OPEN GENDER JOURNAL

Supplementary File

Lischinsky, Alon (2025): “Cheating, No Matter How You Try to Dress It up”. Newspaper Coverage of Polyamory, 1994–2023. In: Open Gender Journal 9. doi: https://doi.org/10.17169/ogj.2025.282

Alon Lischinsky

Supplementary File

Factiva

The terms and conditions of Factiva do not allow automated search and retrieval. The initial phase of data collection was therefore conducted manually by the author using the following parameters:

```
Search query     la=en AND rst=tnwp AND (polyamory OR
polyamorous OR polyamorist OR poly-amory OR poly-amorous OR
poly-amorist)
Date        All Dates
Source           All Sources
Author           All Authors
Company          All Companies
Subject          All Subjects
Industry         All Industries
Region           All Regions
Language         English
Duplicates off
```

The search yielded 9228 results as of 1 January 2024.
Using the Factiva web interface, results were selected in batches of 100 and exported using the Format for Saving... → Article format option.

BeautifulSoup parsing

The articles were converted to machine-readable format using the Python BeautifulSoup library (Richardson 2023).
The following code implements three functions to parse, extract and save the unique ID, headline, author bylines, publication date (and time, if available), publication title and code, section, word count and full text of each article.

```
def parse_factiva_corpus(corpus_files, as_dataframe=False):
     """Parses one or more HTML files containing articles
saved from
```

```
      Factiva, and returns their content and metadata as a
list or dataframe

      Keyword arguments:
      corpus_files:     HTML file(s) to parse, as a string or
list
      as_dataframe:    return parsed data as a Pandas
dataframe (default: False)
      """
      parsed_corpus = []
      if type(corpus_files)==list:
      for corpus_file in corpus_files:
      parsed_corpus=parsed_corpus+parse_factiva_page(corpus_
file)
      elif type(corpus_files)==str:
      parsed_corpus=parse_factiva_page(corpus_files)
      else:
      print('Error: the argument must be a single file or a
list of them')
      return None
      if as_dataframe:
      import pandas as pd
      df = pd.DataFrame(parsed_corpus, columns = ['id',
'headline', 'authors', 'date', 'publication_id',
'publication', 'section', 'extra_columns', 'wordcount',
'content'])
      df['date']=pd.to_datetime(df['date'])
      return df
      else:
      return parsed_corpus

def parse_factiva_page(data_file: str) -> list:
      """Parses an HTML file saved from Factiva, separating
each article
      it contains and returning a list with their content and
metadata

      Keyword arguments:
      data_file:  HTML file to parse
```

```
      """
      from bs4 import BeautifulSoup
      import logging
      try:
      with open(data_file, 'r') as page:
            logging.info('Reading %s…', data_file)
            soup = BeautifulSoup(page, 'html.parser')
            # each article is a div with a specific class
            factiva_articles = soup.find_all('div',
class_='enArticle')
            logging.info('Found %d articles in file',
len(factiva_articles))
            parsed_articles=[]
            for article in factiva_articles:
                  parsed_articles.append(extract_factiva_
article(article))
            return parsed_articles
      except IOError as err:
      logging.error('Error reading the file %s: %s', data_file,
err)
      raise

def extract_factiva_article(soup_article) -> list:
      """Takes a BeautifulSoup object containing a div
representing an
      article in an HTML file saved from Factiva and parses it
to
      return their content and metadata

      Keyword arguments:
      soup_article:    div object to parse
      """
      from bs4 import BeautifulSoup
      import logging
      # obtain the unique id of the article, which is an
attribute of the parent
      # div, and trim boilerplate
      article_id = soup_article.parent.attrs['id'][8:]
      # all articles contain a headline with a distinctive id
```

```
('hd')
      article_headline = soup_article.find('div', id='hd')
      article_headline_clean = article_headline.text.strip()
      logging.info('Extracting article %s: %s', article_id,
article_headline_clean)
      # all articles contain full text as a series of
paragraphs with a
      # distinctive class ('enarticleParagraph')
      article_text_list = soup_article.find_all('p',
class_='enarticleParagraph')
      logging.debug('Article contains %d paragraphs',
len(article_text_list))
      # other elements we have to parse by guesswork because
they are not
      # properly identified in the code
      # 1. if there is a section identifier, it appears as a
div just before the
      # article headline
      if article_headline.find_previous_siblings('div'):
      article_section = article_headline.find_previous_
siblings('div')[0].text
      logging.debug('Article section is given as %s', article_
section)
      else:
      article_section = ,NO SECTION GIVEN‘
      logging.debug('Article section not given')
      # the remaining metadata as given as divs following the
headline
      article_metadata = article_headline.find_next_
siblings(,div‘)
      # 2. the last of these divs is always the copyright
statement
      article_copyright = article_metadata.pop().text.strip()
      # 3. the penultimate div is always the language code
      article_language = article_metadata.pop().text.strip()
      # 4. there may be one or more authors immediately
following the headline,
      # but they are not consistently marked with class
'author', so they can
      # only be identified by the fact that they are followed
```

```
by a wordcount
      # with a predictable format
      author_counter=0
      for item in article_metadata:
      if not , words‘ in item.text:
            author_counter=author_counter+1
      else:
            break
      if author_counter > 0:
      article_author=[author.text.strip() for author in
article_metadata[:author_counter]]
      logging.debug('Article contains %d author lines',
len(article_author))
      article_metadata=article_metadata[author_counter:]
      else:
      article_author=['NO AUTHOR GIVEN']
      logging.debug('Article author not given')
      # 5. we now know the first item is the wordcount, from
which we can
      # extract the numeric value
      article_wordcount=int(article_metadata.pop(0).text[:-6])
      logging.debug('Article is %d words long', article_
wordcount)
      # 6. now the first item is the date, but there may be a
time as well, so
      # we store it temporarily as text
      article_date=article_metadata.pop(0).text.strip()
      # 7. check for time and if it is listed add it to the
date field
      if re.match(r"\d\d:\d\d (A|P)M", article_metadata[0].
text):
      article_date=datetime.datetime.strptime('
'.join([article_date,
            article_metadata.pop(0).text.strip()]), '%d %B %Y
%I:%M %p').strftime('%Y-%m-%d %H:%M')
      else:
      article_date=datetime.datetime.strptime(article_date,
'%d %B %Y').strftime('%Y-%m-%d')
```

```
      logging.debug('Article published on %s', article_date)
      # 8. next should be the full name of the source
publication
      article_source = article_metadata.pop(0).text.strip()
      # 9. and then the code of the source publication, though
(very rarely)
      # this is missing
      if len(article_metadata)>0:
      article_source_code = article_metadata.pop(0).text.
strip()
      logging.debug('Article published in %s (%s)', article_
source, article_source_code)
      # 10. any remaining metadata describe publication
subdivisions with no
      # oveararching scheme that I can see
      logging.debug('%d unprocessed elements remaining',
len(article_metadata))
      else:
      article_source_code='NO PUBLICATION CODE GIVEN'
      logging.debug('Publication details incomplete: only %s
given', article_source)
      return [ article_id,
            article_headline_clean,
            article_author,
            article_date,
            article_source_code,
            article_source,
            article_section,
            [metadata.text.strip() for metadata in article_
metadata],
            article_wordcount,
            [paragraph.text.strip() for paragraph in article_
text_list]
            ]
```

The parsed data were compiled into the `article_metadata.csv` file included in the dataset on Zenodo:

```
df=parse_factiva_corpus(DATA_FILES, as_dataframe=True)
```

The columns containing the original publication details were discarded from the distributed data because of inconsistencies.

```
df['publication_id']=[re.sub(r'^([A-Za-z]+).*', r'\1',
str(x)).upper() for x in df['id']]
```

A list of normalised publication codes, titles and additional data (medium and country of publication, ISSN, frequency) was manually prepared instead; it is provided as `publication_metadata.csv`.

To integrate these data with the article database, the following code can be used:

```
df_publications=pd.read_csv('publication_metadata.csv')
df=df.merge(df_publications,on='publication_id')
```

The column containing author data was manually preprocessed to remove extraneous text (such as the name of the section or publication, or the word 'by', and to correct parsing errors. The cleaned-up author list is included as column authors_clean.

The column containing the original full text is not included in the version in the dataset on Zenodo because of copyright restrictions that prevent its distribution.

```
df.to_csv('article_metadata.csv', header=True, index=False,
encoding='utf-8', lineterminator='\n', date_format='%Y-%m-
%d %H:%M', columns=['id', 'headline', 'authors', 'authors_
clean', 'date', 'publication_id', 'section', 'extra_
columns'])
```

Keyword computation

The content of the articles was parsed using the NLTK 3.7 implementation of the Penn Treebank tokeniser (Bird/Klein/Loper 2009):

from nltk.tokenize import word_tokenize

```
df['tokenized_content']=[[[word_tokenize(sentence) for
sentence in sent_tokenize(paragraph)] for paragraph in
fulltext] for fulltext in df.content]
df['tokenized_headline']=[[[word_tokenize(sentence) for
sentence in sent_tokenize(headline)]] for headline in
df.headline]
df['tokenized_fulltext']=df['tokenized_
```

```
headline']+df['tokenized_content']
df['tokenized_fulltext_flat']=[[word for paragraph in fulltext
for sentence in paragraph for word in sentence] for fulltext
in df.tokenized_fulltext]
```

Some preprocessing of the parsed content was undertaken in order to match the format of the reference corpus file as closely as possible:

```
# trim any terminal punctuation
def strip_nonalpha(word):
      while len(word) > 0 and not word[0].isalpha():
            word = word[1:]
      while len(word) > 0 and not word[-1].isalpha():
            word = word[:-1]
      return word

df['tokenized_fulltext_clean']=[[strip_nonalpha(word) for
word in fulltext if strip_nonalpha(word)!=''] for fulltext in
df.tokenized_fulltext_flat]

# discard terms that contain periods, digits or equals signs
df['tokenized_fulltext_clean']=[[word for word in fulltext
if not any([(character.isdigit() or character=="=" or
character==".") for character in word])] for fulltext in
df.tokenized_fulltext_clean]

# split terms on slashes and em dashes
df['tokenized_fulltext_clean']=[[part for word in fulltext
for part in word.split('/')] for fulltext in df.tokenized_
fulltext_clean]
df['tokenized_fulltext_clean']=[[part for word in fulltext
for part in word.split('—')] for fulltext in df.tokenized_
fulltext_clean]

# remove soft hyphens
df['tokenized_fulltext_clean']=[[word.replace('\u00ad','')
for word in fulltext] for fulltext in df.tokenized_fulltext_
clean]
```

After preprocessing, per-document frequencies were computed for comparison

against the reference corpus:

```
df['frequencies']=[Counter([word.lower() for word in
fulltext]) for fulltext in df.tokenized_fulltext_clean]
df['wordcount_actual']=[x.total() for x in df.frequencies]
# store corpus frequencies in a separate Counter variable for
efficiency
corpus_frequencies=Counter(word.lower() for fulltext in
df.tokenized_fulltext_clean for word in fulltext)

# load reference corpus frequencies
import csv
with open('g_wordlist.csv') as f:
    reference_frequencies = dict(filter(None, csv.reader(f)))
```

The following code implements functions to compute the statistical measures used for keyness extraction, and to generate a keyword list from dictionaries for a target and reference corpus:

```
# library needed for effect measure functions
import math

def log_ratio(freq_target_corpus: int,target_corpus_size:
int,freq_reference_corpus: int,reference_corpus_size:
int,laplace = 0.1) -> float:
      """Returns the binary log of the ratio of relative
frequencies as a descriptive measure of lexical keyness

      Keyword arguments:
      freq_target_corpus:    frequency of the term in the
target corpus
      target_corpus_size:    total word count of the target
corpus
      freq_reference_corpus: frequency of the term in the
reference corpus
      reference_corpus_size: total word count of the reference
corpus
      laplace:               Laplace correction for terms
absent from the reference corpus (default: 0.1)
      """
      # if there are zero instances of term in reference
```

```
corpus, replace count by Laplace smoothing parameter
      if freq_reference_corpus == 0: freq_reference_corpus =
laplace
      rel_freq_target_corpus = freq_target_corpus / target_
corpus_size # normalise
      rel_freq_reference_corpus = freq_reference_corpus /
reference_corpus_size # normalise
      # ratio of relative frequencies is rel_freq_target_
corpus / rel_freq_reference_corpus
      measure = math.log2(rel_freq_target_corpus / rel_freq_
reference_corpus) # we return its binary log
      return measure

def log_likelihood(freq_target_corpus: int,target_corpus_
size: int,freq_reference_corpus: int,reference_corpus_size:
int, laplace = 0.1) -> float:
      """Returns the log-likelihood statistic as an
inferential measure of the degree of evidence for keyness

      Keyword arguments:
      freq_target_corpus:    frequency of the term in the
target corpus
      target_corpus_size:    total word count of the target
corpus
      freq_reference_corpus: frequency of the term in the
reference corpus
      reference_corpus_size: total word count of the reference
corpus
      """
      # if there are zero instances of term in reference
corpus, replace count by Laplace smoothing parameter
      if freq_reference_corpus == 0: freq_reference_corpus =
laplace
      other_target_corpus = target_corpus_size - freq_target_
corpus
      other_reference_corpus = reference_corpus_size - freq_
reference_corpus
      freq_both_corpora = freq_target_corpus + freq_reference_
corpus
```

```
      other_both_corpora = other_target_corpus + other_
reference_corpus
      both_corpora_size = target_corpus_size + reference_
corpus_size
      exp_freq_target_corpus = target_corpus_size * freq_both_
corpora / both_corpora_size
      exp_freq_reference_corpus = reference_corpus_size *
freq_both_corpora / both_corpora_size
      exp_other_target_corpus = target_corpus_size * other_
both_corpora / both_corpora_size
      exp_other_reference_corpus = reference_corpus_size *
other_both_corpora / both_corpora_size
      measure = 2 * ( (freq_target_corpus * math.log(freq_
target_corpus/exp_freq_target_corpus)) +
                  (freq_reference_corpus * math.log(freq_
reference_corpus/exp_freq_reference_corpus)) +
                  (other_target_corpus * math.log(other_target_
corpus/exp_other_target_corpus)) +
                  (other_reference_corpus * math.log(other_
reference_corpus/exp_other_reference_corpus))
                  )
      return measure

def keyness(frequencies_corpus: dict[str, int],frequencies_
reference: dict[str, int],measure = "log_ratio") -> dict[str,
int]:
      """Takes two dictionaries of word counts for the target
and reference corpora and returns a keyness measure

      Keyword arguments:
      frequencies_corpus:    a dictionary for the target
corpus with words as keys and their counts as values
      frequencies_reference: an equivalent dictionary for the
reference corpus
      measure:               the statistical measure to
be used (default: 'log_ratio'; also implemented: 'log_
likelihood')
      """
      target_corpus_size = sum(frequencies_corpus.values())
```

```
      reference_corpus_size = sum(frequencies_reference.
values())
      keyness_dict = {}
      # iterate over target dictionary
      for key, value in frequencies_corpus.items():
      # dictionary may contain entries with a zero word count;
we skip those
      if value > 0:
            # get reference corpus count and add zero if
absent
            try:
                  reference_count = frequencies_reference[key]
            except KeyError:
                  reference_count = 0
            # get measure
            if measure == 'log_ratio':
                  keyness_dict[key] = log_ratio(value,target_
corpus_size,reference_count,reference_corpus_size)
            elif measure == 'log_likelihood':
                  keyness_dict[key] = log_
likelihood(value,target_corpus_size,reference_
count,reference_corpus_size)
      return keyness_dict

# create dataframe containing the vocabulary and word
# counts in the reference corpus, then append keyness
computations
df_keywords = pd.DataFrame(corpus_frequencies.
items(),columns=['term','count'])
df_keywords['log_ratio']=keyness(corpus_frequencies,reffreq).
values()
df_keywords['log_likelihood']=keyness(corpus_
frequencies,reffreq,measure='log_likelihood').values()
```

The version in the dataset on Zenodo contains the complete lexicon.

```
df_keywords.to_csv('term_keyness_vs_sibol.csv', header=True,
index=False, encoding='utf-8')
```

In subsequent analyses I employ only those terms meeting the thresholds specified in the Methodology.

```
# filter dataframe by measure threshold
condition_lr = df_keywords['log_ratio']>3.322
condition_ll = df_keywords['log_likelihood']>22.22
df_keywords_filtered=df_keywords[condition_lr & condition_ll]
```

Semantic tagging

I used the Python version of the UCREL Semantic Analysis System (Moore/Rayson 2022) to add semantic tags to the terms in the keyword list:

```
# for installation instructions for PymUSAS, see https://
ucrel.github.io/pymusas/installation
# set up processing pipeline, as per https://ucrel.github.io/
pymusas/usage/how_to/tag_text#english
import spacy
nlp = spacy.load('en_core_web_sm', exclude=['parser', 'ner'])
english_tagger_pipeline = spacy.load('en_dual_none_
contextual')
nlp.add_pipe('pymusas_rule_based_tagger', source=english_
tagger_pipeline)

# generate tags
df_keywords_filtered[,auto_tags']=[nlp(term)[0]._.pymusas_tags
for term in df_keywords_filtered['term']]
```

Terms that were missing from the PymUSAS dictionary and tagged as ‚Z99' were subsequently manually tagged on the basis of existing entries, using:

- for abbreviations, the tags from the equivalent full term (e.g., 'lube' ← 'lubricant')
- for derived vocabulary, the tags for the closest root form in the dictionary (e.g., ‚consensually' ← ‚consensual'; 'matrimony' ← 'matrimonial')
- for synonyms of terms in the keyword list, the tags from those (e.g., ‚conjugal' ← ‚marital')
- for hyponyms of terms in the keyword list, the tags from those ('cyberpunk' ← 'science fiction')
- for other terms, the tags from the closest semantic match, amended for any relevant changes (‚fiancée' ← ‚fiance'; 'Mahabharata' ← 'Bible'; ‚biphobia' ← ‚xenophobia')

The keywords table included in the dataset on Zenodo includes a separate manual_tags column including these additions.
To simplify further analysis, tags were split and deduplicated.

```
df_keywords_filtered['clean_tags']=[sorted(set([j.strip() for
term in tags for j in term.split('/')])) for tags in df_
keywords_filtered.manual_tags]
```

A range column was added representing how many documents a given keyword appears in.

```
# add range
df_keywords_filtered['range']=[sum([freq[term]>0 for freq in
df.frequencies]) for term in df_keywords_filtered.term]

# terms in USAS category Z are proper names and grammatical
terms,
# which can be safely excluded from most analyses
df_keywords_filtered['Z']=[any('Z' in tag for tag in kw) for
kw in df_keywords_filtered.clean_tags]
exclude_Z=df_keywords_filtered['Z']==0
```

The version in the dataset on Zenodo contains these additional columns.

```
df_keywords_filtered.to_csv('keywords_vs_sibol.csv',
header=True, index=False, encoding='utf-8', columns=['term',
'count', 'log_ratio', 'log_likelihood', 'auto_tags', 'manual_
tags', 'clean_tags', 'Z'])
```

Figure setup

All figures in the sections below were produced using the Seaborn library, configured with the following defaults:

```
sns.set_context(,paper, font_scale=1.5)
sns.set_style(,ticks‘)
```

Chronological development

To obtain the data for Figure 1, listing the number of articles published per year, the following code was used:

```
articles_by_year=df.groupby(df.date.dt.year).
agg({'id':'nunique','wordcount_actual':'sum'})
fig, ax1 = plt.subplots(figsize=(12, 6))
```

```
ax2=ax1.twinx()
sns.lineplot(data=articles_by_year.id,ax=ax1)
ax1.set_ylabel('Number of articles (solid)')
sns.lineplot(data=articles_by_year.wordcount_
actual,linestyle='--',ax=ax2)
ax2.yaxis.set_major_formatter(ticker.EngFormatter())
ax2.set_ylabel('Total words (dashed)')
ax1.set_xlabel('Year')
sns.despine()
plt.tight_layout()
fig.savefig('figure_01.png')
```

Geographical distribution

To obtain the data for Table 2, listing the number of articles published per country, the following code was used:

```
articles_by_country=df.groupby('country').
agg({'id':'nunique'}).sort_values(by='id',ascending=False)
total_articles=sum(articles_by_country['id'])
articles_by_country['percentage']=[round(x/total_
articles*100,2) for x in articles_by_country['id']]
```

For the sake of readability, the plot for Figure 2 captures only the top 8 countries by number of articles, using the following code:

```
countries_to_plot=list(articles_by_country[:8].index)
articles_by_country_year=df[df.country.isin(countries_
to_plot)].groupby([df.date.dt.year,'country']).
agg({'id':'nunique','wordcount_actual':'sum'})
fig, ax1 = plt.subplots(figsize=(12, 6))
sns.lineplot(palette='magma_r',data=articles_
by_country_year,x='date',y='wordcount_
actual',hue='country',style='country',size='country',ax=ax1)
ax1.yaxis.set_major_formatter(ticker.EngFormatter())
ax1.set_ylabel('Total words')
ax1.set_xlabel('Year')
sns.despine()
plt.tight_layout()
fig.savefig('figure_02.png')
```

Media and platforms

To obtain the data for Table 2, listing the number of articles per publication, the following code was used:

```
articles_by_publication=df.groupby(['publication_
title','country','medium']).agg({'id':'nunique','wordcount_
actual':['sum','median']}).sort_
values(by=('id','nunique'),ascending=False).head(40)
```

Topics

To obtain the data for Table 3 the following code was used:
I first selected the eight top-level categories with the most keywords:

```
usas_l1_categories=['A','B','C','E','F','G','H','I','K','L','
M','N','O','P','Q','S','T','W','X','Y'] # top-level USAS cate-
gories, minus Z
for usas_cat in usas_l1_categories: df_keywords_filtered[usas_
cat]=[any(usas_cat in tag for tag in kw) for kw in df_keywords_
filtered.clean_tags] # add per-category boolean column
keywords_by_usas_l1=df_keywords_filtered.loc[:,usas_l1][exclu-
de_Z].sum().sort_values(ascending=False).head(8)
```

Within each of those top-level categories, I computed the total keyword count for each subcategory.

```
usas_l2_categories=['A1','A2','A3','A4','A5','A6','A7','A8','
A9','A10','A11','A12','A13','A14','A15','B1','B2','B3','B4',
'B5','C1','E1','E2','E3','E4','E5','E6','F1','F2','F3','F4',
'G1','G2','G3','H1','H2','H3','H4','H5','I1','I2','I3','I4',
'K1','K2','K3','K4','K5','K6','L1','L2','L3','M1','M2','M3',
'M4','M5','M6','M7','M8','N1','N2','N3','N4','N5','N6','O1',
'O2','O3','O4','P1','Q1','Q2','Q3','Q4','S1','S2','S3','S4',
'S5','S6','S7','S8','S9','T1','T2','T3','T4','W1','W2','W3',
'W4','W5','X1','X2','X3','X4','X5','X6','X8','X7','X9','Y1',
'Y2']  # second-level USAS categories, minus Z
usas_l2_categories_filtered=[l2_cat for l2_cat in usas_l2_ca-
tegories if l2_cat[:1] in keywords_by_usas_l1.index]
keywords_by_usas_l2=df_keywords_filtered.loc[:,usas_l2_catego-
ries_filtered][exclude_Z].sum()
```

```
keyword_totals_for_table=keywords_by_usas_l2.sort_valu-
es(ascending=False).head(20)
```

The table provides as well an illustrative selection of the most distinctive keywords under each category:

```
l2_filters={}
for l2_cat in keyword_totals_for_table.index:l2_filters[l2_
cat]=[any(l2_cat in tag for tag in kw) for kw in df_keywords_
filtered.clean_tags]
for l2_cat in l2_filters.keys(): df_keywords_
filtered[l2_filters[l2_cat] & exclude_Z].
loc[:,['term','count','range','log_
ratio','log_likelihood','clean_tags']].sort_
values(by='count',ascending=False).head(6)
```