Nontabular Data

Kylie Ariel Bemis

11/7/2017

Why nontabular data?

So far we have primarily discussed tabular data (i.e., data that can be fit described by a table of values organized into rows and columns). However, this does not fit all data that you may encounter in data science.

When working with nontabular data, the data may be representable in a tabular format, but more easily stored in a nontabular format. In this case, you may often end up converting the data to a tabular format in order to work with it.

However, sometimes the data is inherently easier to work with in a nontabular format, and you will import and continue to work with the data in nontabular format for analysis as well.

Unstructured and semi-structured data

- Text data
  - Books
  - Emails
  - Tweets
  - Blogs
  - Facebook posts

Structured data

- Tabular data (.csv, tab-delimited, etc.)
- Relational data (databases)
  - Composed of multiple tables of tabular data
- Hierarchical data (XML)
- Network/graph data
- High-dimensional data
  - May be tabular, but often isn't
- Spatial data
  - May be tabular, but often isn't
- Multimedia (images, audio, movies)
- Binary data
  - May be any of the above

Unstructured data in R

A (very) brief introduction to working with text data in R

Unstructured text data must be transformed into some kind of structured data for analysis. A common way of doing this is mapping documents into a vector space that defines features on aspects of the document such as words, stems, n-grams etc. This results in a document-term matrix, with rows as individual documents and columns as the features (e.g., word and their counts).

There are a wide number of R packages for working with text and natural language data (https://cran.r-project.org/web/views/NaturalLanguageProcessing.html), but we will focus on `tidytext`.

Rather than a document-term matrix, `tidytext` uses an approach that mirrors the `tidyverse`, and uses a "one-token-per-row" approach for tidying text data.

Text data in R using `tidytext`

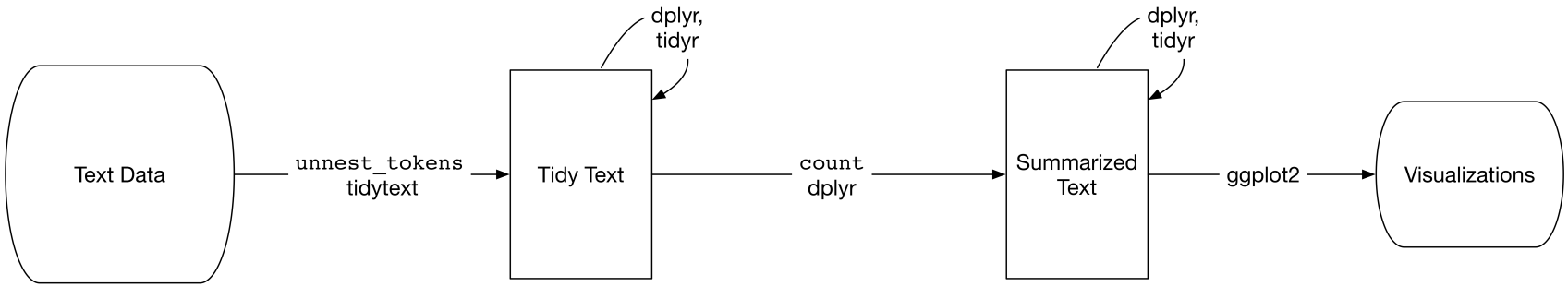


Figure 1: *Text Mining with R* by Julia Silge and David Robinson

Book freely available at http://tidytextmining.com.

```
library(tidytext)
text1 <- readLines("prideprejudice.txt")
head(text1, n=10)
```

```
##  [1] "PRIDE AND PREJUDICE"
##  [2] ""
##  [3] "By Jane Austen"
##  [4] ""
##  [5] ""
##  [6] ""
##  [7] "Chapter 1"
##  [8] ""
##  [9] ""
## [10] "It is a truth universally acknowledged, that a single m
```

```
prideprejudice <- tibble(line=seq_along(text1), text=text1)
```

```
prideprejudice %>%
  unnest_tokens(word, text)
```

```
## # A tibble: 122,204 x 2
##     line      word
##    <int>     <chr>
##  1     1     pride
##  2     1       and
##  3     1 prejudice
##  4     3        by
##  5     3      jane
##  6     3    austen
##  7     7   chapter
##  8     7         1
##  9    10        it
## 10    10        is
## # ... with 122,194 more rows
```

```
prideprejudice %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2)
```

```
## # A tibble: 111,489 x 2
##     line                    bigram
##    <int>                     <chr>
##  1     1                pride and
##  2     1            and prejudice
##  3     3                  by jane
##  4     3              jane austen
##  5     7                chapter 1
##  6    10                   it is
##  7    10                    is a
##  8    10                  a truth
##  9    10         truth universally
## 10    10 universally acknowledged
## # ... with 111,479 more rows
```

```
stop_words
```

```
## # A tibble: 1,149 x 2
##           word lexicon
##          <chr>   <chr>
##  1           a   SMART
##  2         a's   SMART
##  3        able   SMART
##  4       about   SMART
##  5       above   SMART
##  6   according   SMART
##  7 accordingly   SMART
##  8      across   SMART
##  9    actually   SMART
## 10       after   SMART
## # ... with 1,139 more rows
```

```r
prideprejudice1 <- prideprejudice %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words, by="word")
```

```
prideprejudice1 %>%
  count(word, sort=TRUE)
```

```
## # A tibble: 6,018 x 2
##         word     n
##        <chr> <int>
##  1 elizabeth   597
##  2     darcy   373
##  3    bennet   294
##  4      miss   283
##  5      jane   264
##  6   bingley   257
##  7      time   203
##  8      lady   183
##  9    sister   180
## 10   wickham   162
## # ... with 6,008 more rows
```

Structured data in R

There are many types of structured data, and many R packages that exist to read them.

Today, we will focus on:

- JSON (javascript object notation) using the package `jsonlite`
- XML (eXstensible markup language) using the package `xml2`
- Binary data using the `readBin` function from base R

JSON data in R using `jsonlite`

The package `jsonlite` provides a mapping to and from JSON objects and R objects.

Notable functions include:

- `toJSON` and `fromJSON` for converting between JSON and R objects
- `read_json` and `write_json` for reading/writing JSON from/to files
- `stream_in` and `stream_out` for streaming ndjson (newline-delimited JSON)

Paper avaiable at https://arxiv.org/abs/1403.2805.

```r
library(jsonlite)
```

```
##
## Attaching package: 'jsonlite'

## The following object is masked from 'package:purrr':
##
##     flatten
```

```r
write_json(mpg, "mpg.json", pretty=TRUE)
stream_out(mpg, file("mpg_nd.json"))
```

```
## opening file output connection.

##
Complete! Processed total of 234 rows.

## closing file output connection.
```

```
mpg2 <- read_json("mpg.json", simplifyDataFrame=TRUE)
head(mpg2)
```

```
##   manufacturer model displ year cyl      trans drv cty hwy fl
## 1         audi    a4   1.8 1999   4   auto(l5)   f  18  29  p
## 2         audi    a4   1.8 1999   4 manual(m5)   f  21  29  p
## 3         audi    a4   2.0 2008   4 manual(m6)   f  20  31  p
## 4         audi    a4   2.0 2008   4   auto(av)   f  21  30  p
## 5         audi    a4   2.8 1999   6   auto(l5)   f  16  26  p
## 6         audi    a4   2.8 1999   6 manual(m5)   f  18  26  p
```

```r
mpg3 <- toJSON(mpg2[1,], pretty=TRUE)
mpg3
```

```
## [
##   {
##     "manufacturer": "audi",
##     "model": "a4",
##     "displ": 1.8,
##     "year": 1999,
##     "cyl": 4,
##     "trans": "auto(l5)",
##     "drv": "f",
##     "cty": 18,
##     "hwy": 29,
##     "fl": "p",
##     "class": "compact"
##   }
## ]
```

```
mpg4 <- fromJSON(mpg3)
mpg4
```

```
##    manufacturer model displ year cyl    trans drv cty hwy fl
## 1          audi    a4   1.8 1999   4 auto(l5)   f  18  29  p c
```

```r
con <- file("mpg_nd.json")
mpg5 <- stream_in(con)
```

```
## opening file input connection.

##
 Found 234 records...
 Imported 234 records. Simplifying...

## closing file input connection.
```

```r
head(mpg5)
```

```
##   manufacturer model displ year cyl      trans drv cty hwy fl
## 1         audi    a4   1.8 1999   4   auto(l5)   f  18  29  p
## 2         audi    a4   1.8 1999   4 manual(m5)   f  21  29  p
## 3         audi    a4   2.0 2008   4 manual(m6)   f  20  31  p
## 4         audi    a4   2.0 2008   4   auto(av)   f  21  30  p
## 5         audi    a4   2.8 1999   6   auto(l5)   f  16  26  p
## 6         audi    a4   2.8 1999   6 manual(m5)   f  18  26  p
```

XML data in R using `xml2`

The package `xml2` is a non-default member of the `tidyverse`, and is designed for parsing XML data with the pipe %>% operator.

Notable functions include:

- `read_xml` and `write_xml` for reading and writing XML
- `xml_child`, `xml_children`, `xml_sibling`,and `xml_parent` for navigating the hierarchical XML parse tree
- `xml_attr`, `xml_attrs`, and `xml_text` for extracting information from nodes

Note that many of the functions in `xml2` are vectorized and work on both a single node and on sets of nodes.

```r
library(xml2)
plant <- read_xml("plant_catalog.xml")
plant
```

```
## {xml_document}
## <CATALOG>
##  [1] <PLANT>\n  <COMMON>Bloodroot</COMMON>\n  <BOTANICAL>Sang
##  [2] <PLANT>\n  <COMMON>Columbine</COMMON>\n  <BOTANICAL>Aqui
##  [3] <PLANT>\n  <COMMON>Marsh Marigold</COMMON>\n  <BOTANICAL
##  [4] <PLANT>\n  <COMMON>Cowslip</COMMON>\n  <BOTANICAL>Caltha
##  [5] <PLANT>\n  <COMMON>Dutchman's-Breeches</COMMON>\n  <BOTA
##  [6] <PLANT>\n  <COMMON>Ginger, Wild</COMMON>\n  <BOTANICAL>A
##  [7] <PLANT>\n  <COMMON>Hepatica</COMMON>\n  <BOTANICAL>Hepat
##  [8] <PLANT>\n  <COMMON>Liverleaf</COMMON>\n  <BOTANICAL>Hepa
##  [9] <PLANT>\n  <COMMON>Jack-In-The-Pulpit</COMMON>\n  <BOTAN
## [10] <PLANT>\n  <COMMON>Mayapple</COMMON>\n  <BOTANICAL>Podop
## [11] <PLANT>\n  <COMMON>Phlox, Woodland</COMMON>\n  <BOTANICA
## [12] <PLANT>\n  <COMMON>Phlox, Blue</COMMON>\n  <BOTANICAL>Ph
## [13] <PLANT>\n  <COMMON>Spring-Beauty</COMMON>\n  <BOTANICAL>
## [14] <PLANT>\n  <COMMON>Trillium</COMMON>\n  <BOTANICAL>Trill
## [15] <PLANT>\n  <COMMON>Wake Robin</COMMON>\n  <BOTANICAL>Tri
## [16] <PLANT>\n  <COMMON>Violet, Dog-Tooth</COMMON>\n  <BOTANI
```

```
plant %>%
  xml_child()
```

```
## {xml_node}
## <PLANT>
## [1] <COMMON>Bloodroot</COMMON>
## [2] <BOTANICAL>Sanguinaria canadensis</BOTANICAL>
## [3] <ZONE>4</ZONE>
## [4] <LIGHT>Mostly Shady</LIGHT>
## [5] <PRICE>$2.44</PRICE>
## [6] <AVAILABILITY>031599</AVAILABILITY>
```

```r
plant %>%
  xml_child(2)
```

```
## {xml_node}
## <PLANT>
## [1] <COMMON>Columbine</COMMON>
## [2] <BOTANICAL>Aquilegia canadensis</BOTANICAL>
## [3] <ZONE>3</ZONE>
## [4] <LIGHT>Mostly Shady</LIGHT>
## [5] <PRICE>$9.37</PRICE>
## [6] <AVAILABILITY>030699</AVAILABILITY>
```

```
plant %>%
  xml_child() %>%
  xml_child("COMMON") %>%
  xml_text()
```

```
## [1] "Bloodroot"
```

```
plant %>%
  xml_child() %>%
  xml_child("BOTANICAL") %>%
  xml_text()
```

```
## [1] "Sanguinaria canadensis"
```

```
plant %>%
  xml_child() %>%
  xml_children()
```

```
## {xml_nodeset (6)}
## [1] <COMMON>Bloodroot</COMMON>
## [2] <BOTANICAL>Sanguinaria canadensis</BOTANICAL>
## [3] <ZONE>4</ZONE>
## [4] <LIGHT>Mostly Shady</LIGHT>
## [5] <PRICE>$2.44</PRICE>
## [6] <AVAILABILITY>031599</AVAILABILITY>
```

```
plant %>%
  xml_child() %>%
  xml_children() %>%
  xml_text()
```

```
## [1] "Bloodroot"                "Sanguinaria canadensis"
## [3] "4"                        "Mostly Shady"
## [5] "$2.44"                    "031599"
```

```
plant %>%
  xml_child() %>%
  xml_children() %>%
  xml_name()
```

```
## [1] "COMMON"       "BOTANICAL"    "ZONE"         "LIGHT"
## [5] "PRICE"        "AVAILABILITY"
```

```
to_row <- function(nodeset) {
  row <- as.list(xml_text(nodeset))
  names(row) <- xml_name(nodeset)
  as_tibble(row)
}

plant %>%
  xml_child() %>%
  xml_children() %>%
  to_row()
```

```
## # A tibble: 1 x 6
##      COMMON                BOTANICAL  ZONE         LIGHT PRICE A
##       <chr>                    <chr> <chr>         <chr> <chr>
## 1 Bloodroot Sanguinaria canadensis     4 Mostly Shady $2.44
```

```
plant %>%
  xml_children() %>%
  map_dfr(~ xml_children(.) %>% to_row())
```

```
## # A tibble: 36 x 6
##                    COMMON                  BOTANICAL  ZONE         LI
##                     <chr>                      <chr> <chr>         <c
##  1            Bloodroot Sanguinaria canadensis     4 Mostly Sh
##  2            Columbine   Aquilegia canadensis     3 Mostly Sh
##  3      Marsh Marigold       Caltha palustris     4 Mostly Su
##  4              Cowslip       Caltha palustris     4 Mostly Sh
##  5 Dutchman's-Breeches    Dicentra cucullaria     3 Mostly Sh
##  6        Ginger, Wild       Asarum canadense     3 Mostly Sh
##  7            Hepatica     Hepatica americana     4 Mostly Sh
##  8            Liverleaf     Hepatica americana     4 Mostly Sh
##  9  Jack-In-The-Pulpit    Arisaema triphyllum     4 Mostly Sh
## 10            Mayapple   Podophyllum peltatum     3 Mostly Sh
## # ... with 26 more rows, and 1 more variables: AVAILABILITY <
```

Binary data in R using `readBin`

Base R provides the `readBin` function for reading binary data in R.

It is helpful to remember how basic data types and bytes work when using it.

bytes	C type	R type
1	char	raw
4	int	integer
8	double	numeric

It is sometimes necessary to read integral and floating point types that don't map directly to base R types.

```
f <- file("test.bin", "w+b")
writeBin(c(1L,2L,3L), f)
writeBin(c(4.44), f)
writeBin(c("hello", "world"), f)
close(f)
```

```r
f <- file("test.bin", "r+b")
readBin(f, "integer", n=3)
```

```
## [1] 1 2 3
```

```r
seek(f) # 0 + 3 ints x 4 bytes = 12 bytes
```

```
## [1] 12
```

```r
readBin(f, "double", n=1)
```

```
## [1] 4.44
```

```r
seek(f) # 12 + 1 double x 8 bytes = 20 bytes
```

```
## [1] 20
```

```r
readBin(f, "character", n=2)
```

```
## [1] "hello" "world"
```

```r
seek(f) # why 32 and not 30 (20 + 5 char x 1 byte)?
```

```
## [1] 32
```

```r
close(f)
```

```r
f <- file("test.bin", "r+b")
seek(f, 12) # 0 + 3 ints x bytes 4 = 12 bytes
```

```
## [1] 0
```

```r
readBin(f, "double", n=1)
```

```
## [1] 4.44
```

```r
close(f)
```

```r
f <- file("test.bin", "r+b")
seek(f, 26) # 0 + 3 x 4 + 8 + 6 = 26 bytes
```

```
## [1] 0
```

```r
readBin(f, "character", n=1)
```

```
## [1] "world"
```

```r
close(f)
```

Real-life example: MS Imaging Data

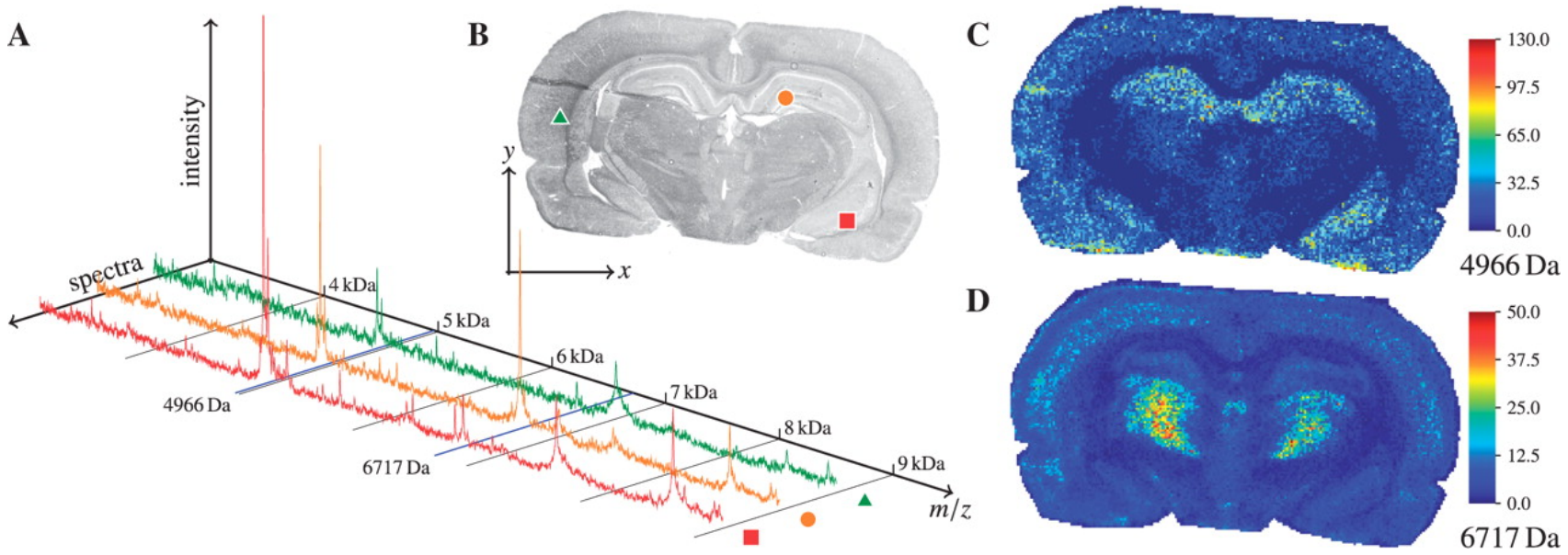


Figure 2: Alexandrov and Kobarg, 2011

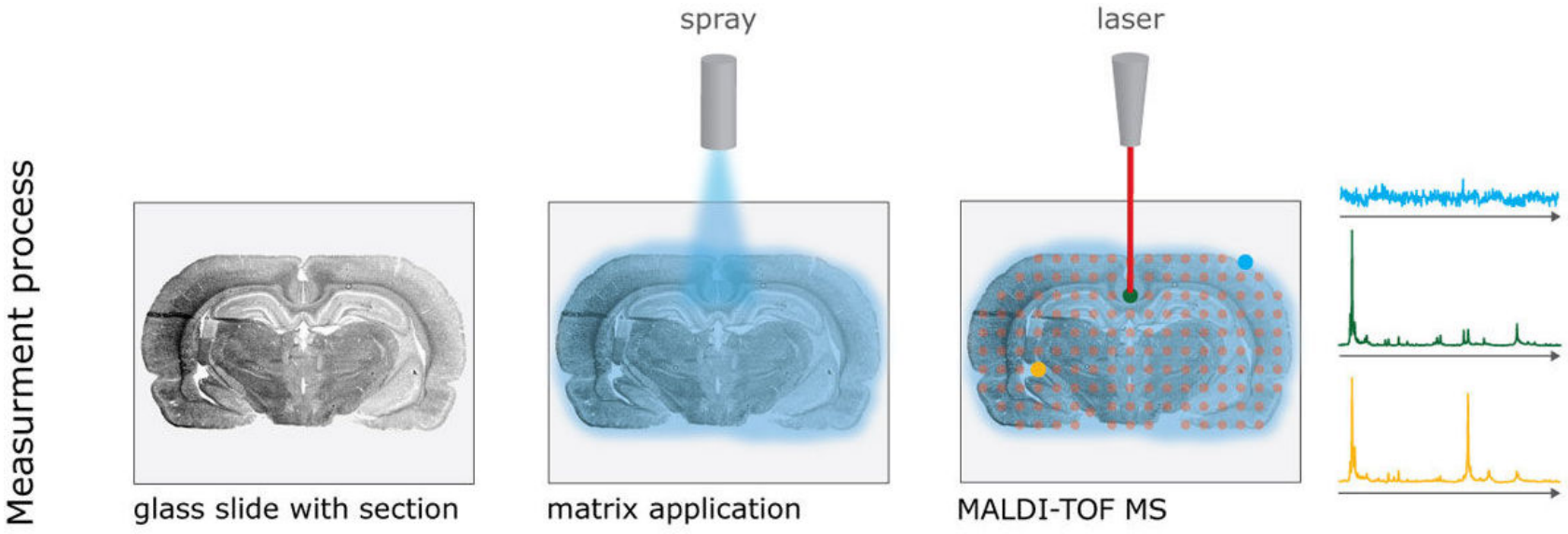


Figure 3: Alexandrov, 2012

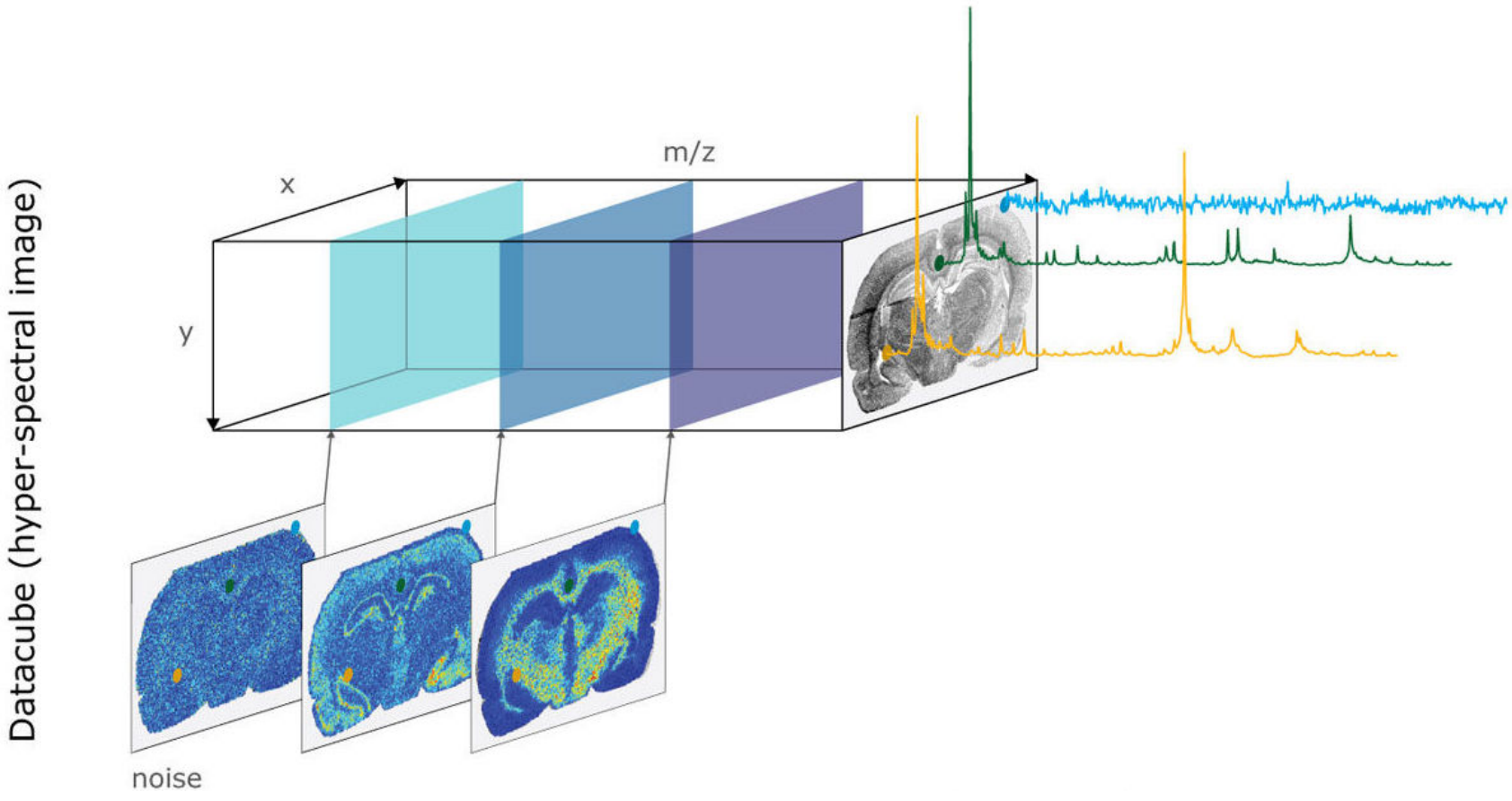


Figure 4: Alexandrov, 2012

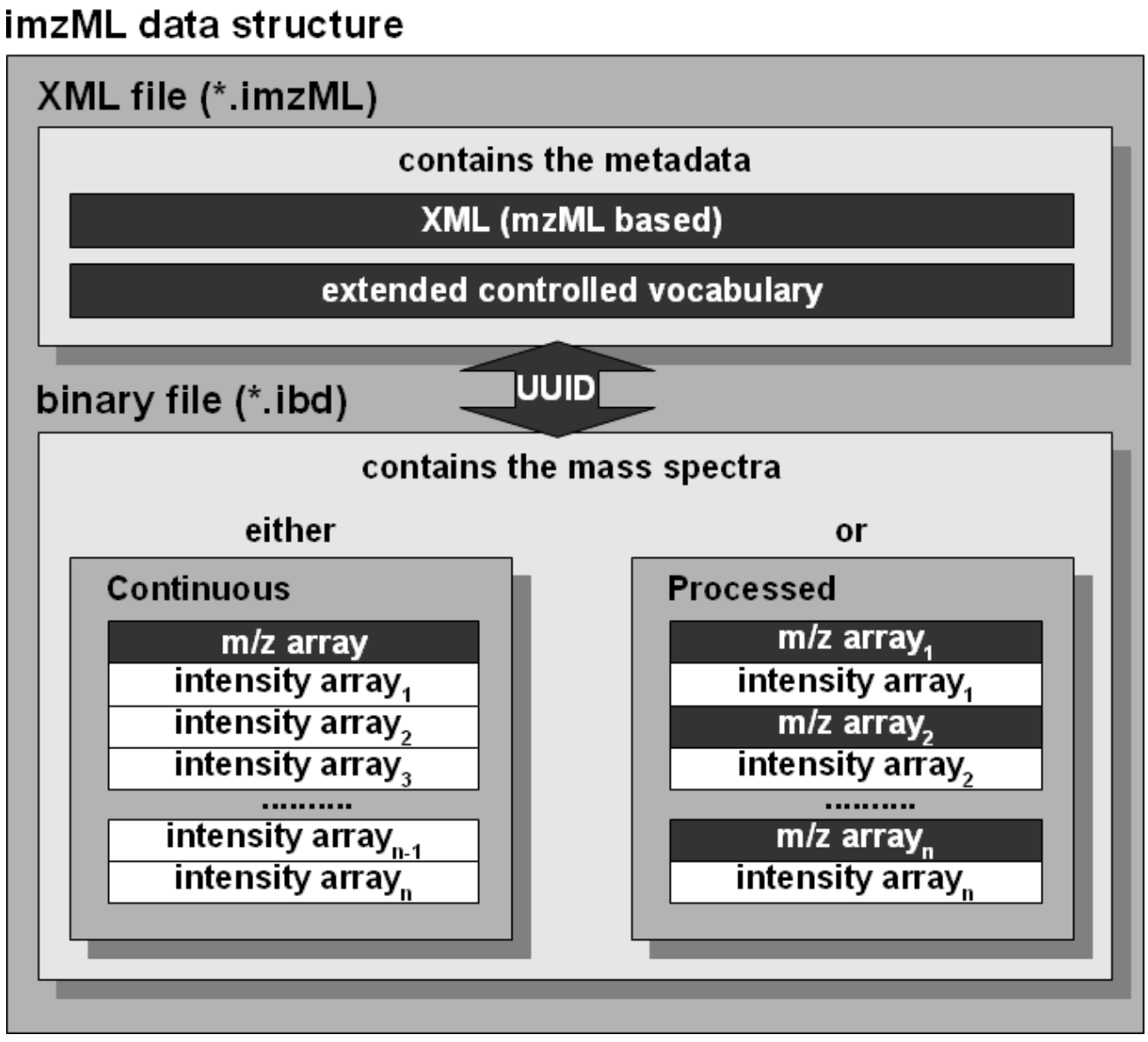


Figure 5: https://imzml.org

```r
library(xml2)
imzml <- read_xml("../../Homework/HW3/Example_Continuous.imzML")
imzml
```

```
## {xml_document}
## <mzML schemaLocation="http://psi.hupo.org/ms/mzml http://psid
## [1] <cvList count="3">\n  <cv id="MS" fullName="Proteomics St
## [2] <fileDescription>\n  <fileContent>\n    <cvParam cvRef="M
## [3] <referenceableParamGroupList count="4">\n  <referenceable
## [4] <sampleList count="1">\n  <sample id="sample1" name="Samp
## [5] <softwareList count="2">\n  <software id="Xcalibur" versi
## [6] <scanSettingsList count="1">\n  <scanSettings id="scanset
## [7] <instrumentConfigurationList count="1">\n  <instrumentCon
## [8] <dataProcessingList count="2">\n  <dataProcessing id="Xca
## [9] <run defaultInstrumentConfigurationRef="LTQFTUltra0" defa
```

```
imzml %>%
  xml_child("d1:run") %>%
  xml_child("d1:spectrumList")
```

```
## {xml_node}
## <spectrumList count="9" defaultDataProcessingRef="XcaliburPro
## [1] <spectrum id="Scan=1" defaultArrayLength="0" index="0">\n
## [2] <spectrum id="Scan=2" defaultArrayLength="0" index="1">\n
## [3] <spectrum id="Scan=3" defaultArrayLength="0" index="2">\n
## [4] <spectrum id="Scan=4" defaultArrayLength="0" index="3">\n
## [5] <spectrum id="Scan=5" defaultArrayLength="0" index="4">\n
## [6] <spectrum id="Scan=6" defaultArrayLength="0" index="5">\n
## [7] <spectrum id="Scan=7" defaultArrayLength="0" index="6">\n
## [8] <spectrum id="Scan=8" defaultArrayLength="0" index="7">\n
## [9] <spectrum id="Scan=9" defaultArrayLength="0" index="8">\n
```

```
imzml %>%
  xml_child("d1:referenceableParamGroupList")
```

```
## {xml_node}
## <referenceableParamGroupList count="4">
## [1] <referenceableParamGroup id="mzArray">\n  <cvParam cvRef=
## [2] <referenceableParamGroup id="intensityArray">\n  <cvParam
## [3] <referenceableParamGroup id="scan1">\n  <cvParam cvRef="M
## [4] <referenceableParamGroup id="spectrum1">\n  <cvParam cvRe
```

```r
insert_ref_groups <- function(x) {
  ref_groups <- xml_root(x) %>%
    xml_child("d1:referenceableParamGroupList") %>%
    xml_children()
  ref <- xml_child(x, "d1:referenceableParamGroupRef")
  name <- xml_attr(ref, "ref")
  ref_groups_exist <- xml_attr(ref_groups, "id") %in% name
  if ( any(ref_groups_exist) )
    group <- ref_groups[[which(ref_groups_exist)]]
  for ( g in xml_children(group) )
    xml_add_child(x, g)
  xml_remove(ref)
  x
}
```

```r
xml_find_by_attribute <- function(x, attr, value) {
  match <- xml_attr(x, attr) == value
  if ( isTRUE(any(match)) ) {
    x[[which(match)]]
  } else {
    NULL
  }
}
```

```r
get_spectrum_data <- function(x, i) {
  spectrum <- x %>%
    xml_child("d1:run") %>%
    xml_child("d1:spectrumList") %>%
    xml_child(i)
  spectrum <- insert_ref_groups(spectrum)
  scan <- spectrum %>%
    xml_child("d1:scanList") %>%
    xml_child("d1:scan")
  scan <- insert_ref_groups(scan)
  data <- spectrum %>%
    xml_child("d1:binaryDataArrayList") %>%
    xml_children()
  for ( d in data ) insert_ref_groups(d)
  data <- lapply(data, xml_children)
  for ( i in seq_along(data) ) {
    if ( !is.null(xml_find_by_attribute(data[[i]], "name", "m/z array")
      names(data)[i] <- "mz"
    if ( !is.null(xml_find_by_attribute(data[[i]], "name", "intensity a
      names(data)[i] <- "intensity"
  }
  data$coord <- xml_children(scan)
  data[c("mz", "intensity", "coord")]
}
```

```r
get_spectra_n <- function(x) {
  x %>%
    xml_child("d1:run") %>%
    xml_child("d1:spectrumList") %>%
    xml_attr("count") %>%
    as.numeric()
}

get_spectra <- function(x) {
  n <- get_spectra_n(x)
  lapply(1:n, function(i) get_spectrum_data(x, i))
}
```

```
x <- imzml

get_spectra_n(x)
```

```
## [1] 9
```

```
spectra_info <- get_spectra(x)
```

```
spectra_info[[1]]
```

```
## $mz
## {xml_nodeset (8)}
## [1] <cvParam cvRef="IMS" accession="IMS:1000103" name="extern
## [2] <cvParam cvRef="IMS" accession="IMS:1000102" name="extern
## [3] <cvParam cvRef="IMS" accession="IMS:1000104" name="extern
## [4] <binary/>
## [5] <cvParam xmlns="http://psi.hupo.org/ms/mzml" cvRef="MS" a
## [6] <cvParam xmlns="http://psi.hupo.org/ms/mzml" cvRef="MS" a
## [7] <cvParam xmlns="http://psi.hupo.org/ms/mzml" cvRef="IMS"
## [8] <cvParam xmlns="http://psi.hupo.org/ms/mzml" cvRef="MS" a
##
## $intensity
## {xml_nodeset (8)}
## [1] <cvParam cvRef="IMS" accession="IMS:1000103" name="extern
## [2] <cvParam cvRef="IMS" accession="IMS:1000102" name="extern
## [3] <cvParam cvRef="IMS" accession="IMS:1000104" name="extern
## [4] <binary/>
## [5] <cvParam xmlns="http://psi.hupo.org/ms/mzml" cvRef="MS" a
## [6] <cvParam xmlns="http://psi.hupo.org/ms/mzml" cvRef="MS" a
## [7] <cvParam xmlns="http://psi.hupo.org/ms/mzml" cvRef="IMS"
```

```r
coord_x <- spectra_info %>%
  map_dbl(~ xml_find_by_attribute(.$coord, "name", "position x") %>%
        xml_attr("value") %>%
          as.numeric())

coord_y <- spectra_info %>%
  map_dbl(~ xml_find_by_attribute(.$coord, "name", "position y") %>%
        xml_attr("value") %>%
          as.numeric())

coord_x
```

```
## [1] 1 2 3 1 2 3 1 2 3
```

```r
coord_y
```

```
## [1] 1 1 1 2 2 2 3 3 3
```

```r
mz_length <- spectra_info[[1]]$mz %>%
  xml_find_by_attribute("name", "external array length") %>%
  xml_attr("value") %>%
  as.numeric()

mz_offset <- spectra_info[[1]]$mz %>%
  xml_find_by_attribute("name", "external offset") %>%
  xml_attr("value") %>%
  as.numeric()

mz_length
```

```
## [1] 8399
```

```r
mz_offset
```

```
## [1] 16
```

```
intensity_length <- spectra_info %>%
  map_dbl(~ xml_find_by_attribute(.$intensity,
                                  "name",
                                  "external array length") %>%
  xml_attr("value") %>%
  as.numeric())

intensity_offset <- spectra_info %>%
  map_dbl(~ xml_find_by_attribute(.$intensity,
                                  "name",
                                  "external offset") %>%
  xml_attr("value") %>%
  as.numeric())

intensity_length
```

```
## [1] 8399 8399 8399 8399 8399 8399 8399 8399 8399
```

```
intensity_offset
```

```
## [1]  33612  67208 100804 134400 167996 201592 235188 268784 302380
```

```r
filename <- "../../Homework/HW3/Example_Continuous.ibd"
intensity <- map2(intensity_offset, intensity_length,
                  function(offset, length) {
                    f <- file(filename, "rb")
                    seek(f, offset)
                    iout <- readBin(f, "double", n=length, size=4)
                    close(f)
                    iout
                  })
f <- file(filename, "rb")
seek(f, mz_offset)
```

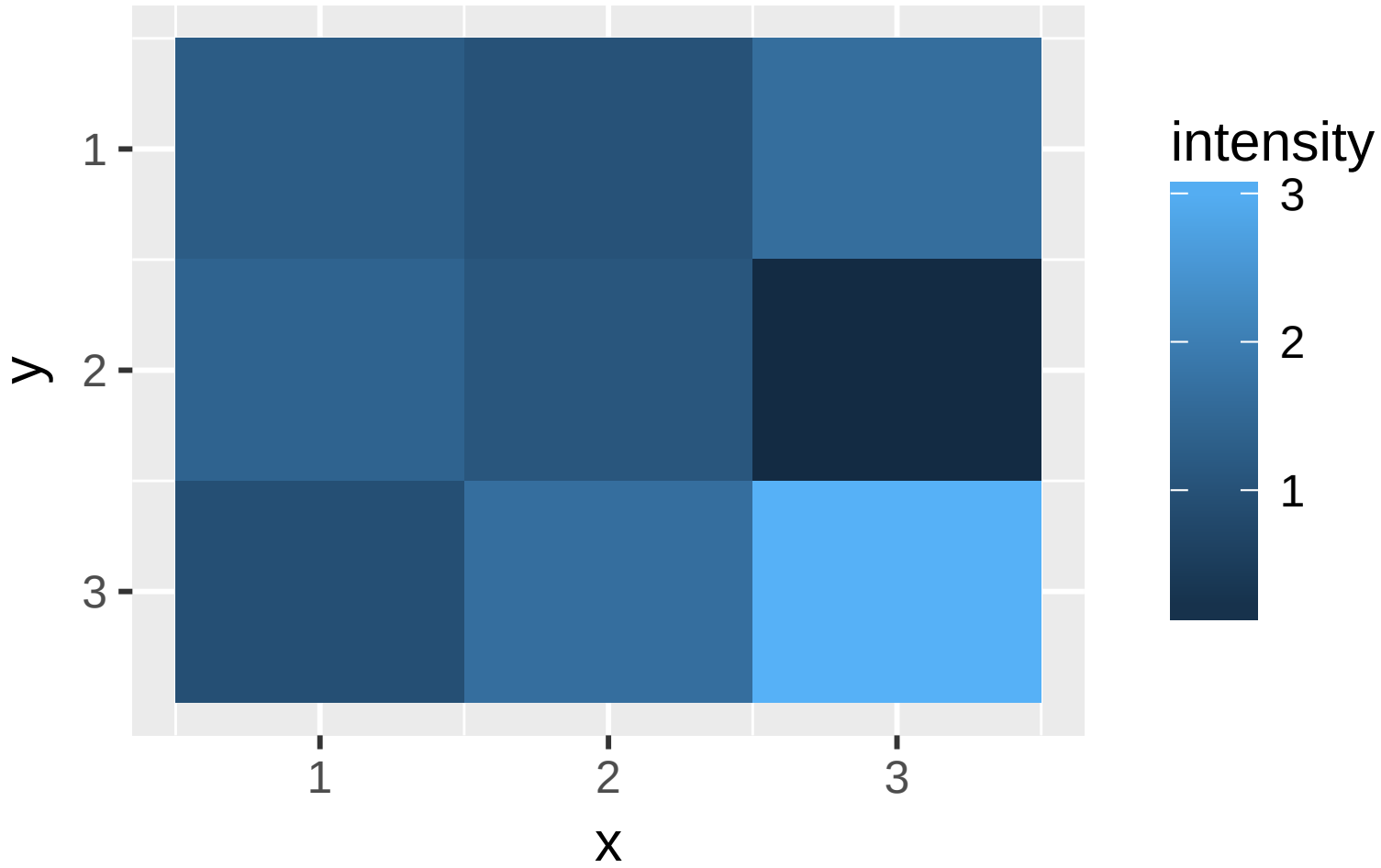
```
## [1] 0
```

```r
mz <- readBin(f, "double", n=mz_length, size=4)
close(f)
```

```
msi <- structure(list(mz=mz,
                      intensity=simplify2array(intensity),
                      coord=tibble(x=coord_x, y=coord_y)),
                 class="msi")
```

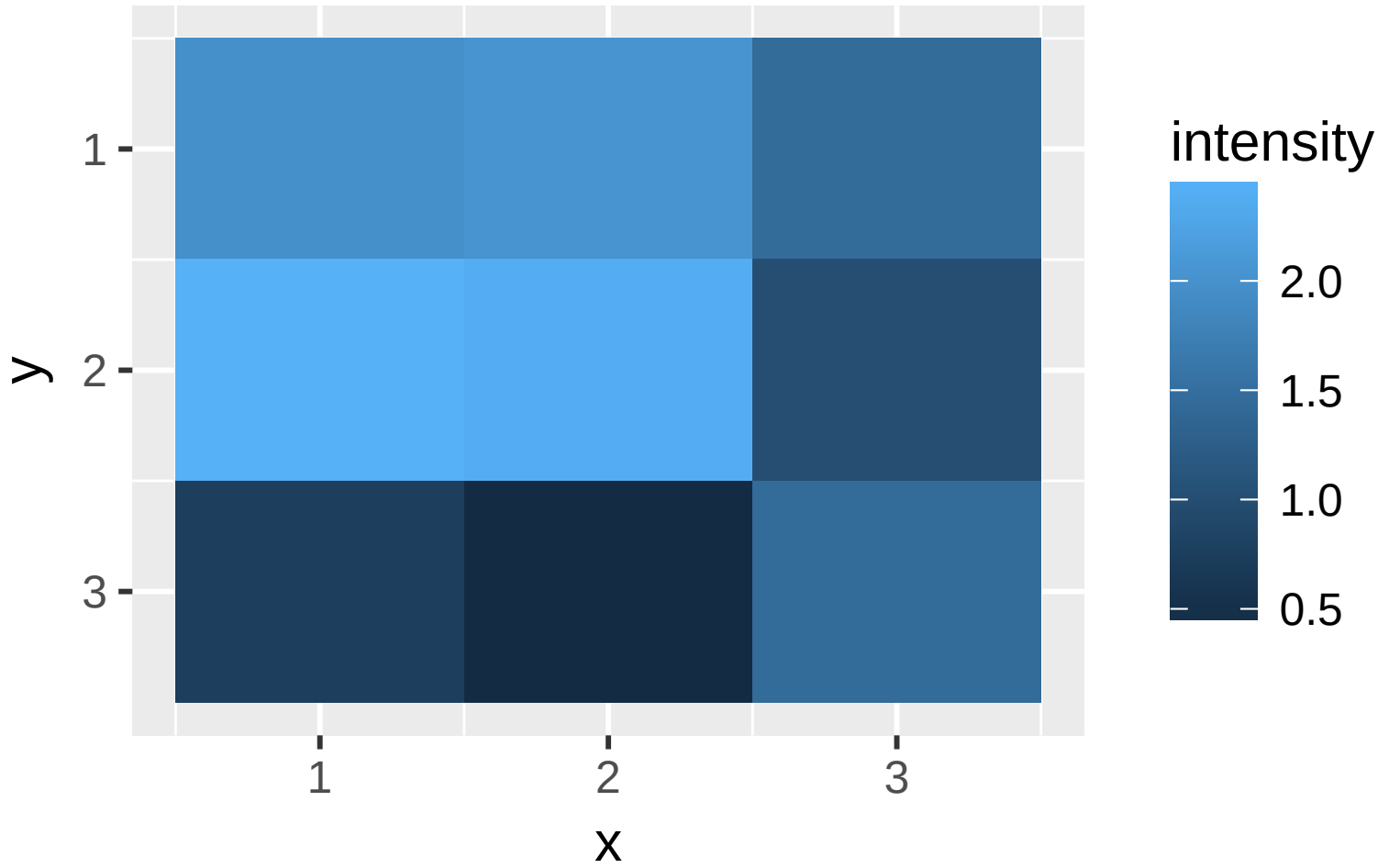
```
plot.msi <- function(x, mz) {
  idx <- which.min(abs(mz - x$mz))
  idf <- x$coord
  idf$intensity <- x$intensity[idx,]
  ggplot(idf) +
    geom_tile(aes(x=x, y=y, fill=intensity)) +
    scale_y_reverse()
}
```

```
plot(msi, mz=151.9)
```



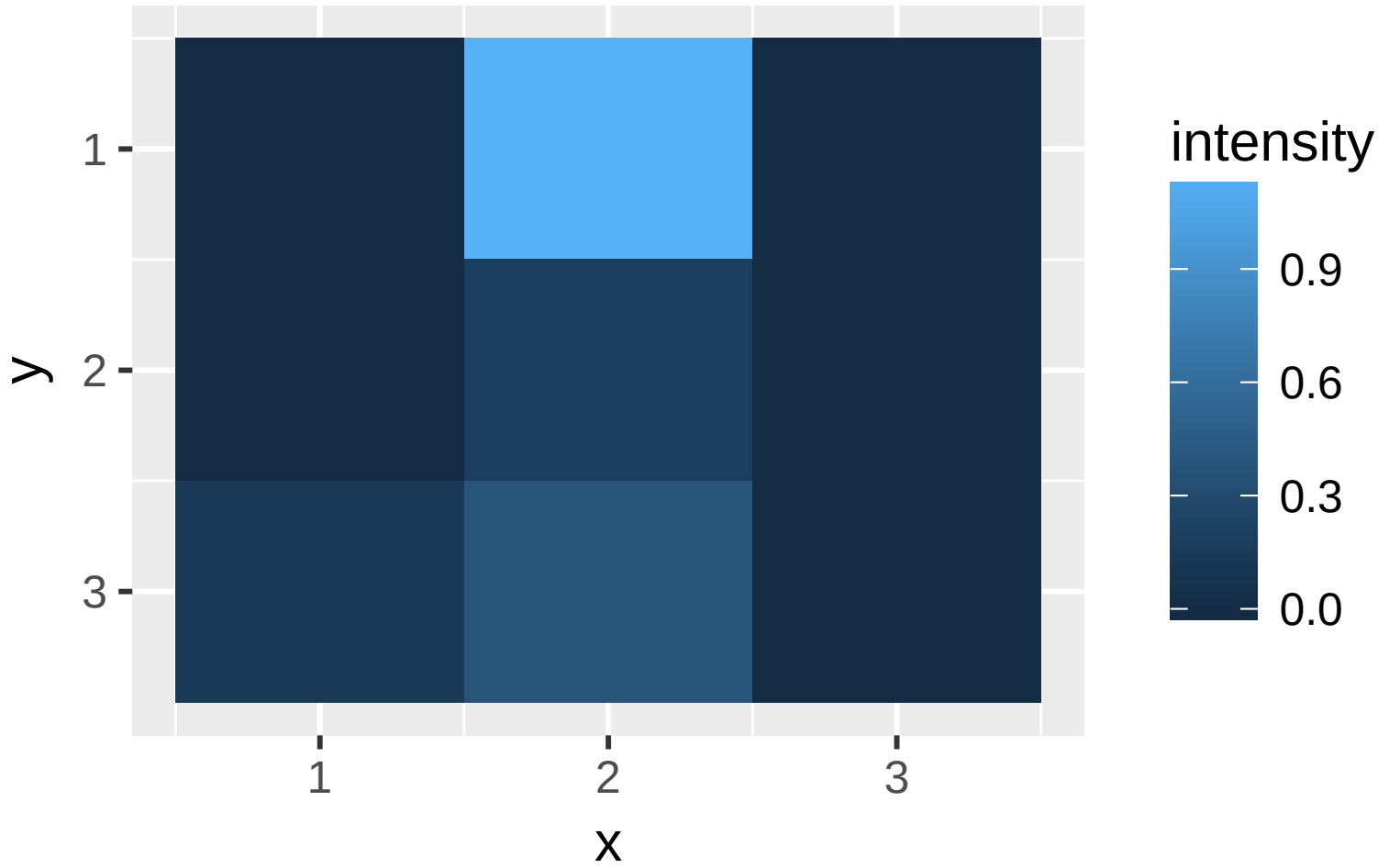
Note that these will not look exactly the same as in https://ms-imaging.org/wp/imzml/example-files-test/ because we did not bother to do averaging over +/- 0.25.

```
plot(msi, mz=328.9)
```



Note that these will not look exactly the same as in https://ms-imaging.org/wp/imzml/example-files-test/ because we did not bother to do averaging over +/- 0.25.

```
plot(msi, mz=185.2)
```



Note that these will not look exactly the same as in https://ms-imaging.org/wp/imzml/example-files-test/ because we did not bother to do averaging over +/- 0.25.