EDA Notebook - Sec Filings 10-K Dataset.

```
- Start date; mid-september.
- Code notebook author & owner - Joel Markapudi
```

Plan for notebook EDA: (potentially)

1. Schema of dataset ovevrview, Column overview, etc.
2. Company stats, Temporal coverage/stats (yearly filings, etc.), Sentence Distribution stats.
3. Section Mapping & Distribution, Section breakdown, Text by Section stats, Common N-Grams (Top 20) ??
4. Missing data, Missing data patterns, Cardinality analysis. (heatmaps, etc.)
5. Data Quality flags, Edge cases.

High level plan for now.

```
[HF datasets] → [Ingest/Cache] → [Chunk+Metadata] → [Embed] → [Index]
                                        |                  ↑
                                        v                  |
                                  [Query Router] → [Retrieve]
                                        |                  |
                                        v                  v
                             [Structured Extractor]  [Summarizer]
                                        |                  |
                                        └───[Normalizer/Validator]───┘
                                                  |
                                           [Fusion & Cache]
                                                  |
                                           [Report/API]
```

In [17]:
```python
# Notebook bootstrap
from pathlib import Path
import sys, os

# point Python to your ./src so `import` works
PROJECT_ROOT = Path.cwd().resolve()
```

```python
SRC = PROJECT_ROOT / "src"
if str(SRC) not in sys.path:
    sys.path.insert(0, str(SRC))

# env vars (reads assets/config.env you already have)
from dotenv import load_dotenv
load_dotenv(PROJECT_ROOT / "assets" / "config.env")

# autoreload so edits in src/ reflect without restarting kernel
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

In [18]:

```python
import nltk
nltk.download('punkt')       # Sentence tokenizer
nltk.download('stopwords')   # English stopwords
nltk.download('averaged_perceptron_tagger')  # POS tagging (optional)
```

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

Out[18]: True

In [19]:

```python
## Starting all EDA:

# ============================================================================
# Section 1: Data Understanding - SEC 10-K Filings Dataset

import pandas as pd
import polars as pl
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```python
from pathlib import Path
from itables import init_notebook_mode, show
import warnings
from IPython.display import display, HTML


# warnings.filterwarnings('ignore')
# init_notebook_mode(all_interactive=True)

# sns.set_style("whitegrid")
# plt.rcParams['figure.figsize'] = (12, 6)
# plt.rcParams['font.size'] = 11


# # Polars display settings (show ALL columns, no truncation)
# pl.Config.set_tbl_width_chars(1000)
# pl.Config.set_tbl_cols(-1)  # -1 means show ALL columns
# pl.Config.set_tbl_rows(100)  # Show up to 100 rows when printing

# # Pandas display settings (for later use)
# pd.set_option('display.max_columns', None)
# pd.set_option('display.max_colwidth', 100)
# pd.set_option('display.precision', 2)


def display_table_with_html(df, title=""):
    """Display pandas DataFrame as styled HTML table"""
    display(HTML(f"<h3>{title}</h3>"))
    html_str = df.to_html(classes='table table-striped table-hover', border=0)
    display(HTML(html_str))


print("Environment ready")


# Load dataset
DATA_PATH = Path("../data/exports/sec_filings_small_full.parquet")
df = pl.read_parquet(DATA_PATH)

print(f"Dataset loaded: {df.shape[0]:,} rows × {df.shape[1]} columns")
print(f"Memory usage: {df.estimated_size('mb'):.1f} MB")
```

```
Environment ready
Dataset loaded: 200,000 rows × 19 columns
Memory usage: 144.1 MB
```

In [21]:
```
display_table_with_html(
    df.head(3).to_pandas(),
    title="Dataset Preview (First 3 Rows)"
)
```

Dataset Preview (First 3 Rows)

	cik	sentence	section	labels	filingDate	name	docID	sentenceID	sentenceCount	tickers	exch
0	0000001750	ITEM 1.BUSINESS General AAR CORP. and its subsidiaries are referred to herein collectively as “AAR,” “Company,” “we,” “us,” and “our” unless the context indicates otherwise.	0	{'1d': 0, '30d': 0, '5d': 1}	2020-07-21	AAR CORP	0000001750_10-K_2020	0000001750_10-K_2020_section_1_0	1	[AIR]	
1	0000001750	AAR was founded in 1951, organized in 1955 and reincorporated in Delaware in 1966.	0	{'1d': 0, '30d': 0, '5d': 1}	2020-07-21	AAR CORP	0000001750_10-K_2020	0000001750_10-K_2020_section_1_1	2	[AIR]	

	cik	sentence	section	labels	filingDate	name	docID	sentenceID	sentenceCount	tickers	exch
2	0000001750	We are a diversified provider of products and services to the worldwide aviation and government and defense markets.	0	{'1d': 0, '30d': 0, '5d': 1}	2020-07-21	AAR CORP	0000001750_10-K_2020	0000001750_10-K_2020_section_1_2	3	[AIR]	

	cik	sentence	section	labels	filingDate	name	docID	sentenceID	sentenceCount	tickers	excl

First View Analysis On Content:

Three columns contain nested JSON/dict structures:

```
Column          Structure                              Content
labels          {'1d': 0, '30d': 0, '5d': 1}       Binary classification targets (market
movement?)
returns         Nested dict with 1d/5d/30d keys     Stock price data: closePriceStartDate,
closePriceEndDate,
                                                    ret (return %), date
tickers         [AIR]                               List of ticker symbols
```

1. These are ML targets. Cannot directly store these in Qdrant payload (need flattening)
2. Tickers list: Understand this better, soon.

```
Row 0: "ITEM 1.BUSINESS General AAR CORP. and its subsidiaries..." (180 chars)
Row 1: "AAR was founded in 1951, organized in 1955..." (84 chars)
Row 2: "We are a diversified provider of products..." (121 chars)
```

1. Sentences are coherent, complete thoughts.

```
"ITEM 1.BUSINESS General AAR CORP. and its subsidiaries are referred to..."
 └───────────────┘
   This part
```

- The text itself starts with "ITEM 1.BUSINESS" as part of the sentence content. It's not in a separate column - it's embedded in the sentence text.
- Are cik, section, reportDate actually useful for filtering? (Do they have good distribution?)
- Are there other patterns we haven't discovered yet?

In [22]:
```
# Table 1: Complete schema with statistics
schema_info = []

for col in df.columns:
    dtype = str(df[col].dtype)
    null_count = df[col].null_count()
    null_pct = (null_count / len(df)) * 100
    unique_count = df[col].n_unique()

    try:
        if df[col].dtype == pl.List(pl.Utf8):
            sample = str(df[col].drop_nulls().head(1).to_list()[0][:2]) + "..."
        else:
            sample = str(df[col].drop_nulls().head(1).to_list()[0])
            if len(sample) > 50:
                sample = sample[:47] + "..."
    except:
        sample = "N/A"

    schema_info.append({
        'Column': col,
        'Type': dtype,
        'Nulls': f"{null_count:,}",
        'Null %': f"{null_pct:.1f}%",
        'Unique': f"{unique_count:,}",
        'Sample': sample
    })

schema_df = pd.DataFrame(schema_info)

display_table_with_html( schema_df, title="Table 1: Full Schema Overview (19 Columns)" )
```

Table 1: Full Schema Overview (19 Columns)

	Column	Type	Nulls	Null %	Unique	Sample
0	cik	String	0	0.0%	10	0000001750
1	sentence	String	0	0.0%	96,465	ITEM 1.BUSINESS General AAR CORP. and its subsi...
2	section	Int64	0	0.0%	20	0
3	labels	Struct({'1d': Int64, '30d': Int64, '5d': Int64})	0	0.0%	8	{'1d': 0, '30d': 0, '5d': 1}
4	filingDate	String	0	0.0%	181	2020-07-21
5	name	String	0	0.0%	10	AAR CORP
6	docID	String	0	0.0%	188	0000001750_10-K_2020
7	sentenceID	String	0	0.0%	200,000	0000001750_10-K_2020_section_1_0
8	sentenceCount	Int64	0	0.0%	200,000	1
9	tickers	List(String)	0	0.0%	10	['AIR']...
10	exchanges	List(String)	0	0.0%	3	['NYSE']...
11	entityType	String	0	0.0%	1	operating
12	sic	String	0	0.0%	10	3720
13	stateOfIncorporation	String	0	0.0%	5	DE
14	tickerCount	Int32	0	0.0%	1	1
15	acceptanceDateTime	String	0	0.0%	188	2020-07-21T17:19:15.000Z
16	form	String	0	0.0%	1	10-K
17	reportDate	String	0	0.0%	91	2020-05-31
18	returns	Struct({'1d': Struct({'closePriceEndDate': Float64, 'closePriceStartDate': Float64, 'endDate': String, 'ret': Float64, 'startDate': String}), '30d':	0	0.0%	188	{'1d': {'closePriceEndDate': 19.010000228881836...

Column	Type	Nulls	Null %	Unique	Sample
	Struct({'closePriceEndDate': Float64, 'closePriceStartDate': Float64, 'endDate': String, 'ret': Float64, 'startDate': String}), '5d': Struct({'closePriceEndDate': Float64, 'closePriceStartDate': Float64, 'endDate': String, 'ret': Float64, 'startDate': String})})				

In [24]:

```
# Table 2: Logical grouping of columns by purpose
categories = {
    '🔑 Identifiers': ['cik', 'docID', 'sentenceID', 'name', 'tickers'],
    '📝 Text Content': ['sentence'],
    '📊 Document Metadata': ['section', 'filingDate', 'reportDate', 'period', 'form', 'acceptanceDateTime'],
    '🏢 Company Info': ['exchanges', 'entityType', 'sic', 'stateOfIncorporation', 'tickerCount'],
    '🎯 ML Targets': ['labels', 'returns'],
    '📈 Derived/Counters': ['sentenceCount']
}

# Build summary table
cat_summary = []
for category, cols in categories.items():
    for col in cols:
        if col in df.columns:
            dtype = str(df[col].dtype)
            null_pct = (df[col].null_count() / len(df)) * 100
            unique = df[col].n_unique()

            cat_summary.append({
                'Category': category,
                'Column': col,
                'Type': dtype,
                'Null %': f"{null_pct:.1f}%",
                'Unique': f"{unique:,}"
            })

cat_df = pd.DataFrame(cat_summary)

display_table_with_html(
    cat_df,
```

```
    title="Table 2: Column Categorization by Purpose"
)
```

Table 2: Column Categorization by Purpose

	Category	Column	Type	Null %	Unique
0	🔑 Identifiers	cik	String	0.0%	10
1	🔑 Identifiers	docID	String	0.0%	188
2	🔑 Identifiers	sentenceID	String	0.0%	200,000
3	🔑 Identifiers	name	String	0.0%	10
4	🔑 Identifiers	tickers	List(String)	0.0%	10
5	📝 Text Content	sentence	String	0.0%	96,465
6	📊 Document Metadata	section	Int64	0.0%	20
7	📊 Document Metadata	filingDate	String	0.0%	181
8	📊 Document Metadata	reportDate	String	0.0%	91
9	📊 Document Metadata	form	String	0.0%	1
10	📊 Document Metadata	acceptanceDateTime	String	0.0%	188
11	🏢 Company Info	exchanges	List(String)	0.0%	3
12	🏢 Company Info	entityType	String	0.0%	1
13	🏢 Company Info	sic	String	0.0%	10
14	🏢 Company Info	stateOfIncorporation	String	0.0%	5
15	🏢 Company Info	tickerCount	Int32	0.0%	1
16	🎯 ML Targets	labels	Struct({'1d': Int64, '30d': Int64, '5d': Int64})	0.0%	8
17	🎯 ML Targets	returns	Struct({'1d': Struct({'closePriceEndDate': Float64, 'closePriceStartDate': Float64, 'endDate': String, 'ret': Float64, 'startDate': String}), '30d': Struct({'closePriceEndDate': Float64, 'closePriceStartDate': Float64, 'endDate': String, 'ret': Float64, 'startDate': String}), '5d':	0.0%	188

	Category	Column	Type	Null %	Unique
			Struct({'closePriceEndDate': Float64, 'closePriceStartDate': Float64, 'endDate': String, 'ret': Float64, 'startDate': String})})		
18	📈 Derived/Counters	sentenceCount	Int64	0.0%	200,000

- Expected: 10 sections (0-9 for 10-K items)
- Actual: section has 20 unique values
  - What this means:
  - Either sections go beyond 0-9 (10-19 range?)
  - OR there are sub-sections (e.g., 1.1, 1.2 encoded as 11, 12?)

```
Field       Unique Values          Interpretation                   Filtering Value
cik             10                     10 companies only                ✅ HIGH - company
filter
name        10                 1:1 with CIK                 ✅ HIGH - display name
tickers     10                 1:1 with company                 ✅ HIGH - user-friendly
filter
section     20                 ⚠️ MORE than expected        ✅ HIGH - but needs mapping
docID       188                ~19 docs/company                 ✅ MEDIUM - document-level
filter
filingDate      181                    Nearly 1:1 with docID          ⚠️ LOW - use
reportDate instead
reportDate      91                     ~2 filings/period                ✅ MEDIUM - time
filter
sentence        96,465             48% unique                       ❌ N/A - this is the
content
sentenceID      200,000            100% unique                      ✅ HIGH - primary key
```

- 188 documents total across 10 companies:
  - Average: 18.8 documents per company
  - This suggests ~19 years of filings (if annual 10-Ks)
- 91 unique reportDates vs 181 filingDates:

    - Multiple companies share same fiscal year-ends
    - Common fiscal year-ends: Dec 31, Jun 30, Sep 30
    - filingDate spread (181 unique) = different filing times
- Three binary sentiment labels (1d, 5d, 30d) derived from market reaction windows.
- Potential supervised target for finetuning sentiment or volatility predictors.
- Label-conditioned embeddings: correlate language tone with short-term market moves.
- Later: contrastive training of “risk-positive vs risk-negative” sentences.

Metadata Richness:

- can enrich embeddings with these categorical tags (via adapters or metadata vectors).
- these features are extremely useful for metadata-filtered search and bias analysis later.

Text Content Insights

- 96,465 unique sentences out of 200,000 total:
    - Duplication rate: 51.7% (103,535 repeated sentences)
    - This is NORMAL for 10-Ks because:
        - Boilerplate language (risk disclaimers, accounting policies)
        - Repeated across years ("We are incorporated in Delaware...")
        - Standard regulatory phrases

For RAG:

- Duplicates are FINE (same sentence, different context/year)
- Embeddings will cluster similar content

Low-value fields (don't change, not useful for filtering):

```
Field                   Unique      Why Low Value
entityType              1           Always "operating" (no variance)
tickerCount             1           Always 1 ticker (derived field)
form                    1           Always "10-K" (dataset definition)
```

```
exchanges              3          Only 3 exchanges (NYSE, NASDAQ, ?) - low filtering value
stateOfIncorporation   5          Only 5 states (mostly DE) - not relevant for financial
analysis
```

Medium-value field:

```
Field                  Unique     Potential Uses
sic10                  10         Industry codes (1:1 with company, but useful for "show me
tech companies")
```

ML Target Fields (Finance-Specific)

- labels structure: {'1d': 0, '30d': 0, '5d': 1}
- Binary classification: Did stock go UP (1) or DOWN/FLAT (0)?
- Timeframes: 1-day, 5-day, 30-day post-filing
- 8 unique combinations = all possible 0/1 patterns ($2^3 = 8$)
- returns structure: Nested price data
- 188 unique values = 1 per document (document-level targets, not sentence-level)
- Contains: start/end prices, return %, dates
- Used for regression tasks (predict magnitude of movement)

Temporal Metadata Analysis

Three timestamp fields:

```
Field              Unique       Granularity        Use Case
reportDate         91           Fiscal period end  ✅ "Show me Q4 2019 results"
filingDate         181          SEC filing date    ⚠️ "When did market learn this?"
acceptanceDateTime 188          Exact timestamp    ❌ Too granular (hour/minute irrelevant)
```

- Why reportDate (91) < filingDate (181)?
- Multiple companies file on same calendar date (e.g., 2020-02-28)
- But their fiscal year-ends differ (some Dec 31, some Jan 31, some custom)
- ❌ labels, returns (ML targets)
- ❌ entityType, tickerCount, form (no variance)

- ❌ exchanges, stateOfIncorporation (low value)
- ❌ acceptanceDateTime (too granular)
- ❌ sentenceCount (internal counter)

Chunking:

- Chunking window ≈ 3–5 sentences will yield ~100–150 tokens — perfect for encoder models (MiniLM, E5, Titan Embeddings).
- Avoid over-chunking: 10-K prose is repetitive; smaller chunks improve recall for factual retrieval.
- “sentenceID” includes hierarchy: cik_form_year_section_index — perfect for traceability and citation.

Consideration	Insight from EDA	Recommended Approach
Chunk length	Sentences average 28 words; coherent across 2–5-sentence spans.	Use sliding window of 3 sentences (≈ 100–150 tokens).
Chunk boundaries	Section IDs (0–19) define strong topical boundaries.	Chunk within each section; reset window at new section.
Metadata filters	CIK, section, reportDate are perfectly populated.	Use these as metadata filters in vector DB (OpenSearch).
Embedding schema	Text + metadata + docID	Each vector record → `{cik, section, reportDate, sentence_text, embedding}` .
Edge cases	Section imbalance (see dataset card: Item 7 > Item 14 etc.)	Weighted sampling or per-section retrieval balancing.

In [25]:

```
# Answer Q2: Company distribution balance
company_stats = (
    df.group_by("cik", "name")
    .agg([
        pl.count("sentence").alias("sentence_count"),
        pl.col("tickers").first().alias("tickers"),
        pl.col("sic").first().alias("sic"),
        pl.n_unique("docID").alias("num_filings"),
        pl.col("reportDate").min().alias("earliest_report"),
        pl.col("reportDate").max().alias("latest_report"),
    ])
    .sort("sentence_count", descending=True)
)
```

```python
# Convert to pandas for display
company_stats_pd = company_stats.to_pandas()

# Extract first ticker from list (clean display)
company_stats_pd['ticker'] = company_stats_pd['tickers'].apply(lambda x: x[0] if isinstance(x, list) and len(x) > 0 els
company_stats_pd = company_stats_pd.drop(columns=['tickers'])

# Reorder columns for clarity
company_stats_pd = company_stats_pd[['cik', 'name', 'ticker', 'sic', 'sentence_count', 'num_filings', 'earliest_report'

display_table_with_html(
    company_stats_pd,
    title="Table 3: Company Statistics (Answers Q2: Distribution Balance)"
)

# Quick summary
total_sentences = company_stats_pd['sentence_count'].sum()
avg_sentences = company_stats_pd['sentence_count'].mean()
std_sentences = company_stats_pd['sentence_count'].std()

print(f"\n📊 Distribution Summary:")
print(f"   Total sentences: {total_sentences:,}")
print(f"   Average per company: {avg_sentences:,.0f}")
print(f"   Std deviation: {std_sentences:,.0f}")
print(f"   Imbalance ratio: {company_stats_pd['sentence_count'].max() / company_stats_pd['sentence_count'].min():.2f}x
```

Table 3: Company Statistics (Answers Q2: Distribution Balance)

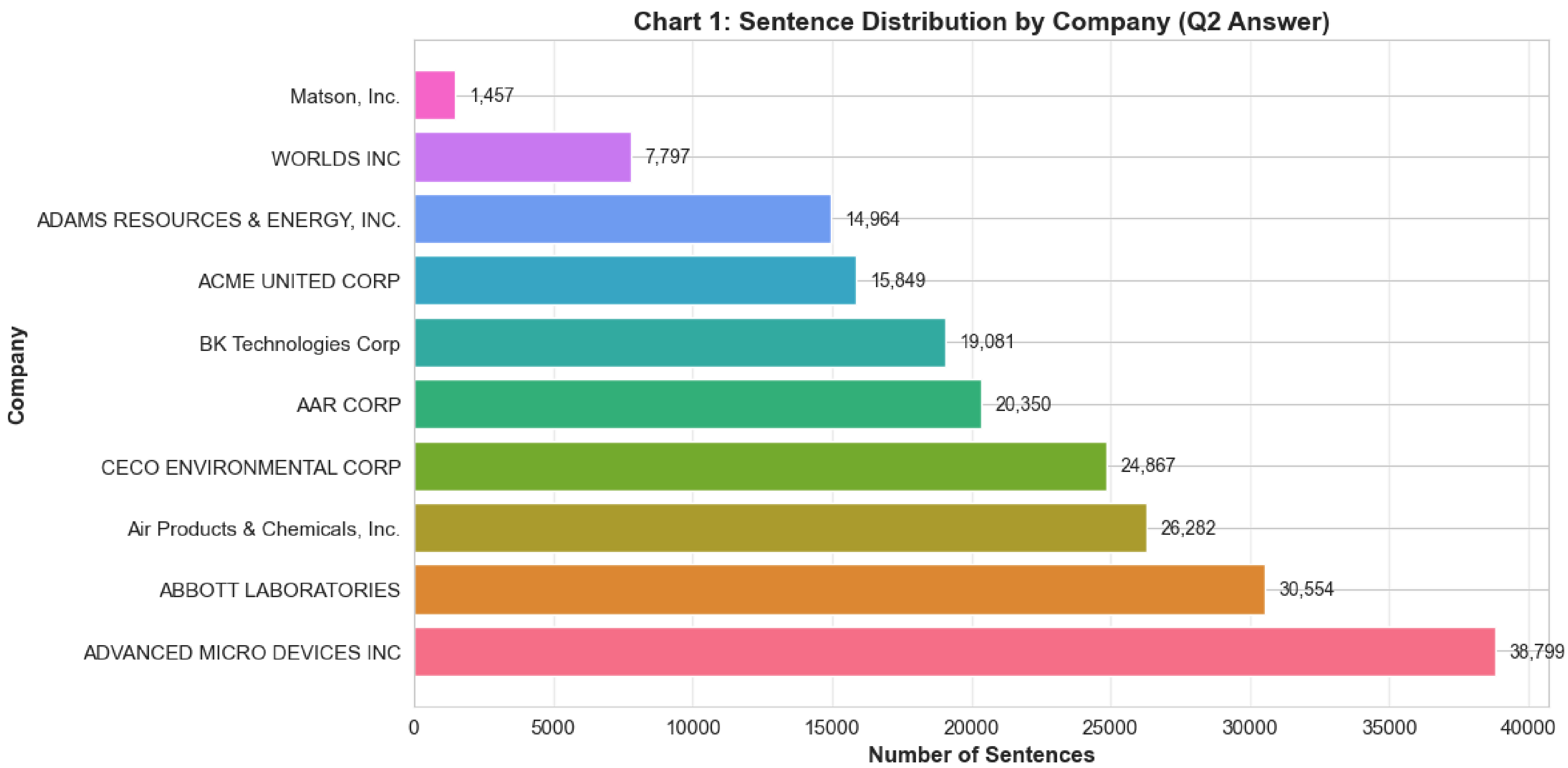
	cik	name	ticker	sic	sentence_count	num_filings	earliest_report	latest_report
0	0000002488	ADVANCED MICRO DEVICES INC	N/A	3674	38799	24	1993-12-26	2020-12-26
1	0000001800	ABBOTT LABORATORIES	N/A	2834	30554	25	1993-12-31	2020-12-31
2	0000002969	Air Products & Chemicals, Inc.	N/A	2810	26282	20	2001-09-30	2020-09-30
3	0000003197	CECO ENVIRONMENTAL CORP	N/A	3564	24867	17	2004-12-31	2020-12-31
4	0000001750	AAR CORP	N/A	3720	20350	21	1994-05-31	2020-05-31
5	0000002186	BK Technologies Corp	N/A	3663	19081	21	1995-12-31	2020-12-31
6	0000002098	ACME UNITED CORP	N/A	3420	15849	26	1995-12-31	2020-12-31
7	0000002178	ADAMS RESOURCES & ENERGY, INC.	N/A	5172	14964	19	2002-12-31	2020-12-31
8	0000001961	WORLDS INC	N/A	7372	7797	13	2008-12-31	2020-12-31
9	0000003453	Matson, Inc.	N/A	4400	1457	2	2019-12-31	2020-12-31

```
📊 Distribution Summary:
   Total sentences: 200,000
   Average per company: 20,000
   Std deviation: 10,874
   Imbalance ratio: 26.63x (max/min)
```

In [26]:

```
# Visualize company balance (bar chart)
fig, ax = plt.subplots(figsize=(12, 6))

companies = company_stats_pd['name'].tolist()
counts = company_stats_pd['sentence_count'].tolist()

bars = ax.barh(companies, counts, color=sns.color_palette("husl", 10))
ax.set_xlabel('Number of Sentences', fontsize=12, fontweight='bold')
ax.set_ylabel('Company', fontsize=12, fontweight='bold')
ax.set_title('Chart 1: Sentence Distribution by Company (Q2 Answer)', fontsize=14, fontweight='bold')
ax.grid(axis='x', alpha=0.3)

# Add value labels on bars
for i, (bar, count) in enumerate(zip(bars, counts)):
    ax.text(count + 500, bar.get_y() + bar.get_height()/2,
            f'{count:,}', va='center', fontsize=10)
```

```
plt.tight_layout()
plt.show()
```



In [27]:
```
# Answer Q3: Temporal distribution
# Extract year from reportDate
df_temporal = df.with_columns([
    pl.col("reportDate").str.strptime(pl.Date, "%Y-%m-%d").alias("report_date_parsed")
]).with_columns([
    pl.col("report_date_parsed").dt.year().alias("year")
])

# Sentences per year
year_stats = (
    df_temporal.group_by("year")
    .agg([
```

```python
        pl.count("sentence").alias("sentence_count"),
        pl.n_unique("docID").alias("num_filings")
    ])
    .sort("year")
)

year_stats_pd = year_stats.to_pandas()

display_table_with_html(
    year_stats_pd,
    title="Table 4: Temporal Distribution by Year (Answers Q3: Date Range)"
)

# Summary stats
print(f"\n📅 Temporal Coverage:")
print(f"   Earliest year: {year_stats_pd['year'].min()}")
print(f"   Latest year: {year_stats_pd['year'].max()}")
print(f"   Total span: {year_stats_pd['year'].max() - year_stats_pd['year'].min() + 1} years")
print(f"   Average sentences/year: {year_stats_pd['sentence_count'].mean():,.0f}")
```

Table 4: Temporal Distribution by Year (Answers Q3: Date Range)

	year	sentence_count	num_filings
0	1993	853	2
1	1994	1257	3
2	1995	1135	3
3	1996	1211	3
4	1997	1510	3
5	1998	1365	3
6	1999	370	1
7	2000	1181	2
8	2001	1298	3
9	2002	6361	7
10	2003	6631	7
11	2004	7878	8
12	2005	8290	8
13	2006	9637	8
14	2007	9916	8
15	2008	9929	9
16	2009	10473	9
17	2010	10511	9
18	2011	10879	9
19	2012	9976	9
20	2013	9713	9
21	2014	10228	9

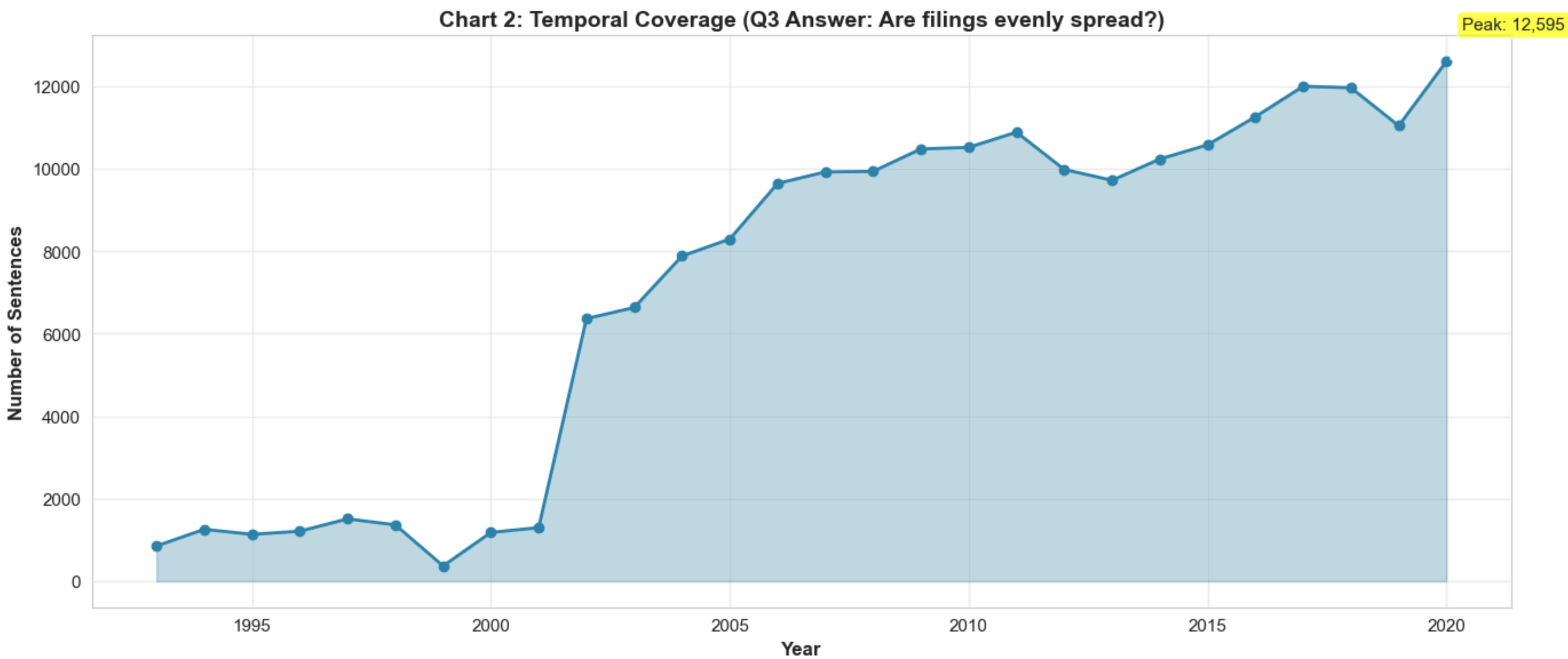
	year	sentence_count	num_filings
22	2015	10572	9
23	2016	11251	9
24	2017	11990	9
25	2018	11956	9
26	2019	11034	10
27	2020	12595	10

```
📅 Temporal Coverage:
   Earliest year: 1993
   Latest year: 2020
   Total span: 28 years
   Average sentences/year: 7,143
```

In [28]:

```
# Chart 2: Sentences over time
fig, ax = plt.subplots(figsize=(14, 6))

years = year_stats_pd['year'].tolist()
counts = year_stats_pd['sentence_count'].tolist()

ax.plot(years, counts, marker='o', linewidth=2, markersize=6, color='#2E86AB')
ax.fill_between(years, counts, alpha=0.3, color='#2E86AB')

ax.set_xlabel('Year', fontsize=12, fontweight='bold')
ax.set_ylabel('Number of Sentences', fontsize=12, fontweight='bold')
ax.set_title('Chart 2: Temporal Coverage (Q3 Answer: Are filings evenly spread?)', fontsize=14, fontweight='bold')
ax.grid(True, alpha=0.3)

# Add annotations for min/max years
max_idx = counts.index(max(counts))
min_idx = counts.index(min(counts))
ax.annotate(f'Peak: {max(counts):,}', xy=(years[max_idx], counts[max_idx]),
            xytext=(10, 20), textcoords='offset points',
            bbox=dict(boxstyle='round', fc='yellow', alpha=0.7),
            arrowprops=dict(arrowstyle='->', connectionstyle='arc3,rad=0'))
```

```python
plt.tight_layout()
plt.show()
```



In [29]:

```python
# Answer Q1: What are the 20 section codes?
section_stats = (
    df.group_by("section")
    .agg([
        pl.count("sentence").alias("sentence_count"),
        pl.n_unique("cik").alias("num_companies")
    ])
    .sort("section")
)

section_stats_pd = section_stats.to_pandas()

# Add percentage
section_stats_pd['percentage'] = (section_stats_pd['sentence_count'] / len(df) * 100).round(2)

display_table_with_html(
    section_stats_pd,
```

```
    title="Table 5: Section Code Distribution (Answers Q1: What are the 20 codes?)"
)

print(f"\n🔍 Section Analysis:")
print(f"   Expected sections: 0-9 (10 codes)")
print(f"   Actual sections: {section_stats_pd['section'].min()} to {section_stats_pd['section'].max()} ({len(section_st
print(f"   Section coverage: {section_stats_pd['num_companies'].min()}-{section_stats_pd['num_companies'].max()} compan
```

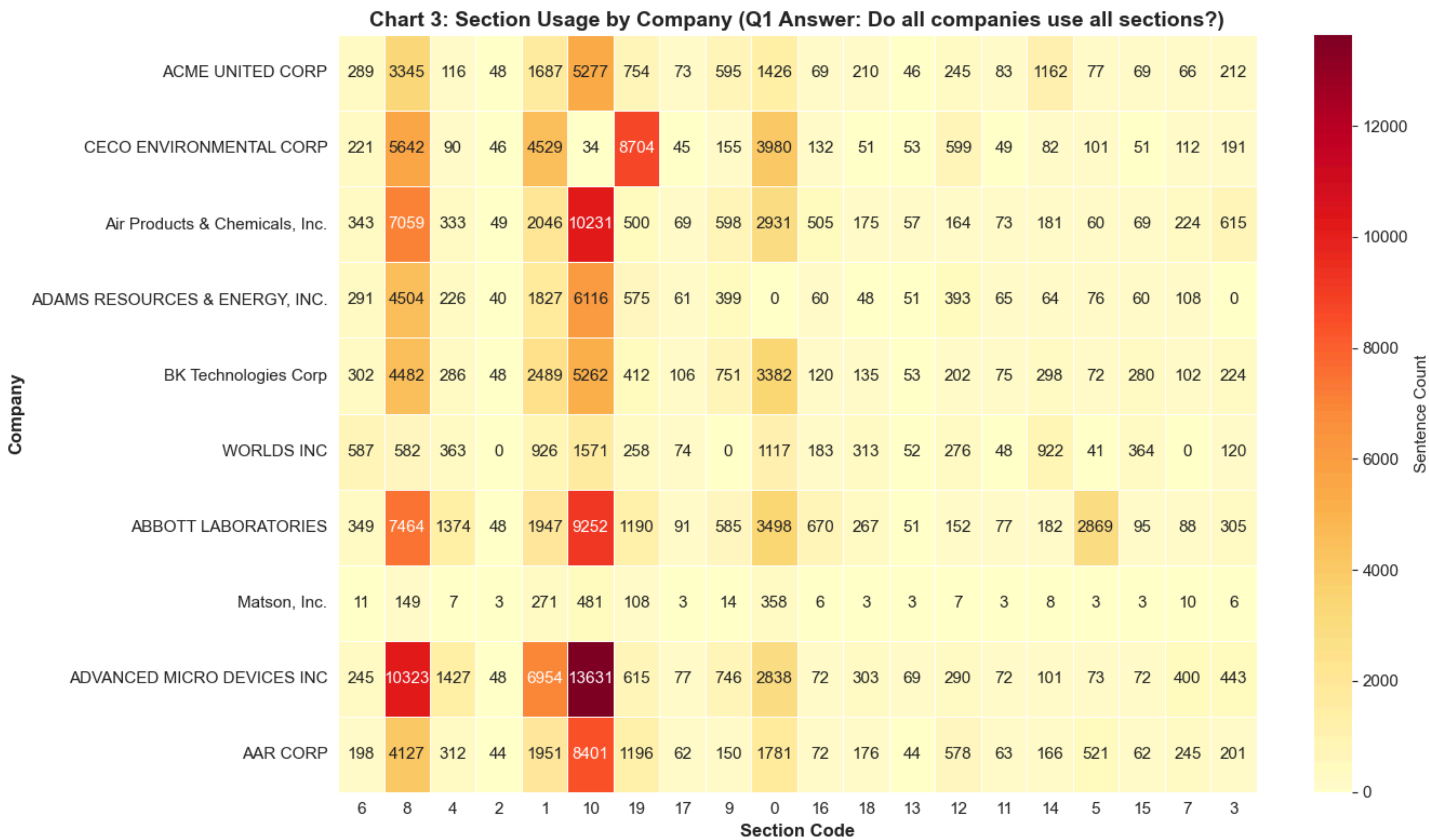
Table 5: Section Code Distribution (Answers Q1: What are the 20 codes?)

	section	sentence_count	num_companies	percentage
0	0	21311	9	10.66
1	1	24627	10	12.31
2	2	374	9	0.19
3	3	2317	9	1.16
4	4	4534	10	2.27
5	5	3893	10	1.95
6	6	2836	10	1.42
7	7	1355	9	0.68
8	8	47677	10	23.84
9	9	3993	9	2.00
10	10	60256	10	30.13
11	11	608	10	0.30
12	12	2906	10	1.45
13	13	479	10	0.24
14	14	3166	10	1.58
15	15	1125	10	0.56
16	16	1889	10	0.94
17	17	661	10	0.33
18	18	1681	10	0.84
19	19	14312	10	7.16

```
🔍 Section Analysis:
   Expected sections: 0-9 (10 codes)
   Actual sections: 0 to 19 (20 codes)
   Section coverage: 9-10 companies per section
```

In [30]:

```
# Cross-tab: Which companies use which sections?
section_company_cross = (
    df.group_by(["section", "name"])
    .agg(pl.count("sentence").alias("count"))
    .pivot(index="name", columns="section", values="count")
    .fill_null(0)
)

section_company_pd = section_company_cross.to_pandas().set_index('name')

# Heatmap
fig, ax = plt.subplots(figsize=(14, 8))
sns.heatmap(section_company_pd, annot=True, fmt='.0f', cmap='YlOrRd',
            linewidths=0.5, cbar_kws={'label': 'Sentence Count'}, ax=ax)
ax.set_title('Chart 3: Section Usage by Company (Q1 Answer: Do all companies use all sections?)',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Section Code', fontsize=12, fontweight='bold')
ax.set_ylabel('Company', fontsize=12, fontweight='bold')
plt.tight_layout()
plt.show()
```



In [ ]:

```

```

In [ ]:

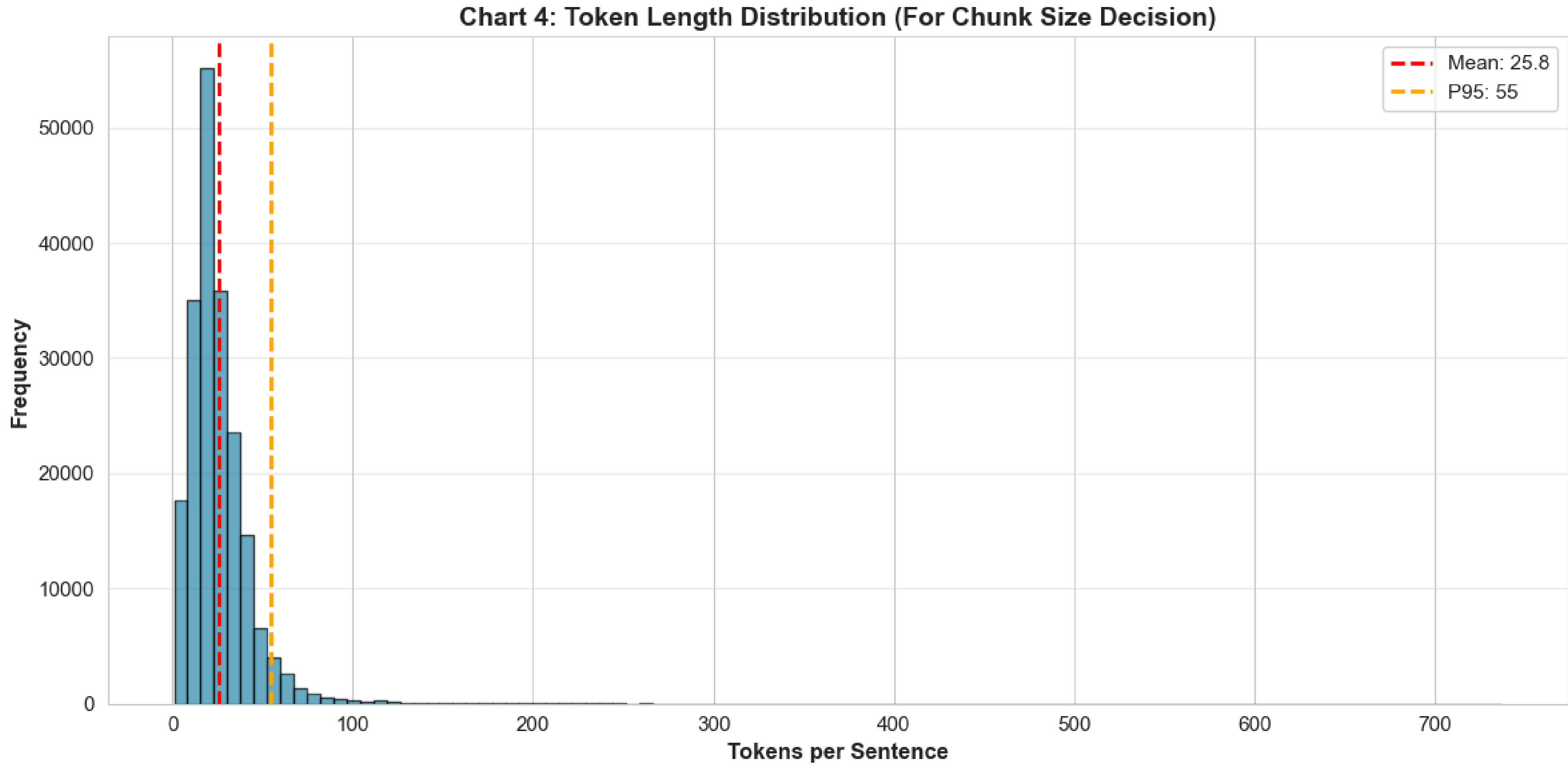
```

```

In [31]:

```
# Token length analysis (word-based tokenization)
import nltk
nltk.download('punkt', quiet=True)

# Calculate token counts
token_counts = df.with_columns([
    pl.col("sentence").str.split(" ").list.len().alias("token_count")
```

```
])

# Statistics
token_stats = token_counts.select([
    pl.col("token_count").mean().alias("mean"),
    pl.col("token_count").median().alias("median"),
    pl.col("token_count").quantile(0.95).alias("p95"),
    pl.col("token_count").max().alias("max"),
    pl.col("token_count").min().alias("min")
]).to_pandas()

print("📊 Token Length Statistics:")
print(f"   Mean: {token_stats['mean'][0]:.1f} tokens")
print(f"   Median: {token_stats['median'][0]:.0f} tokens")
print(f"   95th percentile: {token_stats['p95'][0]:.0f} tokens")
print(f"   Max: {token_stats['max'][0]:.0f} tokens")

# Histogram
fig, ax = plt.subplots(figsize=(12, 6))
token_data = token_counts.select("token_count").to_pandas()['token_count']

ax.hist(token_data, bins=100, color='#2E86AB', alpha=0.7, edgecolor='black')
ax.axvline(token_stats['mean'][0], color='red', linestyle='--', linewidth=2, label=f"Mean: {token_stats['mean'][0]:.1f}
ax.axvline(token_stats['p95'][0], color='orange', linestyle='--', linewidth=2, label=f"P95: {token_stats['p95'][0]:.0f}
ax.set_xlabel('Tokens per Sentence', fontsize=12, fontweight='bold')
ax.set_ylabel('Frequency', fontsize=12, fontweight='bold')
ax.set_title('Chart 4: Token Length Distribution (For Chunk Size Decision)', fontsize=14, fontweight='bold')
ax.legend()
ax.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()

# Chunk size recommendation
print(f"\n💡 Chunk Size Recommendation:")
print(f"   3-sentence chunks: ~{token_stats['mean'][0] * 3:.0f} tokens (avg)")
print(f"   5-sentence chunks: ~{token_stats['mean'][0] * 5:.0f} tokens (avg)")
print(f"   Embedding limit (512 tokens): Fits ~{512 / token_stats['mean'][0]:.1f} sentences")
```

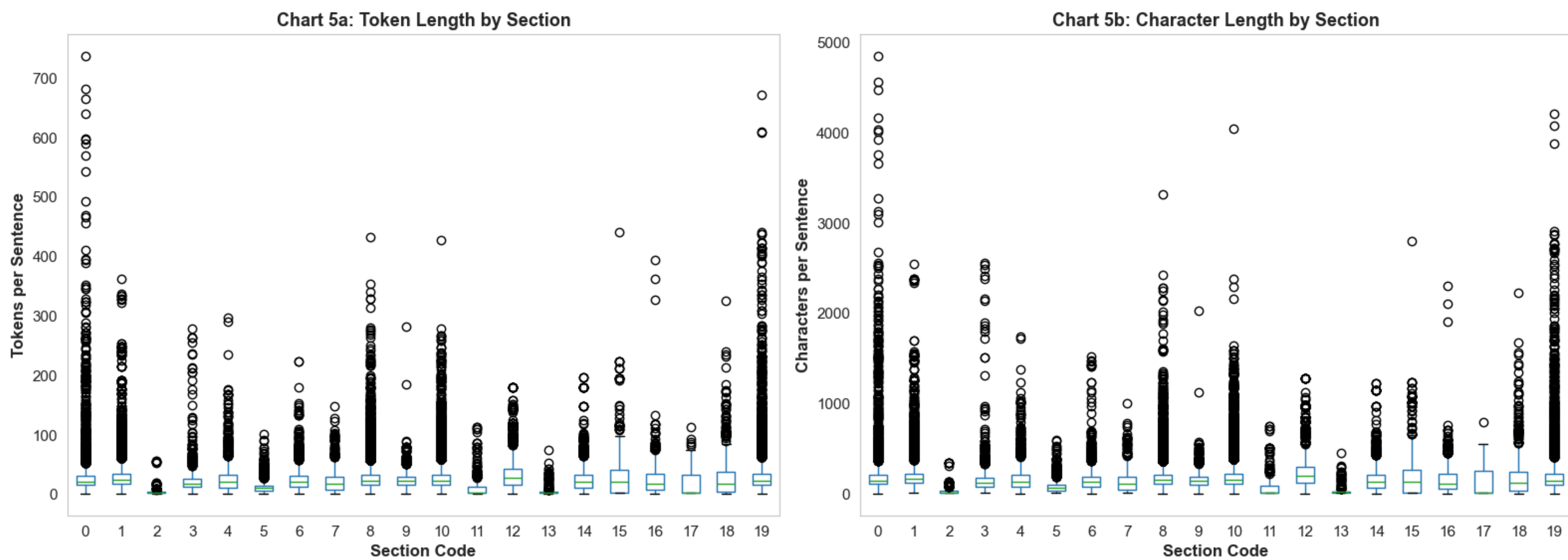
```
📊 Token Length Statistics:
   Mean: 25.8 tokens
   Median: 22 tokens
   95th percentile: 55 tokens
   Max: 737 tokens
```




```
💡 Chunk Size Recommendation:
   3-sentence chunks: ~77 tokens (avg)
   5-sentence chunks: ~129 tokens (avg)
   Embedding limit (512 tokens): Fits ~19.9 sentences
```

In [32]:
```
# Text length by section
section_text_stats = df.with_columns([
    pl.col("sentence").str.len_chars().alias("char_count"),
    pl.col("sentence").str.split(" ").list.len().alias("token_count")
]).select(["section", "char_count", "token_count"])

section_text_pd = section_text_stats.to_pandas()
```

```python
# Boxplot for token count by section
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Token count boxplot
section_text_pd.boxplot(column='token_count', by='section', ax=axes[0], grid=False)
axes[0].set_xlabel('Section Code', fontsize=12, fontweight='bold')
axes[0].set_ylabel('Tokens per Sentence', fontsize=12, fontweight='bold')
axes[0].set_title('Chart 5a: Token Length by Section', fontsize=13, fontweight='bold')
axes[0].get_figure().suptitle('')  # Remove default title

# Char count boxplot
section_text_pd.boxplot(column='char_count', by='section', ax=axes[1], grid=False)
axes[1].set_xlabel('Section Code', fontsize=12, fontweight='bold')
axes[1].set_ylabel('Characters per Sentence', fontsize=12, fontweight='bold')
axes[1].set_title('Chart 5b: Character Length by Section', fontsize=13, fontweight='bold')
axes[1].get_figure().suptitle('')

plt.tight_layout()
plt.show()
```



In [33]:
```python
# Detailed stats by section
section_density = (
```

```
    df.with_columns([
        pl.col("sentence").str.len_chars().alias("char_count"),
        pl.col("sentence").str.split(" ").list.len().alias("token_count")
    ])
    .group_by("section")
    .agg([
        pl.col("char_count").mean().alias("avg_chars"),
        pl.col("token_count").mean().alias("avg_tokens"),
        pl.col("char_count").median().alias("median_chars"),
        pl.col("token_count").median().alias("median_tokens"),
        pl.col("char_count").max().alias("max_chars"),
        pl.col("token_count").max().alias("max_tokens"),
        pl.count("sentence").alias("sentence_count")
    ])
    .sort("section")
)

section_density_pd = section_density.to_pandas()
section_density_pd = section_density_pd.round(1)

display_table_with_html(
    section_density_pd,
    title="Table 6: Text Density by Section (Item 7 vs Item 1A Comparison)"
)

# Highlight extremes
densest_section = section_density_pd.loc[section_density_pd['avg_tokens'].idxmax(), 'section']
sparsest_section = section_density_pd.loc[section_density_pd['avg_tokens'].idxmin(), 'section']

print(f"\n🔍 Density Analysis:")
print(f"   Densest section: {densest_section} ({section_density_pd.loc[section_density_pd['avg_tokens'].idxmax(), 'avg_
print(f"   Sparsest section: {sparsest_section} ({section_density_pd.loc[section_density_pd['avg_tokens'].idxmin(), 'av
```

Table 6: Text Density by Section (Item 7 vs Item 1A Comparison)

	section	avg_chars	avg_tokens	median_chars	median_tokens	max_chars	max_tokens	sentence_count
0	0	174.2	25.4	145.0	21.0	4850	737	21311
1	1	182.5	27.7	160.0	24.0	2544	362	24627
2	2	27.6	4.5	8.0	2.0	342	55	374
3	3	140.4	20.7	116.0	18.0	2557	279	2317
4	4	152.0	23.6	131.0	21.0	1743	296	4534
5	5	74.6	11.6	68.0	10.0	596	101	3893
6	6	149.2	23.5	127.0	20.0	1515	223	2836
7	7	129.7	20.5	112.0	18.0	1002	147	1355
8	8	168.8	26.0	148.0	23.0	3317	433	47677
9	9	146.2	22.7	140.0	22.0	2031	281	3993
10	10	172.1	26.2	150.0	23.0	4045	428	60256
11	11	67.2	10.3	8.0	2.0	751	113	608
12	12	229.8	33.4	199.0	28.0	1273	179	2906
13	13	27.0	4.7	17.0	3.0	451	74	479
14	14	162.6	25.1	127.0	20.0	1223	197	3166
15	15	179.3	27.6	129.0	21.0	2800	440	1125
16	16	158.9	24.5	113.0	17.0	2298	394	1889
17	17	117.4	17.4	8.0	2.0	795	112	661
18	18	165.8	25.7	114.0	18.0	2222	325	1681
19	19	183.7	28.4	140.0	22.0	4214	672	14312

```
🔍 Density Analysis:
   Densest section: 12 (33.4 avg tokens)
   Sparsest section: 2 (4.5 avg tokens)
```

In [34]:
```python
# Identify extreme outliers (likely tables)
outliers = df.with_columns([
    pl.col("sentence").str.len_chars().alias("char_count")
]).filter(
    pl.col("char_count") > 1000  # Threshold for "table-like" text
).select(["section", "char_count", "sentence"]).head(5)

outliers_pd = outliers.to_pandas()

print("⚠️ Outlier Examples (Likely Tables/Lists):")
for idx, row in outliers_pd.iterrows():
    print(f"\n   Section {row['section']} | {row['char_count']} chars")
    print(f"   Preview: {row['sentence'][:150]}...")
```

```
⚠️ Outlier Examples (Likely Tables/Lists):

   Section 1 | 1174 chars
   Preview: Consequently, we are subject to a variety of risks that are specific to international operations, including
the following: •military conflicts, civil ...

   Section 10 | 1022 chars
   Preview: Sales by segment for these customers are as follows: AAR CORP. AND SUBSIDIARIES NOTES TO CONSOLIDATED FINANC
IAL STATEMENTS (Continued) (Dollars in mil...

   Section 12 | 1040 chars
   Preview: The Company’s internal control over financial reporting is a process designed by, or under the supervision o
f, our Chief Executive Officer and Chief F...

   Section 19 | 1330 chars
   Preview: 4.3 Description of Capital Stock (filed herewith) 4.4 Rights Agreement, dated as of March 30, 2020, by and b
etween AAR CORP. and Computershare Trust C...

   Section 19 | 1850 chars
   Preview: Material Contracts 10.1* Amended and Restated AAR CORP. Stock Benefit Plan effective October 1, 2001 (incorp
orated by reference to Exhibit 10.1 to the...
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: