

# 06\_sec\_preprocessing

September 29, 2021

## 1 Word vectors from SEC filings using Gensim: Preprocessing

In this section, we will learn word and phrase vectors from annual SEC filings using gensim to illustrate the potential value of word embeddings for algorithmic trading. In the following sections, we will combine these vectors as features with price returns to train neural networks to predict equity prices from the content of security filings.

In particular, we use a dataset containing over 22,000 10-K annual reports from the period 2013-2016 that are filed by listed companies and contain both financial information and management commentary (see chapter 3 on Alternative Data). For about half of 11K filings for companies that we have stock prices to label the data for predictive modeling

### 1.1 Imports & Settings

```
[1]: import warnings
      warnings.filterwarnings('ignore')

[2]: from dateutil.relativedelta import relativedelta
      from pathlib import Path
      import numpy as np
      import pandas as pd
      from time import time
      from collections import Counter
      import logging
      import spacy

      from gensim.models import Word2Vec
      from gensim.models.word2vec import LineSentence
      from gensim.models.phrases import Phrases, Phraser

[3]: np.random.seed(42)

[4]: def format_time(t):
      m, s = divmod(t, 60)
      h, m = divmod(m, 60)
      return f'{h:02.0f}:{m:02.0f}:{s:02.0f}'
```

### 1.1.1 Logging Setup

```
[5]: logging.basicConfig(
    filename='preprocessing.log',
    level=logging.DEBUG,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    datefmt='%H:%M:%S')
```

## 1.2 Data Download

The data can be downloaded from [here](#). Unzip and move into the **data** folder in the repository's root directory and rename to **filings**.

### 1.2.1 Paths

Each filing is a separate text file and a master index contains filing metadata. We extract the most informative sections, namely - Item 1 and 1A: Business and Risk Factors - Item 7 and 7A: Management's Discussion and Disclosures about Market Risks

The notebook preprocessing shows how to parse and tokenize the text using spaCy, similar to the approach in chapter 14. We do not lemmatize the tokens to preserve nuances of word usage.

We use gensim to detect phrases. The Phrases module scores the tokens and the Phraser class transforms the text data accordingly. The notebook shows how to repeat the process to create longer phrases.

```
[6]: sec_path = Path('.', 'data', 'sec-filings')
    filing_path = sec_path / 'filings'
    sections_path = sec_path / 'sections'
```

```
[7]: if not sections_path.exists():
    sections_path.mkdir(exist_ok=True, parents=True)
```

## 1.3 Identify Sections

```
[8]: for i, filing in enumerate(filing_path.glob('*.txt'), 1):
    if i % 500 == 0:
        print(i, end=' ', flush=True)
    filing_id = int(filing.stem)
    items = {}
    for section in filing.read_text().lower().split('°'):
        if section.startswith('item '):
            if len(section.split()) > 1:
                item = section.split()[1].replace('.', '').replace(':', '').
                ↪replace(',', ' ')
                text = ' '.join([t for t in section.split()[2:]])
                if items.get(item) is None or len(items.get(item)) < len(text):
                    items[item] = text
```

```
txt = pd.Series(items).reset_index()
txt.columns = ['item', 'text']
txt.to_csv(sections_path / (filing.stem + '.csv'), index=False)
```

```
500 1000 1500 2000 2500 3000 3500 4000 4500 5000 5500 6000 6500 7000 7500 8000
8500 9000 9500 10000 10500 11000 11500 12000 12500 13000 13500 14000 14500 15000
15500 16000 16500 17000 17500 18000 18500 19000 19500 20000 20500 21000 21500
22000 22500
```

## 1.4 Parse Sections

Select the following sections:

```
[9]: sections = ['1', '1a', '7', '7a']
```

```
[9]: clean_path = sec_path / 'selected_sections'
if not clean_path.exists():
    clean_path.mkdir(exist_ok=True)
```

```
[14]: nlp = spacy.load('en', disable=['ner'])
nlp.max_length = 6000000
```

```
[16]: vocab = Counter()
t = total_tokens = 0
stats = []

start = time()
to_do = len(list(sections_path.glob('*.csv')))
done = len(list(clean_path.glob('*.csv'))) + 1
for text_file in sections_path.glob('*.csv'):
    file_id = int(text_file.stem)
    clean_file = clean_path / f'{file_id}.csv'
    if clean_file.exists():
        continue
    items = pd.read_csv(text_file).dropna()
    items.item = items.item.astype(str)
    items = items[items.item.isin(sections)]
    if done % 100 == 0:
        duration = time() - start
        to_go = (to_do - done) * duration / done
        print(f'{done:>5}\t{format_time(duration)}\t{total_tokens / duration:.,.
→0f}\t{format_time(to_go)}')

    clean_doc = []
    for _, (item, text) in items.iterrows():
        doc = nlp(text)
        for s, sentence in enumerate(doc.sents):
```

```

        clean_sentence = []
        if sentence is not None:
            for t, token in enumerate(sentence, 1):
                if not any([token.is_stop,
                            token.is_digit,
                            not token.is_alpha,
                            token.is_punct,
                            token.is_space,
                            token.lemma_ == '-PRON-',
                            token.pos_ in ['PUNCT', 'SYM', 'X']]):
                    clean_sentence.append(token.text.lower())
            total_tokens += t
            if len(clean_sentence) > 0:
                clean_doc.append([item, s, ' '.join(clean_sentence)])
    (pd.DataFrame(clean_doc,
                  columns=['item', 'sentence', 'text'])
     .dropna()
     .to_csv(clean_file, index=False))
done += 1

```

100	00:02:38	18,125	09:53:45
200	00:05:36	17,183	10:28:08
300	00:08:30	16,514	10:32:46
400	00:10:57	17,093	10:08:36
500	00:13:21	17,482	09:50:42
600	00:15:56	17,806	09:45:08
700	00:18:33	18,003	09:41:23
800	00:20:46	18,139	09:26:55
900	00:23:07	18,262	09:18:06
1000	00:25:33	18,342	09:12:43
1100	00:27:51	18,425	09:05:06
1200	00:30:27	18,486	09:03:41
1300	00:33:05	18,536	09:02:49
1400	00:35:36	18,579	08:59:47
1500	00:38:15	18,621	08:58:47
1600	00:40:39	18,666	08:54:19
1700	00:42:57	18,714	08:48:44
1800	00:45:36	18,759	08:47:41
1900	00:47:52	18,805	08:42:17
2000	00:50:14	18,853	08:38:10
2100	00:52:23	18,879	08:32:06
2200	00:54:43	18,908	08:28:11
2300	00:57:17	18,908	08:26:20
2400	00:59:48	18,834	08:24:02
2500	01:01:56	18,868	08:18:43
2600	01:04:21	18,898	08:15:43
2700	01:06:23	18,924	08:10:03

2800	01:08:29	18,951	08:05:05
2900	01:10:42	18,981	08:01:03
3000	01:12:49	19,008	07:56:28
3100	01:15:12	19,030	07:53:45
3200	01:17:46	19,052	07:52:11
3300	01:20:04	19,074	07:49:02
3400	01:22:17	19,098	07:45:27
3500	01:24:32	19,117	07:42:03
3600	01:26:45	19,134	07:38:35
3700	01:28:52	19,151	07:34:43
3800	01:31:00	19,167	07:30:58
3900	01:33:15	19,189	07:27:52
4000	01:35:44	19,204	07:25:53
4100	01:37:58	19,217	07:22:49
4200	01:40:21	19,227	07:20:22
4300	01:42:40	19,241	07:17:39
4400	01:45:03	19,250	07:15:15
4500	01:47:19	19,258	07:12:23
4600	01:49:34	19,269	07:09:30
4700	01:51:51	19,277	07:06:45
4800	01:53:55	19,286	07:03:12
4900	01:56:07	19,298	07:00:11
5000	01:58:29	19,305	06:57:47
5100	02:00:38	19,316	06:54:42
5200	02:02:47	19,322	06:51:34
5300	02:04:53	19,328	06:48:21
5400	02:07:18	19,336	06:46:11
5500	02:09:56	19,347	06:44:43
5600	02:12:05	19,354	06:41:42
5700	02:14:05	19,359	06:38:18
5800	02:16:29	19,368	06:36:04
5900	02:18:34	19,378	06:32:57
6000	02:20:58	19,382	06:30:43
6100	02:23:23	19,388	06:28:33
6200	02:25:33	19,396	06:25:44
6300	02:27:41	19,405	06:22:51
6400	02:30:02	19,412	06:20:31
6500	02:32:16	19,418	06:17:53
6600	02:34:22	19,424	06:14:57
6700	02:36:32	19,430	06:12:13
6800	02:38:56	19,432	06:10:01
6900	02:41:05	19,433	06:07:15
7000	02:43:38	19,436	06:05:23
7100	02:46:09	19,439	06:03:27
7200	02:48:23	19,443	06:00:53
7300	02:50:49	19,442	05:58:44
7400	02:53:09	19,444	05:56:24
7500	02:55:20	19,448	05:53:45

7600	02:57:48	19,452	05:51:38
7700	03:00:12	19,451	05:49:26
7800	03:02:39	19,453	05:47:18
7900	03:04:59	19,456	05:44:57
8000	03:06:55	19,460	05:41:51
8100	03:09:12	19,464	05:39:26
8200	03:11:20	19,469	05:36:44
8300	03:13:31	19,473	05:34:09
8400	03:15:45	19,478	05:31:38
8500	03:18:02	19,483	05:29:13
8600	03:20:21	19,488	05:26:52
8700	03:22:28	19,494	05:24:12
8800	03:24:48	19,498	05:21:53
8900	03:27:06	19,504	05:19:31
9000	03:29:28	19,511	05:17:16
9100	03:31:41	19,514	05:14:45
9200	03:33:53	19,518	05:12:14
9300	03:36:15	19,521	05:09:59
9400	03:38:35	19,528	05:07:41
9500	03:40:50	19,534	05:05:14
9600	03:43:02	19,539	05:02:45
9700	03:45:23	19,539	05:00:28
9800	03:47:45	19,541	04:58:12
9900	03:49:56	19,545	04:55:41
10000	03:51:60	19,549	04:53:02
10100	03:54:13	19,553	04:50:36
10200	03:56:37	19,558	04:48:22
10300	03:59:01	19,562	04:46:09
10400	04:01:29	19,566	04:44:00
10500	04:03:49	19,568	04:41:41
10600	04:06:03	19,573	04:39:16
10700	04:08:28	19,577	04:37:04
10800	04:10:45	19,581	04:34:41
10900	04:13:10	19,585	04:32:28
11000	04:15:13	19,588	04:29:51
11100	04:17:46	19,592	04:27:47
11200	04:20:06	19,593	04:25:27
11300	04:22:25	19,596	04:23:09
11400	04:24:29	19,599	04:20:34
11500	04:26:39	19,603	04:18:06
11600	04:29:06	19,605	04:15:54
11700	04:31:25	19,609	04:13:35
11800	04:33:43	19,612	04:11:14
11900	04:35:54	19,613	04:08:48
12000	04:38:21	19,617	04:06:36
12100	04:40:38	19,619	04:04:15
12200	04:43:09	19,621	04:02:05
12300	04:45:18	19,623	03:59:37

12400	04:47:37	19,626	03:57:18
12500	04:49:53	19,629	03:54:57
12600	04:51:59	19,631	03:52:27
12700	04:54:05	19,634	03:49:58
12800	04:56:16	19,636	03:47:33
12900	04:58:43	19,639	03:45:20
13000	05:01:04	19,643	03:43:02
13100	05:03:13	19,646	03:40:36
13200	05:05:34	19,648	03:38:19
13300	05:07:57	19,650	03:36:03
13400	05:10:23	19,652	03:33:49
13500	05:12:43	19,654	03:31:31
13600	05:14:42	19,657	03:28:59
13700	05:17:04	19,658	03:26:42
13800	05:19:11	19,662	03:24:15
13900	05:21:22	19,665	03:21:51
14000	05:23:52	19,668	03:19:40
14100	05:26:12	19,669	03:17:22
14200	05:28:30	19,671	03:15:03
14300	05:30:26	19,674	03:12:30
14400	05:32:47	19,676	03:10:13
14500	05:35:05	19,679	03:07:54
14600	05:37:30	19,682	03:05:39
14700	05:39:50	19,685	03:03:21
14800	05:41:54	19,689	03:00:54
14900	05:44:06	19,692	02:58:32
15000	05:46:16	19,694	02:56:10
15100	05:48:32	19,696	02:53:50
15200	05:50:45	19,698	02:51:29
15300	05:52:56	19,700	02:49:06
15400	05:55:12	19,702	02:46:47
15500	05:57:27	19,704	02:44:27
15600	05:59:48	19,704	02:42:10
15700	06:01:60	19,705	02:39:48
15800	06:04:10	19,707	02:37:27
15900	06:06:17	19,708	02:35:04
16000	06:08:18	19,710	02:32:38
16100	06:10:21	19,713	02:30:14
16200	06:12:48	19,713	02:27:60
16300	06:14:58	19,715	02:25:38
16400	06:17:09	19,714	02:23:18
16500	06:19:26	19,715	02:20:59
16600	06:21:36	19,717	02:18:38
16700	06:23:39	19,719	02:16:15
16800	06:25:57	19,720	02:13:57
16900	06:28:29	19,721	02:11:44
17000	06:30:55	19,723	02:09:29
17100	06:33:19	19,725	02:07:13

17200	06:35:40	19,725	02:04:56
17300	06:37:54	19,727	02:02:37
17400	06:40:24	19,728	02:00:22
17500	06:42:43	19,730	01:58:05
17600	06:44:57	19,733	01:55:45
17700	06:47:14	19,734	01:53:27
17800	06:49:27	19,735	01:51:08
17900	06:51:35	19,737	01:48:47
18000	06:53:39	19,738	01:46:25
18100	06:55:52	19,740	01:44:06
18200	06:58:01	19,741	01:41:46
18300	07:00:07	19,741	01:39:26
18400	07:02:40	19,739	01:37:11
18500	07:04:53	19,737	01:34:52
18600	07:07:11	19,736	01:32:35
18700	07:09:44	19,735	01:30:20
18800	07:12:01	19,733	01:28:02
18900	07:14:23	19,731	01:25:45
19000	07:16:51	19,730	01:23:29
19100	07:19:04	19,730	01:21:10
19200	07:21:27	19,728	01:18:53
19300	07:23:60	19,725	01:16:38
19400	07:26:18	19,726	01:14:20
19500	07:28:50	19,724	01:12:04
19600	07:31:13	19,724	01:09:47
19700	07:33:37	19,722	01:07:29
19800	07:35:47	19,721	01:05:10
19900	07:38:20	19,719	01:02:54
20000	07:40:44	19,719	01:00:37
20100	07:42:56	19,720	00:58:18
20200	07:45:08	19,720	00:55:59
20300	07:47:17	19,720	00:53:39
20400	07:49:17	19,720	00:51:19
20500	07:51:35	19,721	00:49:01
20600	07:53:55	19,720	00:46:43
20700	07:56:16	19,720	00:44:26
20800	07:58:14	19,719	00:42:06
20900	08:00:33	19,719	00:39:48
21000	08:02:53	19,718	00:37:30
21100	08:04:54	19,718	00:35:11
21200	08:07:18	19,717	00:32:54
21300	08:09:40	19,716	00:30:36
21400	08:11:56	19,716	00:28:18
21500	08:14:09	19,716	00:25:60
21600	08:16:32	19,715	00:23:42
21700	08:18:49	19,715	00:21:24
21800	08:21:02	19,714	00:19:06
21900	08:23:18	19,714	00:16:48



22000	08:25:44	19,713	00:14:30
22100	08:28:19	19,711	00:12:13
22200	08:30:33	19,709	00:09:55
22300	08:33:16	19,698	00:07:37
22400	08:35:44	19,696	00:05:19
22500	08:38:06	19,704	00:03:01
22600	08:39:59	19,712	00:00:43

## 1.5 Create ngrams

```
[10]: ngram_path = sec_path / 'ngrams'
stats_path = sec_path / 'corpus_stats'
for path in [ngram_path, stats_path]:
    if not path.exists():
        path.mkdir(parents=True)
```

```
[19]: unigrams = ngram_path / 'ngrams_1.txt'
```

```
[20]: def create_unigrams(min_length=3):
    texts = []
    sentence_counter = Counter()
    vocab = Counter()
    for i, f in enumerate(clean_path.glob('*.csv')):
        if i % 1000 == 0:
            print(i, end=' ', flush=True)
        df = pd.read_csv(f)
        df.item = df.item.astype(str)
        df = df[df.item.isin(sections)]
        sentence_counter.update(df.groupby('item').size().to_dict())
        for sentence in df.text.dropna().str.split().tolist():
            if len(sentence) >= min_length:
                vocab.update(sentence)
                texts.append(' '.join(sentence))

    (pd.DataFrame(sentence_counter.most_common(),
                  columns=['item', 'sentences'])
     .to_csv(stats_path / 'selected_sentences.csv', index=False))
    (pd.DataFrame(vocab.most_common(), columns=['token', 'n'])
     .to_csv(stats_path / 'sections_vocab.csv', index=False))

    unigrams.write_text('\n'.join(texts))
    return [l.split() for l in texts]
```

```
[21]: start = time()
if not unigrams.exists():
    texts = create_unigrams()
else:
```

```

    texts = [l.split() for l in unigrams.open()]
print('\nReading: ', format_time(time() - start))

```

```

0 1000 2000 3000 4000 5000 6000 7000 8000 9000 10000 11000 12000 13000 14000
15000 16000 17000 18000 19000 20000 21000 22000
Reading: 00:04:14

```

```

[22]: def create_ngrams(max_length=3):
        """Using gensim to create ngrams"""

        n_grams = pd.DataFrame()
        start = time()
        for n in range(2, max_length + 1):
            print(n, end=' ', flush=True)

            sentences = LineSentence(ngram_path / f'ngrams_{n - 1}.txt')
            phrases = Phrases(sentences=sentences,
                               min_count=25, # ignore terms with a lower count
                               threshold=0.5, # accept phrases with higher score
                               max_vocab_size=40000000, # prune of less common
                               ↪words to limit memory use
                               delimiter=b' ', # how to join ngram tokens
                               progress_per=50000, # log progress every
                               scoring='npmi')

            s = pd.DataFrame([[k.decode('utf-8'), v] for k, v in phrases.
                               ↪export_phrases(sentences)]),
                              columns=['phrase', 'score']).assign(length=n)

            n_grams = pd.concat([n_grams, s])
            grams = Phraser(phrases)
            sentences = grams[sentences]
            (ngram_path / f'ngrams_{n}.txt').write_text('\n'.join([' '.join(s) for
                               ↪s in sentences]))

            n_grams = n_grams.sort_values('score', ascending=False)
            n_grams.phrase = n_grams.phrase.str.replace('_', ' ')
            n_grams['ngram'] = n_grams.phrase.str.replace(' ', '_')

            n_grams.to_parquet(sec_path / 'ngrams.parquet')

            print('\n\tDuration: ', format_time(time() - start))
            print('\tngrams: {:,d}\n'.format(len(n_grams)))
            print(n_grams.groupby('length').size())

```

```

[ ]: create_ngrams()

```

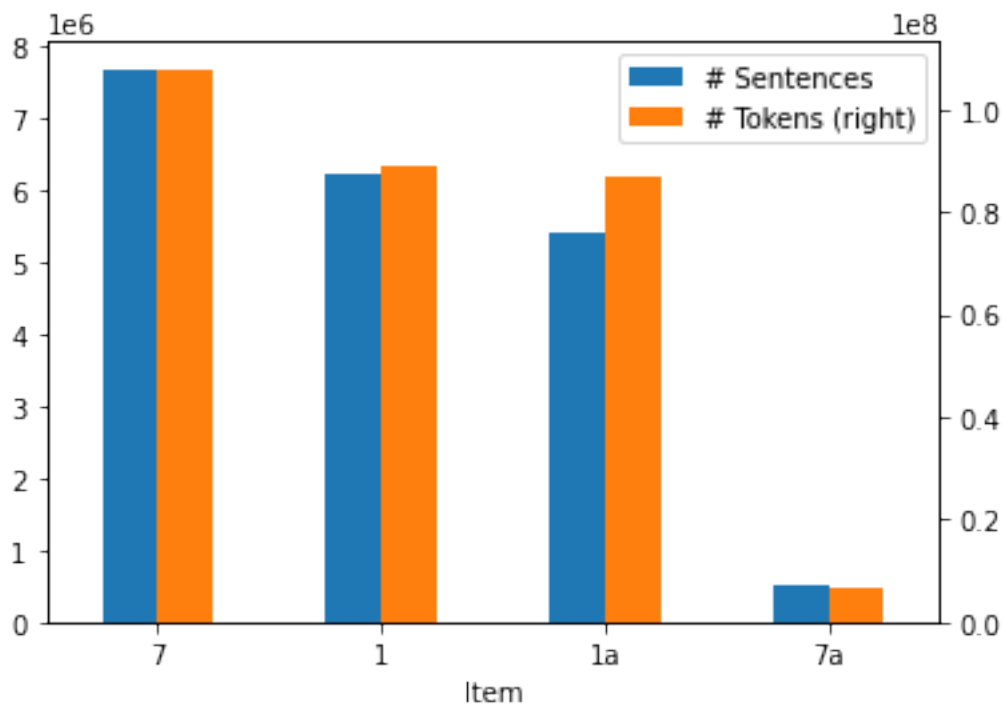
## 1.6 Inspect Corpus

```
[18]: percentiles=np.arange(.1, 1, .1).round(2)
```

```
[11]: nsents, ntokens = Counter(), Counter()
for f in clean_path.glob('*.csv'):
    df = pd.read_csv(f)
    nsents.update({str(k): v for k, v in df.item.value_counts().to_dict().
    ↳items()})
    df['ntokens'] = df.text.str.split().str.len()
    ntokens.update({str(k): v for k, v in df.groupby('item').ntokens.sum().
    ↳to_dict().items()})
```

```
[12]: ntokens = pd.DataFrame(ntokens.most_common(), columns=['Item', '# Tokens'])
nsents = pd.DataFrame(nsents.most_common(), columns=['Item', '# Sentences'])
```

```
[13]: nsents.set_index('Item').join(ntokens.set_index('Item')).plot.
    ↳bar(secondary_y='# Tokens', rot=0);
```



```
[ ]: ngrams = pd.read_parquet(sec_path / 'ngrams.parquet')
```

```
[ ]: ngrams.info()
```

```
[ ]: ngrams.head()
```

```
[ ]: ngrams.score.describe(percentiles=percentiles)
```

```
[ ]: ngrams[ngrams.score>.7].sort_values(['length', 'score']).head(10)
```

```
[15]: vocab = pd.read_csv(stats_path / 'sections_vocab.csv').dropna()
```

```
[16]: vocab.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 200867 entries, 0 to 200868
Data columns (total 2 columns):
#   Column  Non-Null Count  Dtype
---  -
0   token   200867 non-null    object
1   n        200867 non-null    int64
dtypes: int64(1), object(1)
memory usage: 4.6+ MB
```

```
[19]: vocab.n.describe(percentiles).astype(int)
```

```
[19]: count      200867
      mean         1439
      std         22312
      min           1
      10%           1
      20%           2
      30%           3
      40%           4
      50%           7
      60%          12
      70%          24
      80%          61
      90%         260
      max      2574572
      Name: n, dtype: int64
```

```
[20]: tokens = Counter()
      for l in (ngram_path / 'ngrams_2.txt').open():
          tokens.update(l.split())
```

```
[21]: tokens = pd.DataFrame(tokens.most_common(),
                             columns=['token', 'count'])
```

```
[22]: tokens.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 230112 entries, 0 to 230111
Data columns (total 2 columns):
#   Column   Non-Null Count  Dtype
---  -
0   token    230112 non-null object
1   count    230112 non-null int64
dtypes: int64(1), object(1)
memory usage: 3.5+ MB

```

```
[23]: tokens.head()
```

```

[23]:      token    count
0  million  2340187
1  business  1696732
2  december  1512367
3   company  1490617
4  products  1367413

```

```
[24]: tokens.loc[tokens.token.str.contains('_'), 'count'].describe(percentiles).
      ↪ astype(int)
```

```

[24]: count      29951
      mean         926
      std        9611
      min           1
      10%          26
      20%          31
      30%          37
      40%          46
      50%          61
      60%          85
      70%         131
      80%         237
      90%         666
      max       593859
      Name: count, dtype: int64

```

```
[25]: tokens[tokens.token.str.contains('_')].head(20).to_csv(sec_path /
      ↪ 'ngram_examples.csv', index=False)
```

```
[26]: tokens[tokens.token.str.contains('_')].head(20)
```

```

[26]:      token    count
46      year_ended  593859
64  results_operations  492047
71    table_contents  436034

```

```

78             company_s  412971
85    financial_condition  396164
86             common_stock  387629
107            fair_value  341108
152            united_states  276401
158            cash_flows  266725
168    financial_statements  255115
187            interest_rate  234621
188    approximately_million  234385
199            adversely_affect  227984
223            long_term  203600
238            real_estate  192824
239    material_adverse  192238
240            fiscal_year  192189
243            interest_rates  190754
248            income_tax  186923
267            natural_gas  178765

```

## 1.7 Get returns

```
[27]: DATA_FOLDER = Path('..', 'data')
```

```
[28]: with pd.HDFStore(DATA_FOLDER / 'assets.h5') as store:
      prices = store['quandl/wiki/prices'].adj_close
```

```
[29]: sec = pd.read_csv(sec_path / 'filing_index.csv').rename(columns=str.lower)
      sec.date_filed = pd.to_datetime(sec.date_filed)
```

```
[30]: sec.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22631 entries, 0 to 22630
Data columns (total 11 columns):
#   Column          Non-Null Count  Dtype
---  -
0   cik              22631 non-null  int64
1   company_name     22631 non-null  object
2   form_type        22631 non-null  object
3   date_filed       22631 non-null  datetime64[ns]
4   edgar_link       22631 non-null  object
5   quarter          22631 non-null  int64
6   ticker           22631 non-null  object
7   sic              22461 non-null  object
8   exchange         20619 non-null  object
9   hits             22555 non-null  object
10  year             22631 non-null  int64
dtypes: datetime64[ns](1), int64(3), object(7)

```

memory usage: 1.9+ MB

```
[31]: idx = pd.IndexSlice
```

```
[32]: first = sec.date_filed.min() + relativedelta(months=-1)
last = sec.date_filed.max() + relativedelta(months=1)
prices = (prices
          .loc[idx[first:last, :]]
          .unstack().resample('D')
          .ffill()
          .dropna(how='all', axis=1)
          .filter(sec.ticker.unique()))
```

```
[33]: sec = sec.loc[sec.ticker.isin(prices.columns), ['ticker', 'date_filed']]

price_data = []
for ticker, date in sec.values.tolist():
    target = date + relativedelta(months=1)
    s = prices.loc[date: target, ticker]
    price_data.append(s.iloc[-1] / s.iloc[0] - 1)

df = pd.DataFrame(price_data,
                  columns=['returns'],
                  index=sec.index)
```

```
[34]: df.returns.describe()
```

```
[34]: count      11101.000000
mean           0.022839
std            0.126137
min            -0.555556
25%            -0.032213
50%             0.017349
75%             0.067330
max             1.928826
Name: returns, dtype: float64
```

```
[35]: sec['returns'] = price_data
sec.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11375 entries, 0 to 22629
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  -
0   ticker      11375 non-null  object
1   date_filed  11375 non-null  datetime64[ns]
2   returns     11101 non-null  float64
```

```
dtypes: datetime64[ns](1), float64(1), object(1)  
memory usage: 355.5+ KB
```

```
[36]: sec.dropna().to_csv(sec_path / 'sec_returns.csv', index=False)
```