Final report Big Data

wordcount crawl

Sofie Vos

Introduction

For this project, I wanted to determine how often certain irritating skincare ingredients are mentioned on websites, to highlight which of these ingredients are more well-known than others and which domains contribute to this awareness. I also aimed to improve my coding skills along the way.

To measure the popularity of certain ingredients, I counted the following words (which can also be part of another word):

- fragrance and parfum
- alcohol (hypothesis: most well-known)
- hydroquinone (hypothesis: least well-known, curious to see which domains mention it)
- paraben (hypothesis: somewhere in the middle)
- some sunfilters: oxybenzone, octinoxate, and homosalate (hypothesis: somewhere in the middle)

The following sections of this word counting crawl process are in chronological order: One Warcfile, Multiple Warcfiles, Different Implementations, Results and Findings, Conclusions and Limitations.

One Warcfile

To practice a bit, I aimed to count the number of times the word “and” occurred in the course.warc.gz file in the given WARC Zeppelin notebook after reading through it. I thought the HTML text of this file would barely contain the words I wanted to look for in the end (parfum, paraben, etc., see introduction). So I counted “and”.

I wanted to see what the parsed content looked like and check if there was “and” in it:

```
  val wb = warcs.map{ wr => wr._2.getRecord().getHttpStringBody()}.
             map{ wb => {
                      val d = Jsoup.parse(wb)
                      val bodyText = d.body().text()
                      (d.title(), bodyText)
                  }
              }.
              filter{ case (title, bodyText) =>
bodyText.contains("and") }
```

Yes, the word “and” was there, but there was a lot of CSS styling in the bodyText that could be filtered out. So I thought selecting the inner HTML tags would give me more precise results:

```
   val contentElements = d.select("p, a, abbr, address, b, bdi, bdo,
blockquote, br, caption, cite, code, col, colgroup, data, dd, del,
details, dfn, div, dl, dt, em, figcaption, figure, footer, h1, h2, h3,
h4, h5, h6, header, hgroup, hr, i, img, ins, kbd, label, legend, li,
mark, menu, nav, ol, p, pre, q, rp, rt, ruby, s, samp, small, span,
strong, sub, summary, sup, table, tbody, td, template, textarea, tfoot,
th, thead, time, tr, u, ul, var, wbr").asScala
   val bodyText = contentElements.map(_.text()).mkString(" ")
```

Later I realized this caused double counts (。•̀ᴗ•́。), which made the counts around 15x higher than they should be. Thus, to avoid writing overly complicated code, I stuck to the `Jsoup.body().text()` to retrieve the needed data.

```
import org.jsoup.Jsoup
import org.jsoup.nodes.{Document,Element}
import collection.JavaConverters._

val wb = warcs.map{ wr =>
 val warcfile = wr._1 // null, probably bc they are NullWritables
 val bodyText = wr._2.getRecord().getHttpStringBody()
 val d = Jsoup.parse(bodyText)
 val contentElements = d.select("p, a, abbr, address, b, bdi, bdo,
blockquote, br, caption, cite, code, col, colgroup, data, dd, del,
details, dfn, div, dl, dt, em, figcaption, figure, footer, h1, h2, h3,
h4, h5, h6, header, hgroup, hr, i, img, ins, kbd, label, legend, li,
mark, menu, nav, ol, p, pre, q, rp, rt, ruby, s, samp, small, span,
strong, sub, summary, sup, table, tbody, td, template, textarea, tfoot,
th, thead, time, tr, u, ul, var, wbr").asScala
 // I realised double counting after I wrote this to check
 val cleanedBodyText = contentElements.map(_.text()).mkString(" ")
 (warcfile.toString(), cleanedBodyText)
}.filter{ case (warcfile, cleanedBodyText) =>
cleanedBodyText.contains("and") }
val radboudCount = wb.map { case bodyText =>
 bodyText.split("\\band\\b", -1).length - 1
}.sum()

val spark = SparkSession.builder.appName("Warc Count").getOrCreate()
val warcCountPairs = wb.map { case (warcfile, cleanedBodyText) =>
```

```
 (warcfile.toString, cleanedBodyText.split("\\band\\b", -1).length - 1)
}.toDF("warcfile", "count")

warcCountPairs.show()

//Part of resulting table
+--------+-----+
|warcfile|count|
+--------+-----+
|  (null)|    1|
|  (null)| 1941|
|  (null)|   60|
|  (null)|    1|
|  (null)| 1941|
```

Multiple Warcfiles

With enough confidence in my knowledge, I wanted to start running code on the cluster, as would be wise to do when using more than one warcfile (asked a TA („• ᴗ - „)). Testing my setup, I ran the small wordcount program on the rubigdata-text.txt file in the cluster and got:

```
error: error writing
/opt/hadoop/rubigdata/project/project/target/config-classes/$4be4f66c26
853ab9417a.class: java.nio.file.AccessDeniedException
/opt/hadoop/rubigdata/project/project/target/config-classes/$4be4f66c26
853ab9417a.class
[error] Type error in expression
```

This might have been caused by running "microdnf update -y" as root. Luckily, Arjen quickly responded and told me to reinstall the redbad docker.

I chose the following three warcfiles to continue with on the cluster:


```
{"urlkey":
"com,incidecoder)/products/111skin-celestial-black-diamond-emulsion",
"timestamp": "20210420101127", "url":
"https://incidecoder.com/products/111skin-celestial-black-diamond-emulsion",
"mime": "text/html", "mime-detected": "text/html", "status": "200", "digest":
"T6XQM22OJLUOHNSWUH67R2S4KBNG4GIF", "length": "23031", "offset": "405465130",
"filename":
"crawl-data/CC-MAIN-2021-17/segments/1618039388763.75/warc/CC-MAIN-2021042009
1336-20210420121336-00039.warc.gz", "languages": "eng", "encoding": "UTF-8"}
```

```
{"urlkey": "com,incidecoder)/products/bioderma-hydrabio-light-cream",
"timestamp": "20210420095521", "url":
"https://incidecoder.com/products/bioderma-hydrabio-light-cream", "mime":
"text/html", "mime-detected": "text/html", "status": "200", "digest":
"2H2UCZ4UAGLIZPLXYZLBEV3FJLMFHN2Q", "length": "18762", "offset": "402799716",
"filename":
"crawl-data/CC-MAIN-2021-17/segments/1618039388763.75/warc/CC-MAIN-2021042009
1336-20210420121336-00188.warc.gz", "languages": "eng", "encoding": "UTF-8"}

{"urlkey":
"com,incidecoder)/products/helena-rubinstein-prodigy-powercell-youth-grafter-
the-serum", "timestamp": "20210413051141", "url":
"https://incidecoder.com/products/helena-rubinstein-prodigy-powercell-youth-g
rafter-the-serum", "mime": "text/html", "mime-detected": "text/html",
"status": "200", "digest": "ITI2DUWYMZBDZVLUFTEEBS6ADOJNJWPA", "length":
"24565", "offset": "423187572", "filename":
"crawl-data/CC-MAIN-2021-17/segments/1618038072082.26/warc/CC-MAIN-2021041303
1741-20210413061741-00187.warc.gz", "languages": "eng", "encoding": "UTF-8"}
```


And checked there existence:

```
[hadoop@ed8e6370103f rubigdata]$ hdfs dfs -du -h -s
hdfs:///cc-crawl/segments/1618038072082.26/warc/CC-MAIN-20210413031741-202104
13061741-00187.warc.gz
1.1 G  1.7 G
hdfs:///cc-crawl/segments/1618038072082.26/warc/CC-MAIN-20210413031741-202104
13061741-00187.warc.gz
[hadoop@ed8e6370103f rubigdata]$ hdfs dfs -du -h -s
hdfs:///cc-crawl/segments/1618039388763.75/warc/CC-MAIN-20210420091336-202104
20121336-00188.warc.gz
1.1 G  1.7 G
hdfs:///cc-crawl/segments/1618039388763.75/warc/CC-MAIN-20210420091336-202104
20121336-00188.warc.gz
[hadoop@ed8e6370103f rubigdata]$ hdfs dfs -du -h -s
hdfs:///cc-crawl/segments/1618039388763.75/warc/CC-MAIN-20210420091336-202104
20121336-00039.warc.gz
1.1 G  1.7 G
hdfs:///cc-crawl/segments/1618039388763.75/warc/CC-MAIN-20210420091336-202104
20121336-00039.warc.gz
```

These files contain records from incidecoder.com, a website where you can find the ingredients of products. A friend of mine found a website that shows the webpages of crawled websites: http://web.archive.org/web, which is quite nice to get a better feeling of what pages the records represent. I could see that these files would contain at least the word “parfum”.

It was time to write some code again

```
import org.apache.spark.sql.SparkSession
import org.jsoup.Jsoup
import org.jsoup.nodes.{Document,Element}
import collection.JavaConverters._
import org.apache.hadoop.io.NullWritable
import de.l3s.concatgz.io.warc.{WarcGzInputFormat, WarcWritable}
import de.l3s.concatgz.data.WarcRecord

object RUBigDataApp {
 def main(args: Array[String]) {
   val spark =
SparkSession.builder.appName("RUBigDataApp").getOrCreate()
   val threeFiles =
"hdfs:///cc-crawl/segments/1618038072082.26/warc/CC-MAIN-20210413031741
-20210413061741-00187.warc.gz," +

"hdfs:///cc-crawl/segments/1618039388763.75/warc/CC-MAIN-20210420091336
-20210420121336-00188.warc.gz," +

"hdfs:///cc-crawl/segments/1618039388763.75/warc/CC-MAIN-20210420091336
-20210420121336-00039.warc.gz"

   val sc = spark.sparkContext
   val warcs = sc.newAPIHadoopFile(
           threeFiles,
           classOf[WarcGzInputFormat],             // InputFormat
           classOf[NullWritable],                  // Key
           classOf[WarcWritable]                   // Value
       )

   val validWarcs = warcs.map{ wr => wr._2 }.filter{ _.isValid() }
   val perfumRegex = "\\bperfum\\b".r

   val result = validWarcs.map { wr =>
     val url = wr.getRecord().getHeader().getUrl()
     val body = wr.getRecord().getHttpStringBody()
```

```
      val text = Jsoup.parse(body).body().text()
      val count = perfumRegex.findAllMatchIn(text).length
      (url, count)
    }

    result.collect().foreach { case (url, count) =>
      println(s"URL: $url, Perfum Count: $count")
    }
    spark.stop()
  }
}
```

Which failed as I was foolish enough to not use sbt assembly, but sbt package. Fixing that, it still failed... (๑•́ -•̀)
Apparently (with help from Arjen), the cause of this problem was the URL being null and the `Jsoup.parse(body).body().text()` being null sometimes too. It gave null pointer exceptions in the stderr of the job.

Continuing towards the goal of my project, I added words to count and modified the code many times

```
package org.rubigdata

import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession
import org.jsoup.Jsoup
import org.jsoup.nodes.Element
import collection.JavaConverters._
import org.apache.hadoop.io.NullWritable
import de.l3s.concatgz.io.warc.{WarcGzInputFormat, WarcWritable}
import scala.util.matching.Regex

object RUBigDataApp {
 def main(args: Array[String]) {
   val spark =
SparkSession.builder.appName("RUBigDataApp").getOrCreate()
   import spark.implicits._

   val threeFiles =
"hdfs:///cc-crawl/segments/1618038072082.26/warc/CC-MAIN-20210413031741
-20210413061741-00187.warc.gz," +
```

```
"hdfs:///cc-crawl/segments/1618039388763.75/warc/CC-MAIN-20210420091336
-20210420121336-00188.warc.gz," +

"hdfs:///cc-crawl/segments/1618039388763.75/warc/CC-MAIN-20210420091336
-20210420121336-00039.warc.gz"

   val sc = spark.sparkContext
   val warcs = sc.newAPIHadoopFile(
     threeFiles,
     classOf[WarcGzInputFormat],
     classOf[NullWritable],
     classOf[WarcWritable]
   )

   val validWarcs = warcs.map{wr => wr._2}.filter(_.isValid())

//(?i) bc I also want to count if it's not a separate word and if it
has uppercase letters
   val sunfilterRegex = "(?i)(oxybenzone|octinoxate|homosalate)".r
   val parfumRegex = "(?i)(parfum|fragrance)".r
   val pigmentRegex = "(?i)hydroquinone".r
   val alcoholRegex = "(?i)alcohol".r
   val parabenRegex = "(?i)paraben".r
   val combinedRegex =
"(?i)(oxybenzone|octinoxate|homosalate|parfum|fragrance|hydroquinone|al
cohol|paraben)".r

 //I wanted to filter out if counts below 2 to reduce the output a bit
   def hasAtLeastTwoMatches(text: String, regex: Regex): Boolean = {
     var count = 0
     val matcher = regex.pattern.matcher(text)
     while (matcher.find() && count < 2) {
       count += 1
     }
     count >= 2
   }

   val result = validWarcs.map(_.getRecord()).filter(_.isHttp())
     .filter { rec =>
       val url = rec.getHeader.getUrl
       val body = rec.getHttpStringBody
       //Added null checks to deal with null pointer exceptions
```

```
      val content = if (body != null) Jsoup.parse(body).body() else
null
      val text = if (content != null) content.text() else null
      url != null && url.nonEmpty && text != null &&
hasAtLeastTwoMatches(text, combinedRegex)
    }
    .map { rec =>
      val url = rec.getHeader.getUrl
      val text = Jsoup.parse(rec.getHttpStringBody()).body().text()
      val sunfilterCount = sunfilterRegex.findAllIn(text).length
//using findAllIn() to avoid spending more time splitting substrings
and summing them
      val parfumCount = parfumRegex.findAllIn(text).length
      val pigmentCount = pigmentRegex.findAllIn(text).length
      val alcoholCount = alcoholRegex.findAllIn(text).length
      val parabenCount = parabenRegex.findAllIn(text).length
      (url, alcoholCount, parabenCount, parfumCount, pigmentCount,
sunfilterCount)
    }
    .collect()

  println("URL,Alcohol Count,Paraben Count,Parfum Count,Pigment
Count,Sunfilter Count")
  result.foreach { case (url, alcoholCount, parabenCount, parfumCount,
pigmentCount, sunfilterCount) =>
println(s"$url,$alcoholCount,$parabenCount,$parfumCount,$pigmentCount,$
sunfilterCount")
  }

//for segment, removed the collect() and stdout code and used this to
deal with many results and put in excel:
  // val resultDF = result.toDF("URL", "Alcohol Count", "Paraben
Count", "Parfum Count", "Pigment Count", "Sunfilter Count")
  // resultDF.write.option("header",
"true").csv("hdfs:///user/s1068747/rubigdata-threewarcfiles.csv")

  spark.stop()
 }
}
```

Small part of result:

```
http://513-ohio-ads.com/user/panidarnam-hari,1,0,1,0,0
http://amigospetshopmakassar.com/14-alpo-puppy-12-kg-dog-food-makanan-hewan-amigos-petshop-makassar/,0,0,23,0,0
http://aroma1.ru/muzhskaya_parfyumeriya/dezodorant,0,0,3,0,0
http://b2find.eudat.eu/dataset?tags=prescription+drugs,4,0,0,0,0
http://bekahpogue.com/2018/04/23/be-a-bulb/,0,0,2,0,0
http://boncodepromo.net/english/papa-johns-wings-menu-time-magazine-free-gift-watch/,0,0,2,0,0
http://bursaportali.com/urun/sulfate-standard-for-ic-tracecert-1000-mg-l-sulfate-in-water-100-ml-sigma-aldrich-supelco-anions-and-cations-standards/,3,0,4,0,0
http://d195t5qshxliho.cloudfront.net/nl-be/shop/Hygiene-en-verzorging/Lichaamsverzorging/Handverzorging/Handcreme-Geparfumeerd/p/S2013012500836040000,1,0,4,0,0
http://familyvoicesofminnesota.org/helpful-links/,2,0,0,0,0
```

Cleaning up the result more, instead of URLs, I summed and grouped them by their extracted domain and ran it on a segment.

```
def extractDomain(url: String): String = {
   try {
     new URL(url).getHost.replace("www.", "")
   } catch {
     case _: Exception => url
   }
}
```

As this code took around 29.7 hours in the gold queue, I decided to reduce the input to 20 random warcfiles from the single segment in the cluster plus the three warc files I looked up before and tried running three different implementations, hoping one variant would maybe take less time.

Running different implementations

I tried three different implementations on 23 warcfiles:

1. with the `hasAtLeastTwoMatches()` filter and the Jsoup.parse.body(`rec.getHttpStringBody()`).text() (original code)
2. without `hasAtLeastTwoMatches()` filter and with the Jsoup.parse.body(`rec.getHttpStringBody()`).text()
3. with the `hasAtLeastTwoMatches()` filter and without the Jsoup.parse.body(`rec.getHttpStringBody()`).text(), using `rec.getHttpStringBody()` instead as input data to count on

Implementation 1 took 1 hour, 2 took 1.3 hours, and 3 took even longer. We can conclude from this that pre-counting 2 words for filtering probably takes up much more time in the long run than not doing it. In addition, the `Jsoup.parse` function helps in reducing the running time as less input text is counted on.

Results and Findings

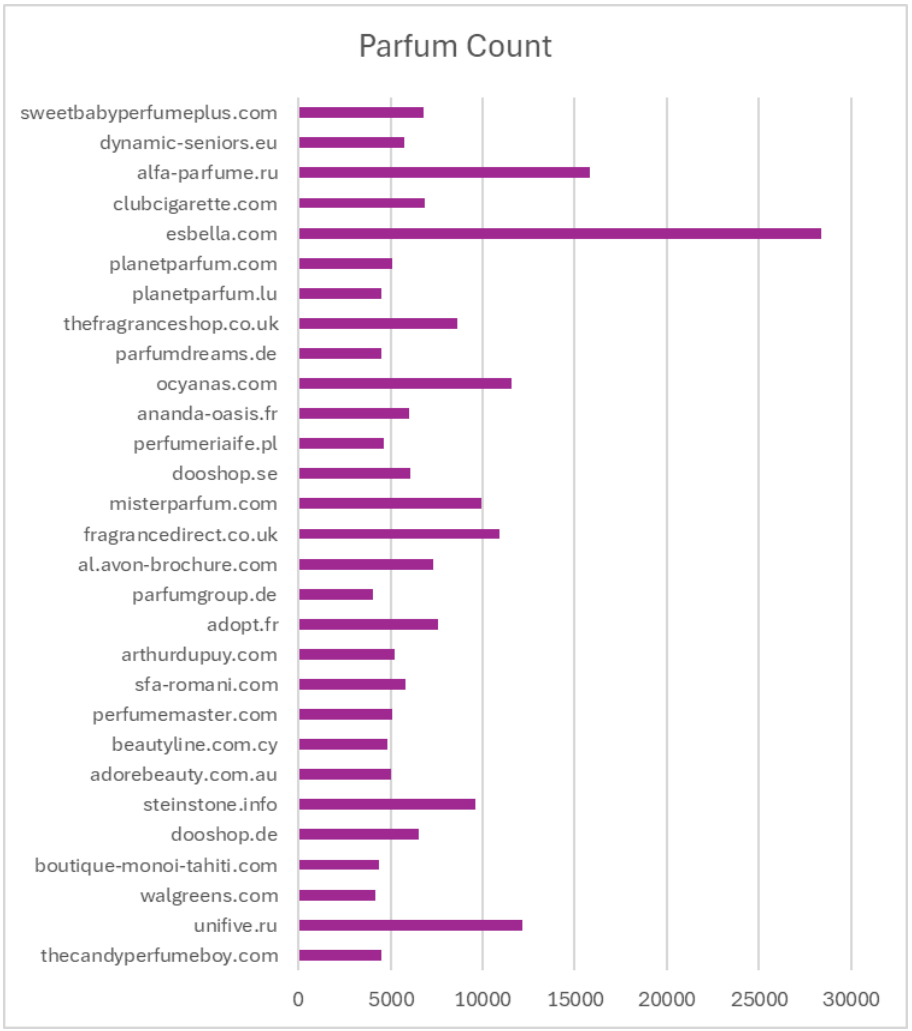
It was quite easy to combine the resulting files and get the results locally with “hdfs dfs -get” and “docker cp”. As I tried to make charts, it crashed in LibreOffice Calc and in Excel at first as there was way too much data for it to handle. So I used commands similar to:

```
sofievos@pop-os:~/Downloads/project_results$ awk -F',' '$2 > 1000 {print $1","$2}' combined_file.csv >> alcohol_above_1000.csv
```

to create charts showing the highest few counts from the segment.

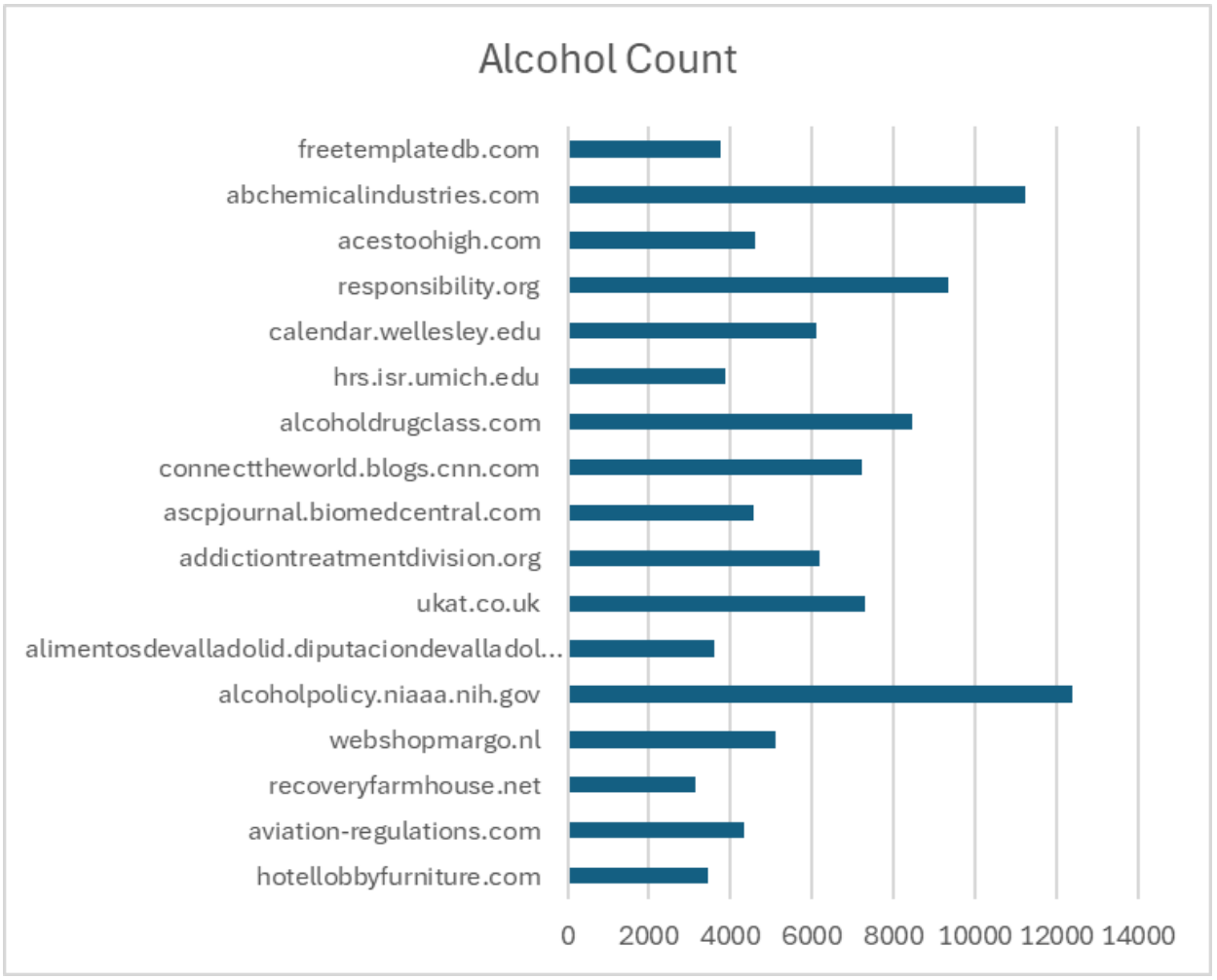
parfum/fragrance

total counted words: 911718



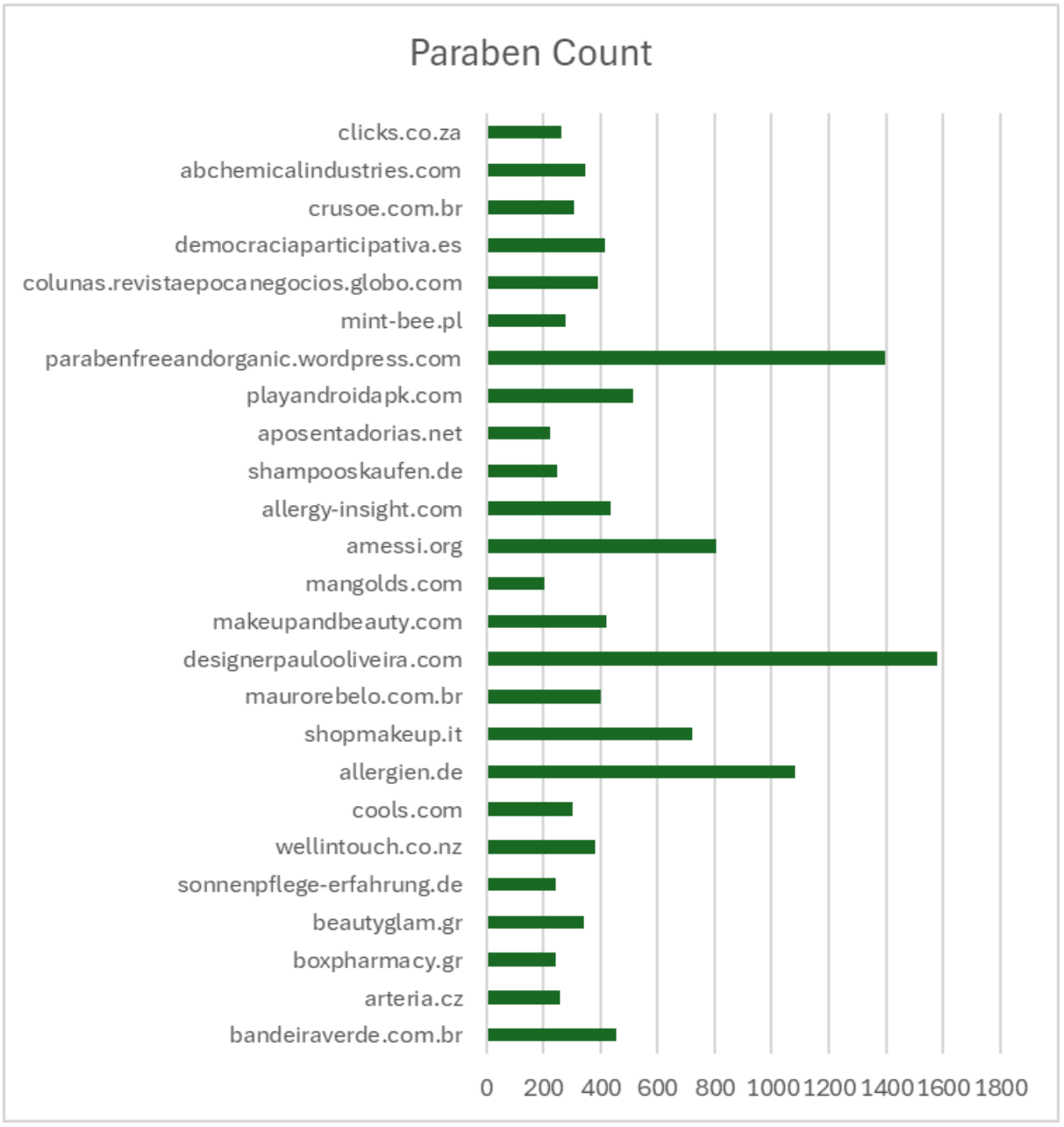
Most of these are websites where you can buy parfum.

alcohol
total word count: 737508



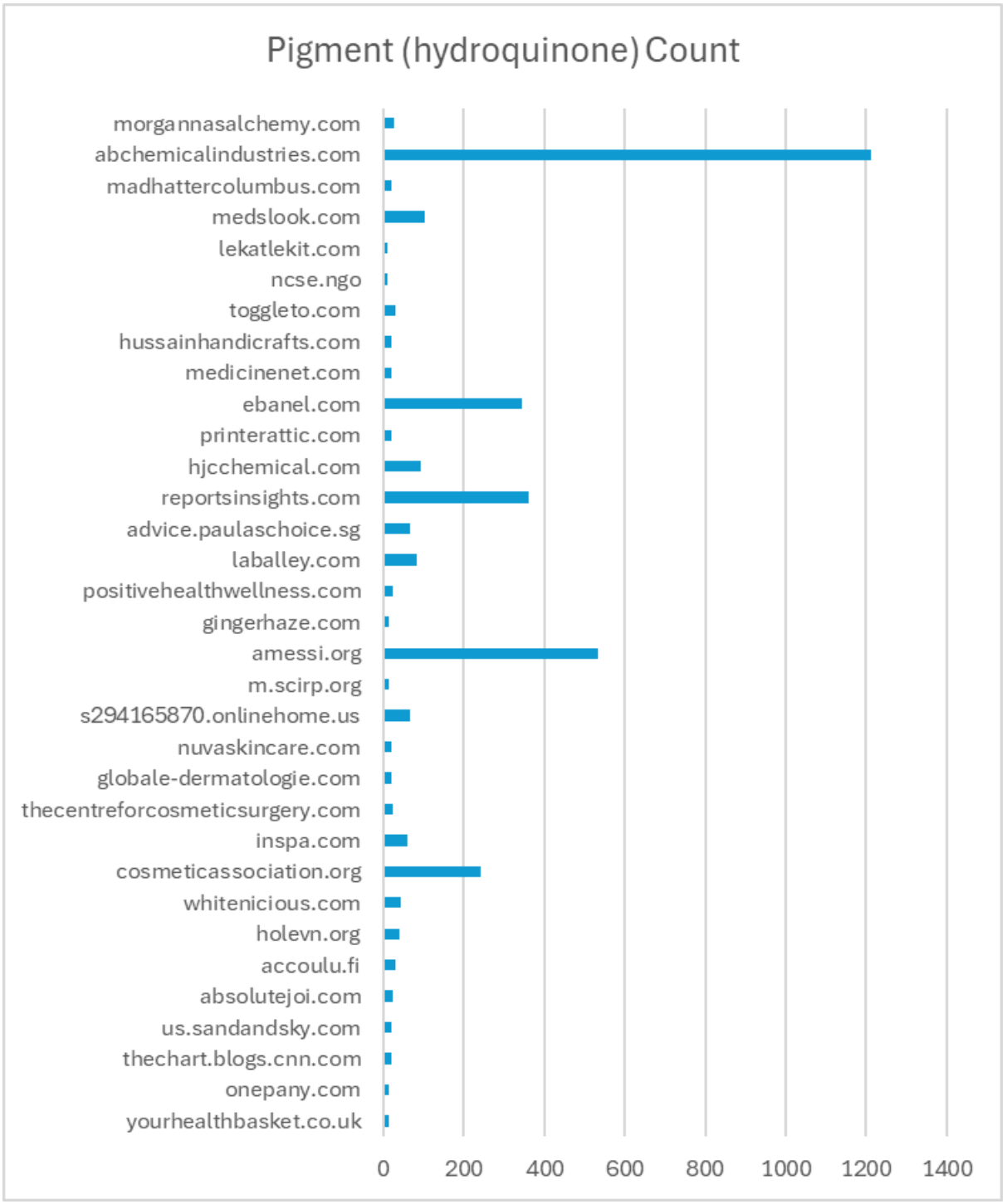
Way less shops than for parfum/fragrance. More websites about other things like news, addiction, blogs etc.

paraben
total counted words: 43188



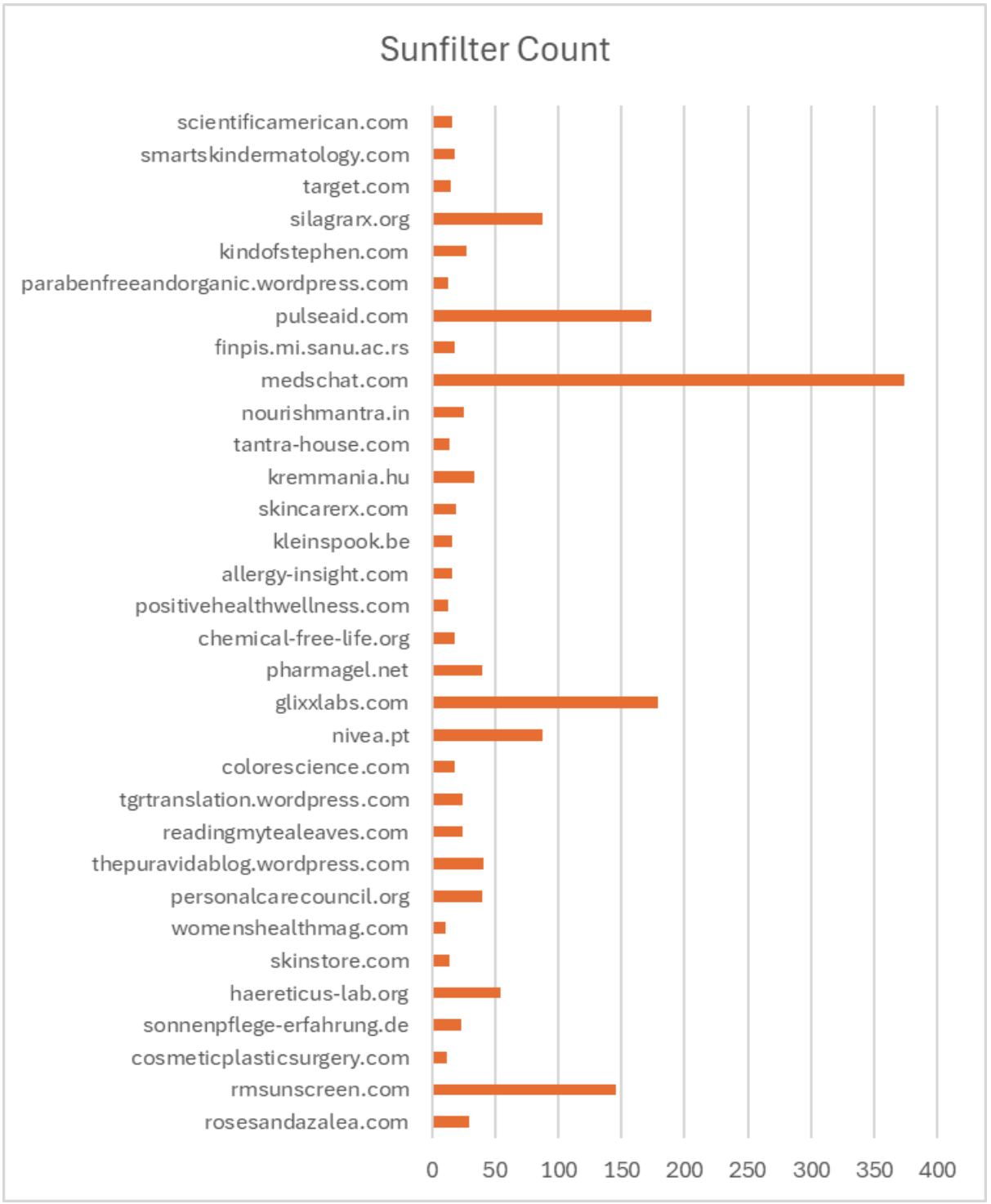
These resulting domains are more about allergies and make-up.

hydroquinone
total counted words: 4483



Mostly chemistry related websites.

oxybenzone/octinoxate/homosalate (bad sunfilters)
total counted words: 2430



Mostly stores, like Nivea , and chemistry websites.

Conclusions and Limitations

It was cool to see the range of websites that gave high word counts in the results. The variety of websites in the alcohol count was particularly notable. Contrary to my hypothesis that alcohol would have the highest count, it turned out that parfum/fragrance was mentioned more frequently. Additionally, the hydroquinone count was higher than the sunfilters, likely because it is discussed way more on chemistry-related websites than the sunfilters were mentioned on such and in webshops.

The counting of words in the HTML body tag does not account for any dynamically generated text with JavaScript. Furthermore, some websites are probably visited much more frequently, which is not reflected in the word counts and thus does not accurately describe the popularity of any of the words I counted. Counting the words provides more of an approximation, and other factors must be considered to measure popularity accurately. As mentioned earlier, alcohol is discussed on a wider variety of websites, which may indicate a higher popularity.

I had fun exploring the crawl and I want to thank you for reading. Wishing you a sweet summer vacation!