Estimating Williams College Faculty Age

Zafer Cesur ('19)

2016-02-03

Abstract This paper describes the R package **ager**. The package is focused on gathering the graduation data of Williams College faculty in order to construct statistical models for faculty age. The paper outlines the functions for selection and organization of the data, which are designed to require little to no manual manipulation during pre-processing and transformation thereof and be as applicable as possible so that it can easily be used for future years. The usage of these functions is illustrated and the results are demonstrated with descriptive histograms.

Introduction:

The package **ager** provides the functionality for easily accessing educational background of Williams College faculty members that consist of professors, lecturers and postdoctoral fellows. This is useful because assuming that most people graduate when they are 22 years old, we can estimate faculty's age with a small margin of error.

The data on faculty members' academic overview is available in the course catalogs that can be accessed online on Williams' website. Let us explore and compare some excerpts taken from these catalogs. For instance, this is how it looked in 2009-2010:

> Colin C. Adams, Thomas T. Read Professor of Mathematics
> B.S. (1978) M.I.T.; Ph.D. (1983) University of Wisconsin

And this in 2013-2014:

> Colin C. Adams, Thomas T. Read Professor of Mathematics, 1978, BS, MA Institute of Technology, 1983, PHD, University of WI, Madison

And finally, in 2015-2016:

> Adams,Colin C., Mathematics and Statistics

As we can observe, the format has significantly changed over the years. Also notice that in the most recent year, we do not get informed about the educational background of the professor, but rather only on the faculty he belongs to. This also holds true for the previous year, 2014-2015. The lack of information and the unpredictability of the presentation format urged us to devise a standard structure so that the data can be represented in a concise way and to develop an indirect method so that the desired information can be extracted.

The rest of the paper is organized as follows. First we are going to explain how we structured the data. Then, we are going to introduce the method we used to extract missing data. The last section concludes the article by presenting histograms and plots.

Methods

This section presents the methodology used in the package **ager**.

1. Selection and preparation

The catalog files are downloaded from the Williams' website in PDF format. Then, using A-PDF Text Extractor, the selected pages are converted into TXT files. The desired pages vary from year to year based on the content of the catalog; for the most academic year 2015-2016 we need pages from 396 through 398. Here is the workflow that we used to prepare the data we need:

1. Run A-PDF Text Extractor
2. Open bulletin2015_16.pdf
3. In the options tab, select pages from 396 to 398 and make sure *In PDF Order* is chosen as the method for extraction.
4. Extract text
5. Edit the lines that end prematurely, by hand. There were 2 manual editings for this document:

- Gibson,Matthew, Economics Department
- Das,Joanna Dee, Dance Department

The workflow for other years is exactly the same.

2. Parsing the data

The next step is cleaning and tidying up the data. In this section I will introduce the *scrub*, *gather_exactly* and *gather_reformated* functions.

2.1 scrub

```
library(magrittr)
library(devtools)
devtools::load_all()
```

```
## Loading ager
```

```
  # Open up a txt file and store lines of information as character strings of the
  # vector 'data'
  data <- paste("2015-16", ".txt", sep = "") %>%
    system.file("extdata", ., package = "ager") %>%
    readLines(skipNul = TRUE)
    summary(data)
```

```
##    Length     Class      Mode
##       936 character character
```

```
    head(data, 12)
```

```
##  [1] "ÿþ"                    ""                      ""
##  [4] ""                      "  "                    ""
##  [7] "FACULTY 2015-2016  " "" "                    "  "
## [10] ""                      ""                      ""
```

As we can see, there is a myriad of unwanted lines in the beginning of the document. We also have to get rid of empty lines inbetween and of the elements containing page numbers. We can remove these with the following set of codes.

```
# Remove leftovers from the previous page. In this case, we do not have any, however
# there are, for instance in the catalog from the year 2010-2011, some.
data <- data[-(1:(grep("FACULTY", data)[1]))]

# Remove all other irrelevant data at the beginning of the document until coming
# across a line containing a comma
data <- data[-(1:(grep(",", data)[1] - 1))]

# Remove the strings which are of length 0 and those containing page numbers. 4 is not
# chosen arbitrarily but rather based on other years' documents. For instance in the
# previous 2 years, the strings that contain pages are formatted as 3-digit integer and
# a space.
data <- setdiff(data, data[nchar(data) <= 4])
head(data)
```

```
## [1] "Aalberts,Daniel P., Physics Department   "
## [2] "+Acha,Beverly D., Art Department   "
## [3] "Adams,Colin C., Mathematics and Statistics   "
## [4] "Adhami,Zaid, Religion Department   "
## [5] "Adler Mandelbaum,Sara E., Economics Department   "
## [6] "Albrecht,Jeannie R, Computer Science Department   "
```

After the basic structure is established, we need to make sure no character string is stored as 2 separate ones. This can be caused while converting PDF documents if soft line breaks are converted to hard line breaks. For this document we do not need to worry about it as the lines are already very short. But for the year 2013-2014, for example, we need to run the following as the last step of cleaning process to make sure it flows smoothly.

```
short_lines <- order(nchar(data), decreasing = FALSE)[1:15]
long_lines <- order(nchar(data), decreasing = TRUE)[1:15]

# Evaluate which of the long lines are immediately followed by a short line
cut_lines <- short_lines[(short_lines - 1) %in% long_lines]

# Join the cut off line with the line above
data[cut_lines - 1] <- paste(data[cut_lines - 1], data[cut_lines], sep=" ")

# Remove cut off lines
data <- setdiff(data, data[cut_lines])
```

Of course, this step requires an assumption and an arbitrary integer to be chosen and there might be a better algorithm or program suited for this, but I have observed that it is precise enough for our purposes.

This section can be reproduced simply with the *scrub* function.

```
data <- scrub("2015-16")
head(data)
```

```
## [1] "Aalberts,Daniel P., Physics Department   "
```

```
## [2] "+Acha,Beverly D., Art Department   "
## [3] "Adams,Colin C., Mathematics and Statistics   "
## [4] "Adhami,Zaid, Religion Department   "
## [5] "Adler Mandelbaum,Sara E., Economics Department   "
## [6] "Albrecht,Jeannie R, Computer Science Department   "
```

2.2. gather_reformatted / gather_exactly

For the third part, it is crucial that we extract the names of faculty members from the character vector we have just recently obtained. Because the names are in *last-first* format in the year 2015-2016, we are going to use the function *gather_reformatted*, but the other one operates on the same principle as well.

```
  # Obtain first names using regular expressions
  first_name <-
    ",(| )(.*?)," %>%
    regexpr(data) %>%
    regmatches(data, .) %>%
    gsub(",(| )|,", "", .)
  head(first_name)
```

```
## [1] "Daniel P."  "Beverly D." "Colin C."   "Zaid"       "Sara E."
## [6] "Jeannie R"
```

It finds and returns the first match inbetween two commas. Although there is no space after the first comma in this document, (|) is added to the pattern in order not to lose generality.

```
    # Similarly,
  last_name <-
    "[A-Z](.*?)," %>%
    regexpr(data) %>%
    regmatches(data, .) %>%
    gsub(",.*", "", .)
  head(last_name)
```

```
## [1] "Aalberts"         "Acha"             "Adams"
## [4] "Adhami"           "Adler Mandelbaum" "Albrecht"
```

This one explicitly indicates that the match should start with a capital letter. This is in order to get rid of special characters that represent if a faculty member is visiting or on leave, and also to get rid of spaces that are present in some documents.

```
    names <- paste(first_name, last_name, sep = " ")
  head(names)
```

```
## [1] "Daniel P. Aalberts"       "Beverly D. Acha"
## [3] "Colin C. Adams"           "Zaid Adhami"
## [5] "Sara E. Adler Mandelbaum" "Jeannie R Albrecht"
```

This section can be reproduced with the function *gather_reformatted*. Let us see how the other related function works by directly implementing it.

```
  scrub("2013-14") %>%
  gather_exactly() %>%
    head()
```

```
## [1] "Daniel P. Aalberts"  "Sayaka Abe"          "Colin C. Adams"
## [4] "Jeannie R Albrecht"  "Laylah Ali"          "Kristopher D. Allen"
```

3 Extracting data

3.1 From another catalog

Now using the names we have gathered, we can obtain information from another document that has the desired content. For instance, so as to access the information we need for the analysis of the academic year 2015-16, we could extract data using the names of the faculty members who are common, from another source, for instance the catalog of the year 2013-14. We store the source data in another character vector called *data2*, but we do not need to gather names from this data source because it is unnecessary.

```
  data2 <- scrub("2013-14")
  head(data2)
```

```
## [1] "*Daniel P. Aalberts, Professor of Physics, 1989, BS, MA Institute of Technology, 1994, PHD, MA I
## [2] "Sayaka Abe, Visiting Assistant Professor of Japanese, 1997, BA, State University of NY, Buffalo
## [3] "Colin C. Adams, Thomas T. Read Professor of Mathematics, 1978, BS, MA Institute of Technology, I
## [4] "**Jeannie R Albrecht, Associate Professor of Computer Science, 2001, BS, Gettysburg College, 20(
## [5] "*Laylah Ali, Professor of Art, 1991, BA, Williams College, 1994, MFA, Washington University "
## [6] "Kristopher D. Allen, Clay Artist in Residence in Jazz Activities and Lecturer in Music, 1998, BI
```

```
  # Define an empty data frame to store the values
  faculty_df <- data.frame(matrix(vector(), length(names), 5,
    dimnames = list(c(), c("Name", "Graduation.Year", "Degree", "College", "Current.Age"))),
    stringsAsFactors = FALSE)

  # Locate the names from main data in the source data (i is arbitrarily chosen for
  # demonstrative purposes)
  i <- 50
  grep(names[i], data2)
```

```
## [1] 42
```

```
  # Find who it is
  data2[grep(names[i], data2)]
```

```
## [1] "Sandra L. Burton, Lipp Family Director of Dance and Senior Lecturer in Dance, 1983, BA, City Col
```

```
  # Locate relevant information
  pattern <- "\\d{4}, (B\\w+|AB), (.*?),"
  location <- regexpr(pattern, data2[grep(names[i], data2)])

  # Return the match
  education_info <- regmatches(data2[grep(names[i], data2)],
    regexpr(pattern, data2[grep(names[i], data2)])) %>%
  print
```

```
## [1] "1983, BA, City Coll of NY,"
```

```
  # Decompose it
  year <- gsub(",(.*)", "", education_info)
  degree <- gsub("\\d{4}, |,(.*)", "", education_info)
  college <- gsub("\\d{4}(.*), |,", "", education_info)
  print(c(year, degree, college))
```

```
## [1] "1983"            "BA"              "City Coll of NY"
```

Next, we need to add a couple of if-statements to deal with missing information:

```
  # If the faculty member is not on the source data, return NA
  if (length(grep(names[i], data2))==0){
      faculty_df[i, ] <- c(names[i], rep(NA, 4))
  }

  # If academic data is unavailable (some professors have only master's degree or Ph.D.
  # listed), return NA
  if (length(education_info) == 0){
      faculty_df[i, ] <- c(names[i], rep(NA, 4))
  }
```

Store the values in the data frame and define a for loop (not shown here).

```
  faculty_df[i, ] <- c(names[i],
                       year,
                       degree,
                       college,
                       2016-as.numeric(year)+22)
  print(faculty_df[i, ])
```

```
##                Name Graduation.Year Degree         College Current.Age
## 50 Sandra L. Burton            1983     BA City Coll of NY          55
```

Finally, convert degrees into the conventional degree format which has dots inbetween letters and columns into their respective, *actual* classes.

```
  by1 <- c("BA", "BS", "BM", "BFA", "BPhil", "AB", "BE")
  by2 <- c("B.A.", "B.S.", "B.M.", "B.F.A.", "B.Phil.", "B.A.", "B.E.")
  for (j in 1:7){
    faculty_df$Degree <- sub(by1[j], by2[j], faculty_df$Degree)
  }
  print(faculty_df[i, ])
```

```
##                Name Graduation.Year Degree         College Current.Age
## 50 Sandra L. Burton            1983   B.A. City Coll of NY          55
```

```
  faculty_df$Graduation.Year <- as.integer(faculty_df$Graduation.Year)
  faculty_df$Degree <- as.factor(faculty_df$Degree)
  faculty_df$Current.Age <- as.factor(faculty_df$Current.Age)
```

This section can be reproduced simply as follows.

```
faculty_df <- extract_info("2015-16", source_year = "2013-14",
              transformNames = TRUE, fillGaps = FALSE)
str(faculty_df)
```

```
## 'data.frame':    438 obs. of  5 variables:
##  $ Name           : chr  "Daniel P. Aalberts" "Beverly D. Acha" "Colin C. Adams" "Zaid Adhami" ...
##  $ Graduation.Year: int  1989 NA 1978 NA NA 2001 1991 NA NA NA ...
##  $ Degree         : Factor w/ 5 levels "B.A.","B.F.A.",..: 5 NA 5 NA NA 5 1 NA NA NA ...
##  $ College        : chr  "MA Institute of Technology" NA "MA Institute of Technology" NA ...
##  $ Current.Age    : Factor w/ 49 levels "28","30","31",..: 21 NA 32 NA NA 9 19 NA NA NA ...
```

In order to fill missing values, *fillGaps* argument of the function can be changed to TRUE. However this can also be done as a separate step as it takes some time to retrieve desired information from the internet.

3.2 From the internet

Using the names we have gathered in **2.2**, we can create search queries to investigate desired data further online by making use of Williams' directory search engine, which can generically accessed with the URL format http://www.williams.edu/people/?s_directory=firstname+lastname without middle names. We need the following set of codes.

```
# The carrot makes sure that we only get the first word of each name element.
first_name <- regmatches(names, regexpr("^\\w+", names))

# The optional \\w+[-] pattern makes sure that we take hypenated last names such as
# 'Robert Baker-White' into account. Such names cause no loss of generality in the
# generic search URL format given above.
last_name <- regmatches(names, regexpr("(\\w+[-])?\\w+$", names))

search_query <- paste(first_name, "+", last_name, sep = "")
search_link <- paste("http://www.williams.edu/people/?s_directory=", search_query,
               sep = "")
head(search_link)
```

```
## [1] "http://www.williams.edu/people/?s_directory=Daniel+Aalberts"
## [2] "http://www.williams.edu/people/?s_directory=Beverly+Acha"
## [3] "http://www.williams.edu/people/?s_directory=Colin+Adams"
## [4] "http://www.williams.edu/people/?s_directory=Zaid+Adhami"
## [5] "http://www.williams.edu/people/?s_directory=Sara+Mandelbaum"
## [6] "http://www.williams.edu/people/?s_directory=Jeannie+Albrecht"
```

Unfortunately, the search page only gives out limited information that consists of profession, department, office, phone number and e-mail address. It does not contain any links that we can follow, either. Therefore, we need to take an additional step and create a generic profile URL since we can extract Unix ID's from the e-mail address. We observed that the profile URL is in the format http://www.williams.edu/profile/unixid. Let us take Robert E. Baker-White, who is in the 18th row of our *names* vector, as an example.

```
library(rvest)
k <- 18
unix_id <-
```

```
    read_html(search_link[k]) %>%
    html_nodes(css = ".phone+ .email a") %>%
    html_attr("href") %>%
    gsub("mailto:|@williams.edu", "", .)
  print(unix_id)
```

```
## [1] "rbakerwh"
```

To select the CSS, we simply used SelectorGadget as Hadley Wickham also suggests in his package **rvest**. Next, we create profile URLs using the Unix IDs we have obtained and repeat the step above.

```
  profile_data <-
    paste("http://www.williams.edu/profile/", unix_id, sep="") %>%
    read_html() %>%
    html_nodes(css = ".profile-education .profile-subsection") %>%
    html_text()
  print(profile_data)
```

```
## [1] "B.A. Williams College (1980)M.F.A. University of Washington, Theatre (1983)Ph.D. Stanford Unive
```

Similarly, we extract the desired data as we did in **3.1**, but with a slightly different pattern.

```
  pattern <- "(B\\.(.*?)\\.|A\\.B\\.|Diploma) .*? \\(\\d{4}\\)"

  # Return the match
  education_info <- regexpr(pattern, profile_data) %>%
    regmatches(profile_data, .) %>%
    print
```

```
## [1] "B.A. Williams College (1980)"
```

```
  # Decompose it
  year <- gsub("(B\\.(.*?)\\.|A\\.B\\.|Diploma) .* \\(|\\)", "", education_info)
  degree <- gsub(" .*", "", education_info)
  college <- gsub("(B\\.(.*?)\\.|A\\.B\\.|Diploma) | \\(\\d{4}\\)", "", education_info)
  print(c(year, degree, college))
```

```
## [1] "1980"            "B.A."            "Williams College"
```

We also need an if-statement to deal with missing people, i.e., postdoctoral fellows and some visiting lecturers who do not appear on the directory search. We can do this by simply defining a variable named *email_node* and wrapping around our code with the following.

```
  email_node <-
    read_html(search_link[k]) %>%
    html_nodes(css = ".phone+ .email a")

  # If the person is not under the directory, we can safely assume the length of the variable
  # we have defined to be zero.
  if (length(email_node) != 0){
}
```

There are also two conditions that can stop the iteration. In order to handle such exceptions without breaking our for-loop and to see where they occur, we need to use the function **tryCatch**.

1. There may be multiple people with the same first and last name, in which case we obtain two different Unix IDs and they are both stored in the same vector *unix_id*. However, the function *read_html* takes only a single value and if a vector containing multiple URLs is provided, an error arises. Running the code with the *tryCatch* function around it shows us that there are two such cases: 'Matthew Gibson' and 'Joel Lee'. In each case, there is a professor and a student with the same first and last name. Since faculty members are always shown above students and other staff, we could command it to take the first value assuming it is unlikely that two professors share the same name. Just to be safe, we add an if-statement to check manually if our assumption is true and remove the *tryCatch* function because it is redundant. To demonstrate:

```
  names[132]
```

```
## [1] "Matthew Gibson"
```

```
  unix_id <-
    read_html(search_link[132]) %>%
    html_nodes(css = ".phone+ .email a") %>%
    html_attr("href") %>%
    gsub("mailto:|@williams.edu", "", .)
  if (length(unix_id) > 1){
    print(cat("Warning: More than one people with the name", print(names[132]), "\n"))
  }
```

```
## [1] "Matthew Gibson"
## Warning: More than one people with the name Matthew Gibson
## NULL
```

```
  return(unix_id[1])
```

```
## [1] "mg17"
```

2. There may be some faculty members who have web profile pages, but their undergraduate degree information is missing. With the exact method as above (i.e., by using *tryCatch*), we observe that there are 5 such cases:

- Nicole S. Desrosiers
- Wang Guowei
- Mamoru Hatakeyama
- Christophe A. Kone
- Julia L. Kowalski

```
  tryCatch({

  }, error=function(e){cat("Missing information on", print(names[i]), "\n")})
```

This section can be reproduced simply with the following function. Note that if we want to do this section separately rather than simply passing TRUE value to *fillGaps* argument of *extract_info*, we need to define the *names* variable because *extract_info* only returns a data frame.

```
  names <-
    scrub("2015-16") %>%
    gather_reformatted
  fill_gaps(faculty_df)
```

Results

For the sake of easy access to data frames, I have stored the data frames we need in the folder *inst/extdata* in advance. Let us load all our data and and combine them into a single data frame after adding another column that specifies the academic year.

```
  library(dplyr)
```

```
  faculty1516_df <- read.csv(system.file("extdata", "dataframe1516.csv", package = "ager"),
                    row.names = seq(along = 438),
                    stringsAsFactors = FALSE)
  faculty1516_df$Academic.Year <- "2015-2016"

  faculty1415_df <- read.csv(system.file("extdata", "dataframe1415.csv", package = "ager"),
                    row.names = seq(along = 421),
                    stringsAsFactors = FALSE)
  faculty1415_df$Academic.Year <- "2014-2015"

  faculty1314_df <- read.csv(system.file("extdata", "dataframe1314.csv", package = "ager"),
                    row.names = seq(along = 418),
                    stringsAsFactors = FALSE)
  faculty1314_df$Academic.Year <- "2013-2014"
  facultyoverview_df <- rbind(faculty1516_df, faculty1415_df, faculty1314_df)

  # Split the faculty into age groups
  facultyoverview_df <-
    facultyoverview_df %>%
    mutate(Age.Group = cut(Current.Age, breaks = seq(20, 80, by = 10)))

  # Factorize some of the classes
  facultyoverview_df$Degree <- as.factor(facultyoverview_df$Degree)
  facultyoverview_df$Current.Age <- as.factor(facultyoverview_df$Current.Age)
  facultyoverview_df$Academic.Year <- as.factor(facultyoverview_df$Academic.Year)

  # Get rid of missing values
  facultyoverview_df <- na.omit(facultyoverview_df)
  str(facultyoverview_df)
```

```
## 'data.frame':    1044 obs. of  7 variables:
##  $ Name           : chr  "Daniel P. Aalberts" "Colin C. Adams" "Zaid Adhami" "Sara E. Adler Mandelbau
##  $ Graduation.Year: int  1989 1978 2010 2009 2001 1991 2000 1998 1999 1999 ...
```

```
##  $ Degree         : Factor w/ 7 levels "B.A.","B.E.",..: 6 6 1 1 6 1 1 4 6 1 ...
##  $ College        : chr  "MA Institute of Technology" "MA Institute of Technology" "Stanford Univers
##  $ Current.Age    : Factor w/ 51 levels "27","28","29",..: 23 34 2 3 11 21 12 14 13 13 ...
##  $ Academic.Year  : Factor w/ 3 levels "2013-2014","2014-2015",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ Age.Group      : Factor w/ 6 levels "(20,30]","(30,40]",..: 3 4 1 1 2 3 2 2 2 2 ...
##  - attr(*, "na.action")=Class 'omit'  Named int [1:233] 2 10 11 35 36 44 66 91 124 148 ...
##   .. ..- attr(*, "names")= chr [1:233] "2" "10" "11" "35" ...
```

First let us take a look at the statistical summary of the faculty from the current year to get a general understanding as to how the data looks like.

```
  # Mean
  round(mean(as.numeric(faculty1516_df$Current.Age), na.rm = TRUE))
```

```
## [1] 51
```

```
  # Range
  range(as.numeric(faculty1516_df$Current.Age), na.rm = TRUE)
```

```
## [1] 28 80
```

```
  # Number of college faculty
  nrow(faculty1516_df)
```

```
## [1] 438
```

```
  # Youngest faculty member(s)
  faculty1516_df[which(faculty1516_df$Current.Age == 28), c(1,5)]
```

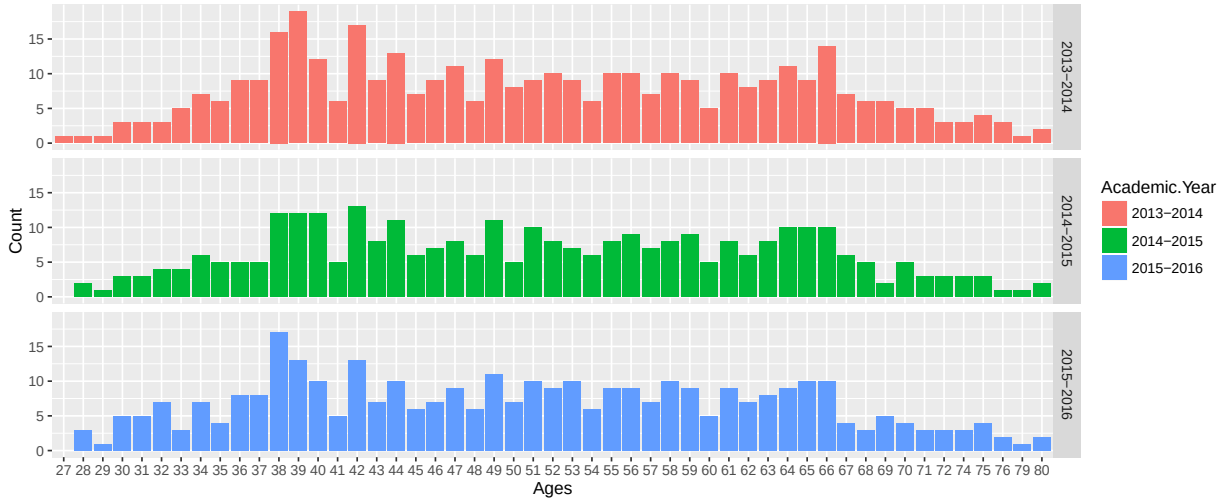
```
##                    Name Current.Age
## 4           Zaid Adhami          28
## 124 Rachel A. Friedman          28
## 174  Scott D. Honecker          28
## 236       Kelsey Levine          28
```

```
  # Oldest faculty member(s)
  faculty1516_df[which(faculty1516_df$Current.Age == 80), c(1,5)]
```

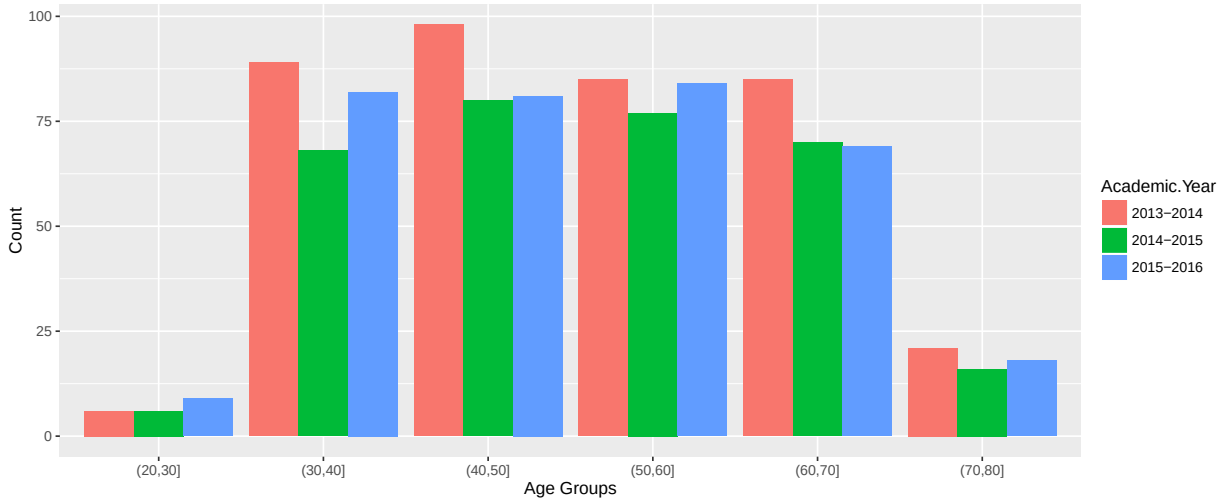
```
##                   Name Current.Age
## 28 Donald deB. Beaver          80
## 94     Charles B. Dew          80
```

Finally, here are some demonstrations that represent the data in different ways:

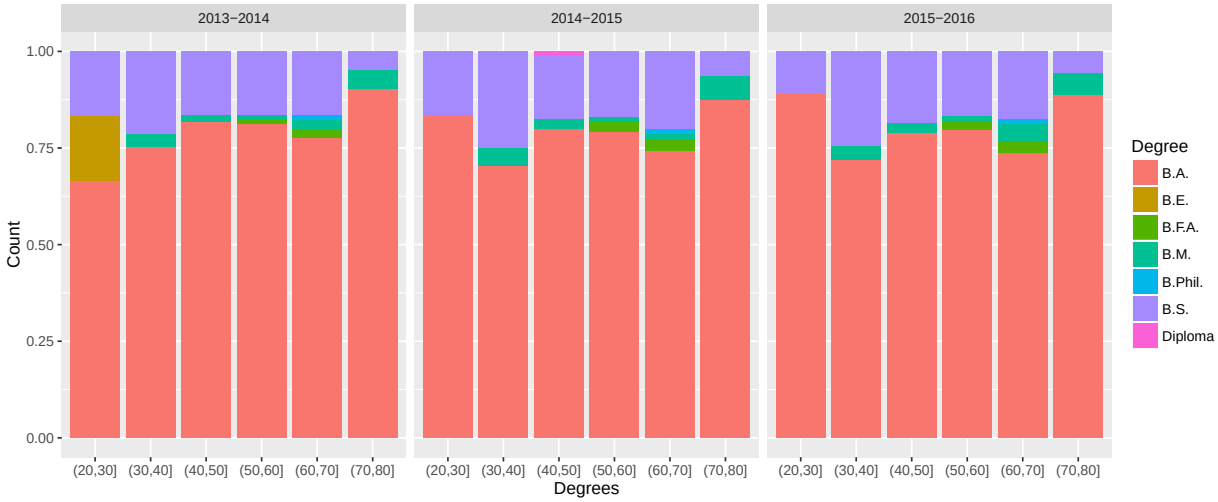
```
  library(ggplot2)
  library(reshape2)
  qplot(Current.Age,
        data = facultyoverview_df,
        facets = Academic.Year~.,
        fill = Academic.Year,
        xlab = "Ages",
        ylab = "Count")
```




```
ggplot(facultyoverview_df) +
    aes(x = Age.Group, fill = Academic.Year) +
    geom_bar(position = "dodge") + xlab("Age Groups") + ylab("Count")
```




```
ggplot(facultyoverview_df) +
    aes(x = Age.Group, fill = Degree) + facet_grid(. ~ Academic.Year) +
    geom_bar(position = "fill") + xlab("Degrees") + ylab("Count")
```


2013−2014
2014−2015
2015−2016
Count
1.00
0.75
0.50
0.25
0.00
(20,30]
(30,40]
(40,50]
(50,60]
(60,70]
(70,80]
Degrees
Degree
B.A.
B.E.
B.F.A.
B.M.
B.Phil.
B.S.
Diploma