv(file_path)

# '災害状況'列のデータを取得し、DataFrameに変換
disaster_situation = data['災害状況']

# DataFrameとして表示
df_disaster_situation = pd.DataFrame(disaster_situation)

# 先頭5件を表示
print("先頭5件:")
print(df_disaster_situation.head())

# 末尾5件を表示
print("¥n末尾5件:")
print(df_disaster_situation.tail())
```

```
先頭5件:
                                                災害状況
0                                                NaN
1   工場内で床に置いていたコードに、荷物を抱えていた状態のときに足が引っ掛かり、よろめいて数歩前...
2   倉庫の出入口の階段を荷物（冷凍商品15kgぐらい）を持って下りる際に、階段が凍っていて滑って...
3   会社構内にて車輌の洗車中、足を滑らせ転倒した際に左手をつき、翌朝に左肩の痛みが大きくなり、左...
4                    厩舎2階でバッカン受け入れ作業中、バッカンを落とす穴から落下した。

末尾5件:
                                                災害状況
2691  重機の整備中、待機している台船へ乗船時に、つまずいて高さ1m40cm～50cmの所から転落し、足...
2692  新聞配達中、アパートにて2階と3階の配達を終え、1階に下りる時に誤って最後の段で足を滑らせて...
2693  左手にしびれを感じ、中指にも痛みが出始めたたため検査した結果、手根管症候群と中指ばね指と診断...
2694  塗装場所へ移動する為、5尺の脚立をはしご状態にして、約2.3m程上がった屋上へ上る途中に使用...
2695  入浴介助後、利用者（男性48kg・全介助・車いす）を洗い場から車いすに移動させる際、新人職員が...
```

In [26]:
```
#spaCyから解析モデルとしてjp_ginzaを指定して分かち書き.

import spacy
import pandas as pd

# jp_ginzaモデルを読み込む
nlp = spacy.load("ja_ginza")

# '災害状況'列のテキストを結合して分かち書きする
documents = []
for text in data['災害状況']:
    if pd.notna(text):
        doc = nlp(text)
        tokens = [token.text for token in doc]
        documents.append(tokens)

# 分かち書きの結果を文書5まで表示
```

```
for idx in range(5):
    tokens = documents[idx]
    print(f"文書 {idx + 1}: {tokens}")
```

```
文書 1: ['工場', '内', 'で', '床', 'に', '置い', 'て', 'い', 'た', 'コード', 'に',
'、', '荷物', 'を', '抱え', 'て', 'い', 'た', '状態', 'の', 'とき', 'に', '足',
'が', '引っ掛かり', '、', 'よろめい', 'て', '数', '歩', '前', 'に', '進ん', 'だ',
'のち', '、', '前方', 'に', 'あっ', 'た', '作業', '台', 'に', '衝突', 'し', 'て',
'受傷', 'し', 'た', '。']
文書 2: ['倉庫', 'の', '出入', '口', 'の', '階段', 'を', '荷物', '(', '冷凍', '商
品', '15', 'kg', 'ぐらい', ')', 'を', '持っ', 'て', '下りる', '際', 'に', '、', '階
段', 'が', '凍っ', 'て', 'い', 'て', '滑っ', 'て', '転倒', 'し', '、', '階段', 'を',
'転げ落ち', '(', '4', '段位', ')', '、', '持っ', 'て', 'い', 'た', '荷物', 'を',
'足', 'に', '落とし', 'て', 'しまい', '、', '右足', 'の', '腓骨', 'を', '骨折',
'し', 'た', '。']
文書 3: ['会社', '構内', 'にて', '車輌', 'の', '洗車', '中', '、', '足', 'を', '滑
ら', 'せ', '転倒', 'し', 'た', '際', 'に', '左手', 'を', 'つき', '、', '翌朝', 'に',
'左肩', 'の', '痛み', 'が', '大きく', 'なり', '、', '左肩', '腱', '板', '剥離',
'と', '診断', 'さ', 'れ', 'た', '。']
文書 4: ['厩舎', '2', '階', 'で', 'バッカン', '受け入れ', '作業中', '、', 'バッカ
ン', 'を', '落とす', '穴', 'から', '落下', 'し', 'た', '。']
文書 5: ['勤務先', 'の', '食堂', '施設内', 'で', '、', 'ダンボール', 'を', '束ね',
'て', 'ビニール', 'の', '荷造り', '紐', 'で', '縛り', '結ん', 'だ', '時', '、',
'手', 'が', '滑り', '勢い', 'よく', '壁', 'に', '左手', '小指', 'を', 'ぶつけ',
'腱', 'が', '切れ', 'て', '全治', '1', '〜', '2', 'ヶ月', 'と', 'なっ', 'た', '。',
'¥n']
```

In [27]:

```
#さらに，出現が20文書に満たない単語と50%以上の文書に出現する単語をそれぞれ示し，これ
# 単語の文書出現数を計算

# 単語の出現頻度を計算
word_counts = pd.Series([word for sublist in documents for word in sublist]).value_

# 出現が20文書に満たない単語を取得
rare_words = word_counts[word_counts < 20].index
print('出現が20文書に満たない単語')
print(rare_words)

# 50%以上の文書に出現する単語を取得
common_words = word_counts[word_counts >= 0.5 * len(documents)].index
print('出現が50%以上の文書に出現する単語')
print(common_words)

# 除外する単語のリスト
exclusion_list = set(rare_words) | set(common_words)

#除外したものをfiltered_documentsと置く。
filtered_documents = [[token for token in tokens if token not in exclusion_list] fo
#printで除外したものを出力すると量がとても大きくなってしまうのでここでは出力は行わな
```

```
出現が20文書に満たない単語
Index(['取ら', 'パネル', '積雪', '直後', 'ヘルメット', 'スライサー', '伸ばし', 'フッ
ク', '無理', 'mm',
       ...
       '井戸', '引っぱる', '余裕', '人指し', 'ゆび', '傾眠', '浮腫', 'おこり', '見た
目', 'ギク'],
      dtype='object', length=7184)
出現が50%以上の文書に出現する単語
Index(['、', 'を', 'た', 'に', 'の', 'し', 'て', 'が', '。', 'で', 'と', 'い'], dtyp
e='object')
```

In [11]:

```
#処理された単語群から単語文書行列を作成

from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
```

```
# 分かち書き後のデータを使う
corpus = [' '.join(tokens) for tokens in filtered_documents]

# CountVectorizerを使って単語文書行列を作成
vectorizer = CountVectorizer()
tdm = vectorizer.fit_transform(corpus)

# 列（単語）名を取得
feature_names = vectorizer.get_feature_names_out()

# 単語文書行列をDataFrameに変換
tdm_df = pd.DataFrame(tdm.toarray(), columns=feature_names)

# 先頭5行を表示
print(tdm_df.head())
```

```
   10  15  20  2人  30  3m  40  50  60  cm  ...  除雪  階段  隙間  電源  頭部  顔面
食堂  ￥
0   0   0   0   0   0   0   0   0   0   0 ...   0   0   0   0   0   0   0
1   0   1   0   0   0   0   0   0   0   0 ...   0   3   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   0 ...   0   0   0   0   0   0   0
3   0   0   0   0   0   0   0   0   0   0 ...   0   0   0   0   0   0   0
4   0   0   0   0   0   0   0   0   0   0 ...   0   0   0   0   0   0   1

   駐車  駐車場  骨折
0   0    0   0
1   0    0   1
2   0    0   0
3   0    0   0
4   0    0   0

[5 rows x 566 columns]
```

In [12]:
```
#単語文書行列に20回以上出現したbi-gram（'_'が含まれるtokenはbi-gram）を追加せよ.
#20回以上出現したbi-gramを抽出して新しい列として追加
bi_grams_to_add = [bi_gram for bi_gram, count in zip(feature_names, tdm.sum(axis=0))
for bi_gram in bi_grams_to_add:
    tdm_df[bi_gram] = tdm_df[bi_gram] + tdm_df[bi_gram.replace(' ', '_')]

#新しい列が追加された単語文書行列を表示
print(tdm_df)
```

```
      10  15  20  2人  30  3m  40  50  60  cm  ...  除雪  階段  隙間  電源  頭部  顔
面  食堂  ¥
0       0   0   0   0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
1       0   2   0   0   0   0   0   0   0   0  ...   0   6   0   0   0   0   0
2       0   0   0   0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
3       0   0   0   0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
4       0   0   0   0   0   0   0   0   0   0  ...   0   0   0   0   0   0   2
...    ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ...  ..  ..  ..  ..  ..  ..  ..
2690    0   0   0   0   0   0   2   2   0   0  ...   0   0   0   0   0   0   0
2691    0   0   0   0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
2692    0   0   0   0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0
2693    0   0   0   0   0   2   0   0   0   0  ...   0   0   0   0   0   0   0
2694    0   0   0   0   0   0   0   0   0   0  ...   0   0   0   0   0   0   0

      駐車  駐車場  骨折
0       0    0   0
1       0    0   2
2       0    0   0
3       0    0   0
4       0    0   0
...    ..  ...  ..
2690    0    0   2
2691    0    0   0
2692    0    0   0
2693    0    0   0
2694    0    0   0

[2695 rows x 566 columns]
```

In [21]:

```python
#gensimのLDAモデルを使用してLDAモデルの計算を実行. オプションの値は以下とする)
#model = LdaModel(corpus=[Q2-4で作成した単語文書行列], id2word=dictionary.id2token, 
#作成したLDAモデルのトピックごとに上位10件の単語を表示.

from gensim import corpora, models
from sklearn.feature_extraction.text import CountVectorizer

# 分かち書き後のデータを使う
corpus = [' '.join(tokens) for tokens in filtered_documents]

# CountVectorizerを使って単語文書行列を作成
vectorizer = CountVectorizer()
tdm = vectorizer.fit_transform(corpus)

# 単語文書行列からgensimのcorpora.Dictionaryを作成
dictionary = corpora.Dictionary([doc.split() for doc in corpus])

# 単語文書行列をgensimのcorpora.MmCorpus形式に変換
corpus_gensim = [dictionary.doc2bow(doc.split()) for doc in corpus]

# 空のコレクションを確認してから LDA モデルを構築
if not corpus_gensim:
    print("Error: Empty corpus.")
else:
    # LDAモデルの計算　トピックの数は10とする。
    model = models.LdaModel(corpus=corpus_gensim, id2word=dictionary, chunksize=2000

    # トピックごとに上位10件の単語を表示
    top_words_per_topic = []
    for topic_id in range(model.num_topics):
        words = [word for word, _ in model.show_topic(topic_id, topn=10)]
        top_words_per_topic.append(words)

    # 結果の表示
```

```
    for topic_id, words in enumerate(top_words_per_topic):
        print(f"トピック {topic_id + 1}: {', '.join(words)}")
```

```
トピック 1: 転倒, 足, ため, 際, から, へ, 凍結, 階段, 駐車場, 骨折
トピック 2: さ, 台車, から, れ, (, ), パレット, リフト, 際, 上
トピック 3: 荷台, トラック, 際, 作業中, から, 鉄板, 車, にて, t, 負傷
トピック 4: 痛み, 利用者, なっ, なり, なく, は, も, ため, れ, へ
トピック 5: さ, m, 作業, 約, 高, 負傷, 落下, から, ), (
トピック 6: 際, 左手, 内, 右手, 時, ため, 作業中, 室, 作業, しまい
トピック 7: ため, から, 被災者, 車両, は, 車, れ, き, 衝突, ところ
トピック 8: ), (, ×, は, 1, 2, 約, 3, cm, 段
トピック 9: 際, から, 足, 転倒, 作業, ため, は, 床, 時, バランス
トピック 10: 右手, れ, 機械, 左手, 機, 工場, 内, いる, する, 負傷
```

In [24]:

```
#model.top_topics()によりトピックの抽出を行い, 評価指標として抽出されたトピックのコヒ
from gensim import corpora, models
from sklearn.feature_extraction.text import CountVectorizer

# 分かち書き後のデータを使う
corpus = [' '.join(tokens) for tokens in filtered_documents]

# CountVectorizerを使って単語文書行列を作成
vectorizer = CountVectorizer()
tdm = vectorizer.fit_transform(corpus)

# 単語文書行列からgensimのcorpora.Dictionaryを作成
dictionary = corpora.Dictionary([doc.split() for doc in corpus])

# 各文書をdoc2bowで変換し、コーパスを作成
corpus_gensim = [dictionary.doc2bow(doc.split()) for doc in corpus]

# LDAモデルの計算
model = models.LdaModel(corpus=corpus_gensim, id2word=dictionary, chunksize=2000, al

# model.top_topics()でトピックを抽出
top_topics = model.top_topics(corpus=corpus_gensim, coherence='u_mass', topn=10)

# トピックごとにコヒーレンスを計算して表示
for i, (topic, coherence) in enumerate(top_topics):
    print(f"トピック {i + 1} のコヒーレンス: {coherence}")
```

```
トピック 1 のコヒーレンス: -1.5461003802469016
トピック 2 のコヒーレンス: -1.5699982951686213
トピック 3 のコヒーレンス: -1.6528390310056706
トピック 4 のコヒーレンス: -1.7271288659137674
トピック 5 のコヒーレンス: -1.757653329418783
トピック 6 のコヒーレンス: -1.8368423960889835
トピック 7 のコヒーレンス: -1.92149406976229
トピック 8 のコヒーレンス: -2.0171476178402488
トピック 9 のコヒーレンス: -2.1266763854048407
トピック 10 のコヒーレンス: -2.254601651236475
```