Lecture 4

Data Preprocessing

Assoc. Prof. Dr. Burkay Genç

Mon Mar 11 16:09:34 2024

Packages used in these slides

```
library(knitr)
library(tidyr)
library(dplyr)
library(readr)
library(stringr)
library(lubridate)
library(Hmisc)
```

Seed used in these slides

```
set.seed(1024)
```

Data Preprocessing

Raw data -> Clean -> Transform -> Create -> Reduce

Tidy Data

Tidy Data

- Wickham (2014) -> Presented the notion of “tidy data”
- Most statistical datasets are data frames made up of *rows* and *columns*.
    - The columns are almost always labeled and the rows are sometimes labeled.

Data is **tidy**, if

1. Each variable is a column, each column is a variable
2. Each observation is a row, each row is an observation
3. Each value is a cell, each cell is a single value

This is basically Codd’s 3NF.

Common Problems in Data

- Real datasets *often* violate the three precepts of tidy data
- While occasionally you do get a dataset that you can start analysing immediately, this is the exception, not the rule.
- The following are the five most common problems seen with messy datasets:
  - Column headers are values, not variable names.
  - Multiple variables are stored in one column.
  - Variables are stored in both rows and columns.
  - Multiple types of observational units are stored in the same table.
  - A single observational unit is stored in multiple tables.

Tidy Data

Which table is *right*?

Name	Math	English
Anna	86	90
John	43	75
Catherine	80	82

Name	Anna	John	Catherine
Math	86	43	80
English	90	75	82

What about this one?

Name	Subject	Grade
Anna	Math	86
Anna	English	90
John	Math	43
John	English	75
Catherine	Math	80
Catherine	English	82

Column headers are values not variable names

Load Table Data

`stud.txt`

```
          Math   English
Anna        86        90
John        43        75
Catherine   80        82
```

```
std <- read_table("stud.txt", skip = 1,
                  col_names = c("StudentName", "Math", "English"))
```

```
##
## ── Column specification ────────────────────────────────────────────────────────
## cols(
##   StudentName = col_character(),
##   Math = col_double(),
##   English = col_double()
## )
```

```
std
```

```
## # A tibble: 3 × 3
##   StudentName  Math English
##   <chr>       <dbl>   <dbl>
## 1 Anna           86      90
## 2 John           43      75
## 3 Catherine      80      82
```

gather()

- `gather` is used to bring together multiple columns of data
  - it produces two columns
  - observations are distributed to multiple rows
- `gather` is **obsolete** now and no longer being developed

```
std
```

```
## # A tibble: 3 × 3
##   StudentName  Math English
##   <chr>       <dbl>   <dbl>
## 1 Anna           86      90
## 2 John           43      75
## 3 Catherine      80      82
```

```
stdL <- gather(std,
               Subject, Grade,
               Math:English)
stdL
```

```
## # A tibble: 6 × 3
##   StudentName Subject Grade
##   <chr>       <chr>   <dbl>
## 1 Anna        Math       86
## 2 John        Math       43
## 3 Catherine   Math       80
## 4 Anna        English    90
## 5 John        English    75
## 6 Catherine   English    82
```

pivot_longer()

- `pivot_longer` is a more modern approach to the same operation
- use `pivot_longer` in your new code

```
std
```

```
## # A tibble: 3 × 3
##   StudentName  Math English
##   <chr>       <dbl>   <dbl>
## 1 Anna           86      90
## 2 John           43      75
## 3 Catherine      80      82
```

```
stdL <- pivot_longer(std,
                     Math:English,
                     names_to = "Subject",
                     values_to = "Grade")
stdL
```

```
## # A tibble: 6 × 3
##   StudentName Subject Grade
##   <chr>       <chr>   <dbl>
## 1 Anna        Math       86
## 2 Anna        English    90
## 3 John        Math       43
## 4 John        English    75
## 5 Catherine   Math       80
## 6 Catherine   English    82
```

gather vs. pivot_longer

```
## # A tibble: 6 × 3
##   StudentName Subject Grade
##   <chr>       <chr>   <dbl>
## 1 Anna        Math       86
## 2 John        Math       43
## 3 Catherine   Math       80
## 4 Anna        English    90
## 5 John        English    75
## 6 Catherine   English    82
```

```
## # A tibble: 6 × 3
##   StudentName Subject Grade
##   <chr>       <chr>   <dbl>
## 1 Anna        Math       86
## 2 Anna        English    90
## 3 John        Math       43
## 4 John        English    75
## 5 Catherine   Math       80
## 6 Catherine   English    82
```

- `gather` produces column-wise output
- `pivot_longer` produces row-wise output
- it is trivial to turn one into the other
    - later

spread()

- `spread` is the opposite of `gather`

```
spread(stdL, Subject, Grade)
```

```
## # A tibble: 3 × 3
##   StudentName English  Math
##   <chr>         <dbl> <dbl>
## 1 Anna             90    86
## 2 Catherine        82    80
## 3 John             75    43
```

- Missing values will be filled with `NA`s

```
stdX <- stdL
stdX[3, 2] <- "French"
spread(stdX, Subject, Grade)
```

```
## # A tibble: 3 × 4
##   StudentName English French  Math
##   <chr>         <dbl>  <dbl> <dbl>
## 1 Anna             90     NA    86
## 2 Catherine        82     NA    80
## 3 John             75     43    NA
```

pivot_wider()

- `pivot_wider` is a more modern approach to the same operation
- use `pivot_wider` in your new code

```
stdL %>% pivot_wider(names_from = "Subject",
                     values_from = "Grade")
```

```
## # A tibble: 3 × 3
##   StudentName  Math English
##   <chr>       <dbl>   <dbl>
## 1 Anna           86      90
## 2 John           43      75
## 3 Catherine      80      82
```

- Missing values will be filled with `NA`s

```
stdX %>% pivot_wider(names_from = "Subject",
                     values_from = "Grade")
```

```
## # A tibble: 3 × 4
##   StudentName  Math English French
##   <chr>       <dbl>   <dbl>  <dbl>
## 1 Anna           86      90     NA
## 2 John           NA      75     43
## 3 Catherine      80      82     NA
```

Multiple variables stored in one column

Load Data

`stud2.txt`

```
Math English Degree_Year
Anna 86 90 Bio_2014
John 43 75 Math_2013
Catherine 80 82 Bio_2012
```

```
std2 <- read_table("stud2.txt", skip = 1,
                   col_names = c("StudentName", "Math", "English", "Degree_Year"))
```

```
##
## ── Column specification ────────────────────────────────────────────────────────
## cols(
##   StudentName = col_character(),
##   Math = col_double(),
##   English = col_double(),
##   Degree_Year = col_character()
## )
```

```
std2
```

```
## # A tibble: 3 × 4
##   StudentName  Math English Degree_Year
##   <chr>       <dbl>   <dbl> <chr>
## 1 Anna           86      90 Bio_2014
## 2 John           43      75 Math_2013
## 3 Catherine      80      82 Bio_2012
```

separate()

- `separate` splits a character column into multiple columns

```
std2L <- pivot_longer(std2, Math:English,
                      names_to = "Subject", values_to = "Grade")
std2L <- separate(std2L, Degree_Year, c("Degree", "Year"))
std2L
```

```
## # A tibble: 6 × 5
##   StudentName Degree Year  Subject Grade
##   <chr>       <chr>  <chr> <chr>   <dbl>
## 1 Anna        Bio    2014  Math       86
## 2 Anna        Bio    2014  English    90
## 3 John        Math   2013  Math       43
## 4 John        Math   2013  English    75
## 5 Catherine   Bio    2012  Math       80
## 6 Catherine   Bio    2012  English    82
```

- parameter `sep` can be used to fine tune
- `unite()` reverses the effect of `separate()`

separate() piped version

```
std2 %>%
  gather("Subject", "Grade", Math:English) %>%
  separate(Degree_Year, c("Degree", "Year"))
```

```
## # A tibble: 6 × 5
##   StudentName Degree Year  Subject Grade
##   <chr>       <chr>  <chr> <chr>   <dbl>
## 1 Anna        Bio    2014  Math       86
## 2 John        Math   2013  Math       43
## 3 Catherine   Bio    2012  Math       80
## 4 Anna        Bio    2014  English    90
## 5 John        Math   2013  English    75
## 6 Catherine   Bio    2012  English    82
```

Dates

Dates

- The `lubridate` package provides useful functions for working with dates

```
ymd("20151021")
```

```
## [1] "2015-10-21"
```

```
ymd("2019/10/31")
```

```
## [1] "2019-10-31"
```

```
ymd("19.2.4")
```

```
## [1] "2019-02-04"
```

```
dmy("4 July 2018")
```

```
## [1] "2018-07-04"
```

```
dmy_hms("4 July 2018, 14:05:01", tz = "Europe/Istanbul")
```

```
## [1] "2018-07-04 14:05:01 +03"
```

Dates

- You can provide multiple dates with different formats

```
dates <- c(20120521, "2010-12-12", "2007/01/5", "2015-2-04",
           "Measured on 2014-12-6", "2013-7+ 25")
dates <- ymd(dates)
dates
```

```
## [1] "2012-05-21" "2010-12-12" "2007-01-05" "2015-02-04" "2014-12-06"
## [6] "2013-07-25"
```

Dates

You can extract components from date objects

```
myDate <- dmy_hms("31 October 2019, 14:05:01", tz = "Europe/Istanbul")
wday(myDate)
```

```
## [1] 5
```

```
wday(myDate, label = TRUE)
```

```
## [1] Prş
## Levels: Paz < Pzt < Sal < Çrş < Prş < Cum < Cts
```

```
wday(myDate, week_start = 1)
```

```
## [1] 4
```

```
month(myDate)
```

```
## [1] 10
```

```
hour(myDate)
```

```
## [1] 14
```

Dates

- you can convert time zones

```
myDate
```

```
## [1] "2019-10-31 14:05:01 +03"
```

```
with_tz(myDate, "Pacific/Auckland")
```

```
## [1] "2019-11-01 00:05:01 NZDT"
```

```
with_tz(myDate, "America/New_York")
```

```
## [1] "2019-10-31 07:05:01 EDT"
```

```
force_tz(myDate, "America/New_York")
```

```
## [1] "2019-10-31 14:05:01 EDT"
```

Dates

- timezones can be listed with `OlsonNames()`

```
head(OlsonNames(), 50)
```

```
##  [1] "Africa/Abidjan"       "Africa/Accra"         "Africa/Addis_Ababa"
##  [4] "Africa/Algiers"       "Africa/Asmara"        "Africa/Asmera"
##  [7] "Africa/Bamako"        "Africa/Bangui"        "Africa/Banjul"
## [10] "Africa/Bissau"        "Africa/Blantyre"      "Africa/Brazzaville"
## [13] "Africa/Bujumbura"     "Africa/Cairo"         "Africa/Casablanca"
## [16] "Africa/Ceuta"         "Africa/Conakry"       "Africa/Dakar"
## [19] "Africa/Dar_es_Salaam" "Africa/Djibouti"      "Africa/Douala"
## [22] "Africa/El_Aaiun"      "Africa/Freetown"      "Africa/Gaborone"
## [25] "Africa/Harare"        "Africa/Johannesburg"  "Africa/Juba"
## [28] "Africa/Kampala"       "Africa/Khartoum"      "Africa/Kigali"
## [31] "Africa/Kinshasa"      "Africa/Lagos"         "Africa/Libreville"
## [34] "Africa/Lome"          "Africa/Luanda"        "Africa/Lubumbashi"
## [37] "Africa/Lusaka"        "Africa/Malabo"        "Africa/Maputo"
## [40] "Africa/Maseru"        "Africa/Mbabane"       "Africa/Mogadishu"
## [43] "Africa/Monrovia"      "Africa/Nairobi"       "Africa/Ndjamena"
## [46] "Africa/Niamey"        "Africa/Nouakchott"    "Africa/Ouagadougou"
## [49] "Africa/Porto-Novo"    "Africa/Sao_Tome"
```

Preprocessing Strings

String Processing

- Package `stringr` provides useful functions
- Package `stringi` provides more complex functions

String Processing

```
uci.repo <- "https://archive.ics.uci.edu/ml/machine-learning-databases/"
dataset <- "audiology/audiology.standardized"
dataF <- str_c(uci.repo,dataset,".data")
namesF <- str_c(uci.repo,dataset,".names")

data <- read_csv(url(dataF), col_names=FALSE, na="?")
print(data, n = 5)
```

```
## # A tibble: 200 × 71
##   X1    X2     X3    X4    X5    X6    X7    X8    X9    X10   X11   X12   X13
##   <lgl> <chr>  <lgl> <chr> <chr> <chr> <lgl> <chr> <lgl> <lgl> <lgl> <lgl> <lgl>
## 1 FALSE mild   FALSE norm… norm… <NA>  TRUE  <NA>  FALSE FALSE FALSE FALSE FALSE
## 2 FALSE moder… FALSE norm… norm… <NA>  TRUE  <NA>  FALSE FALSE FALSE FALSE FALSE
## 3 TRUE  mild   TRUE  <NA>  abse… mild  TRUE  <NA>  FALSE FALSE FALSE FALSE FALSE
## 4 TRUE  mild   TRUE  <NA>  abse… mild  FALSE <NA>  FALSE FALSE FALSE FALSE FALSE
## 5 TRUE  mild   FALSE norm… norm… mild  TRUE  <NA>  FALSE FALSE FALSE FALSE FALSE
## # i 195 more rows
## # i 58 more variables: X14 <lgl>, X15 <lgl>, X16 <lgl>, X17 <lgl>, X18 <lgl>,
## #   X19 <lgl>, X20 <lgl>, X21 <lgl>, X22 <lgl>, X23 <lgl>, X24 <lgl>,
## #   X25 <lgl>, X26 <lgl>, X27 <lgl>, X28 <lgl>, X29 <lgl>, X30 <lgl>,
## #   X31 <lgl>, X32 <lgl>, X33 <lgl>, X34 <lgl>, X35 <lgl>, X36 <lgl>,
## #   X37 <lgl>, X38 <lgl>, X39 <lgl>, X40 <lgl>, X41 <lgl>, X42 <lgl>,
## #   X43 <lgl>, X44 <lgl>, X45 <lgl>, X46 <lgl>, X47 <lgl>, X48 <lgl>, …
```

String Processing

```
text <- read_lines(url(namesF))
text[1:3]
```

```
## [1] "WARNING: This database should be credited to the original owner whenever"
## [2] "         used for any publication whatsoever."
## [3] ""
```

```
namesF
```

```
## [1] "https://archive.ics.uci.edu/ml/machine-learning-
databases/audiology/audiology.standardized.names"
```

```
nms <- str_split_fixed(text[67:135], ":", n = 2)[,1]
nms[1:3]
```

```
## [1] "   age_gt_60"  "   air()"      "   airBoneGap"
```

```
nms <- str_trim(nms)
nms[1:3]
```

```
## [1] "age_gt_60"  "air()"      "airBoneGap"
```

String Processing

```
nms <- str_replace_all(nms, "\\(|\\)", "")
nms[1:3]
```

```
## [1] "age_gt_60"  "air"        "airBoneGap"
```

```
colnames(data)[1:69] <- nms
data[1:3, 1:8]
```

```
## # A tibble: 3 × 8
##   age_gt_60 air      airBoneGap ar_c   ar_u   bone  boneAbnormal bser
##   <lgl>     <chr>    <lgl>      <chr>  <chr>  <chr> <lgl>        <chr>
## 1 FALSE     mild     FALSE      normal normal <NA>  TRUE         <NA>
## 2 FALSE     moderate FALSE      normal normal <NA>  TRUE         <NA>
## 3 TRUE      mild     TRUE       <NA>   absent mild  TRUE         <NA>
```

Unknown Values

Unknown Values

```
dat <- data.frame(X = c("green", "blue", "green", "red"),
                  Y = c(56, "?", 100, -10))
dat
```

```
##       X   Y
## 1 green  56
## 2  blue   ?
## 3 green 100
## 4   red -10
```

```
typeof(dat$Y)
```

```
## [1] "character"
```

Column Y is designated as a character variable due to the missing value

Unknown Values

- `parse_integer` from `readr` converts to integer and takes care of the NA values

```
dat$Y <- parse_integer(dat$Y, na = "?")
typeof(dat$Y)
```

```
## [1] "integer"
```

```
dat
```

```
##       X   Y
## 1 green  56
## 2  blue  NA
## 3 green 100
## 4   red -10
```

- alternatively,

```
dat$Y[dat$Y == "?"] <- NA
dat$Y <- as.integer(dat$Y)
```

Unknown Values

- How to deal with unknown values?
  - **remove** any row containing an unknown value
  - **fill-in** (impute) the unknowns using some **common value** (typically using statistics of centrality)
  - **fill-in** the unknowns using the **most similar rows**
  - using more **sophisticated forms of filling-in** the unknowns

Transforming Variables

Different Scales

- *Standardization* creates a new variable with mean 0 and sd 1

$$Y = \frac{X - \bar{x}}{s_X}$$

- *Normalization* creates a new variable between 0 and 1

$$Y = \frac{X - min_X}{max_X - min_X}$$

- *Log scaling* basically computes the log

$$Y = log(X)$$

Standardization

- `dplyr` method `scale()` does these transformations

```
data(iris)
apply(iris[,1:4], 2, mean)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##     5.843333     3.057333     3.758000     1.199333
```

```
apply(iris[,1:4], 2, range)
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## [1,]          4.3         2.0          1.0         0.1
## [2,]          7.9         4.4          6.9         2.5
```

```
iris_scaled <- cbind(iris %>% select(-Species) %>% scale(),
                     iris %>% select(Species))
apply(iris_scaled[,1:4], 2, mean)
```

```
##  Sepal.Length   Sepal.Width  Petal.Length   Petal.Width
## -4.484318e-16  2.034094e-16 -2.895326e-17 -3.663049e-17
```

```
apply(iris_scaled[,1:4], 2, range)
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## [1,]    -1.863780   -2.425820    -1.562342   -1.442245
## [2,]     2.483699    3.080455     1.779869    1.706379
```

Normalization

```
rng <- apply(iris[,1:4], 2, range)
rng <- rbind(rng, rng[2,] - rng[1,])
rng
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## [1,]          4.3         2.0          1.0         0.1
## [2,]          7.9         4.4          6.9         2.5
## [3,]          3.6         2.4          5.9         2.4
```

```
iris_scaled <- cbind(
  iris %>% select(-Species) %>% scale(center = rng[1,], scale = rng[3,]),
  iris %>% select(Species))
apply(iris_scaled[,1:4], 2, range)
```

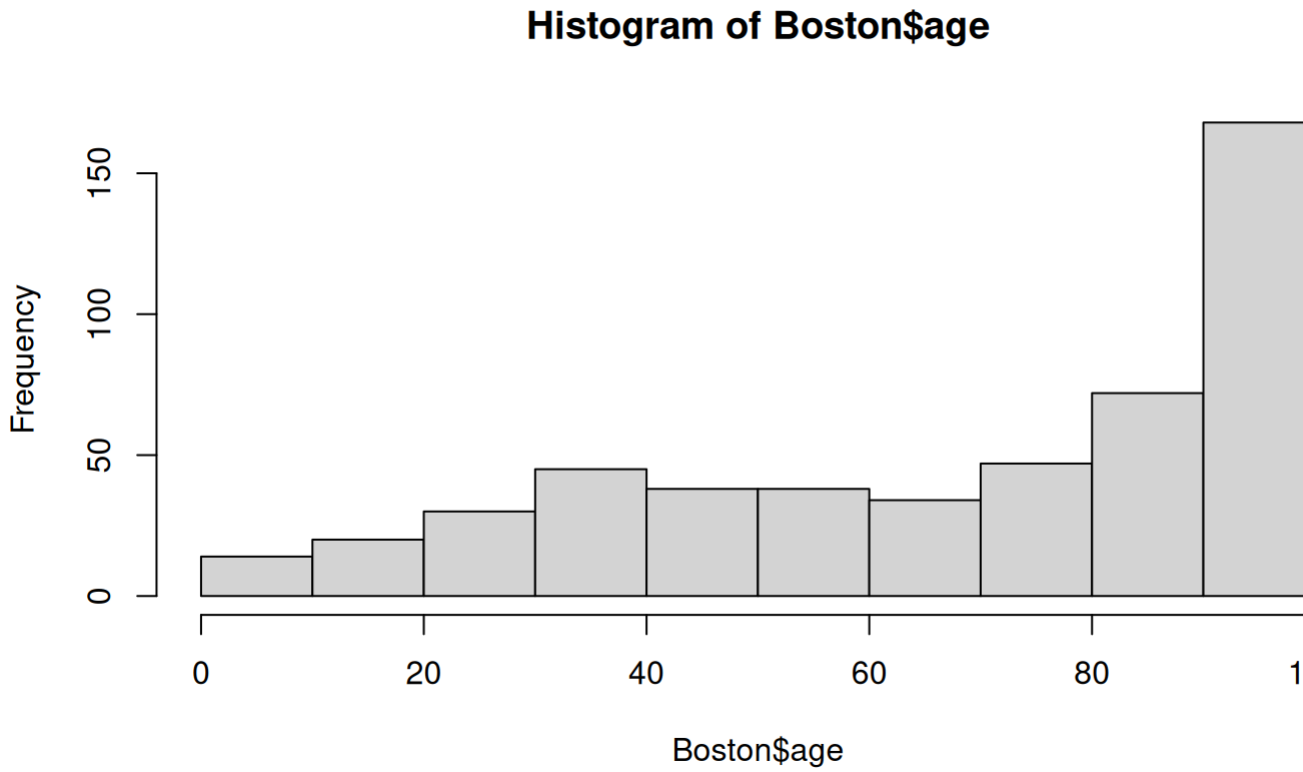
```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## [1,]            0           0            0           0
## [2,]            1           1            1           1
```

Discretization

- `cut2` from `Hmisc`
- `cut` from `base`
- Two main methods:
  - Equal width
  - Equal height

- Boston dataset from package `MASS`

```
data(Boston, package = "MASS")
hist(Boston$age)
```

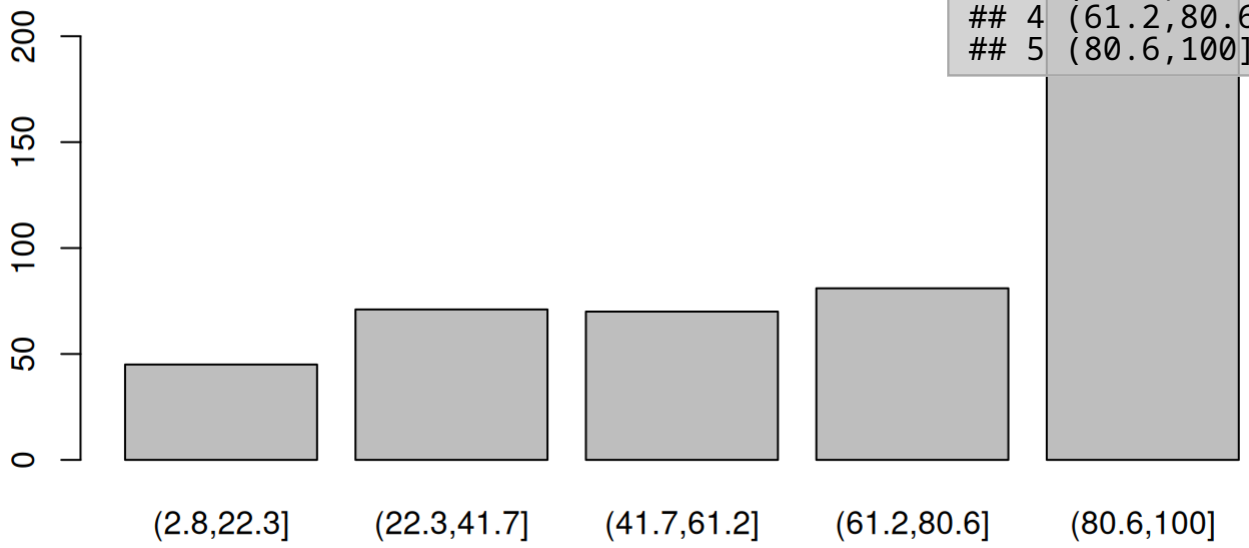


Discretization - Equal Width

```
Boston$newAge <- cut(Boston$age, 5)
barplot(table(Boston$newAge))
```

```
t <- table(Boston$newAge)
tibble(Range = names(t), Count = t)
```

```
## # A tibble: 5 × 2
##   Range       Count
##   <chr>       <table[1d]>
## 1 (2.8,22.3]   45
## 2 (22.3,41.7]  71
## 3 (41.7,61.2]  70
## 4 (61.2,80.6]  81
## 5 (80.6,100]  239
```



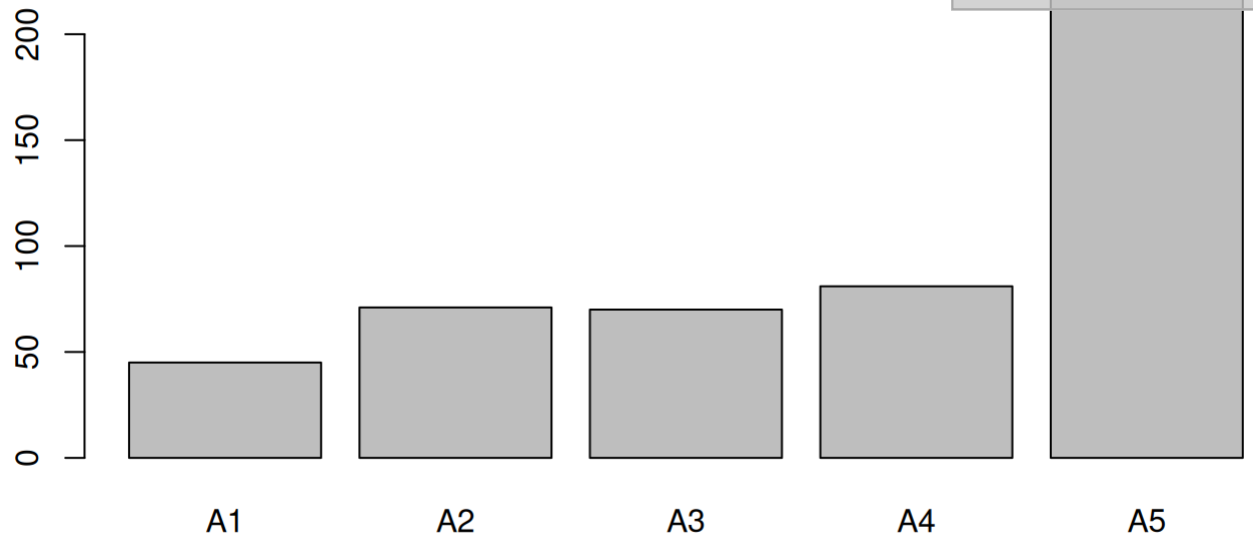
Discretization - Equal Width

- Provide label names

```
Boston$newAge <- cut(
  Boston$age, 5, labels =
    c("A1", "A2", "A3", "A4", "A5"))
barplot(table(Boston$newAge))
```

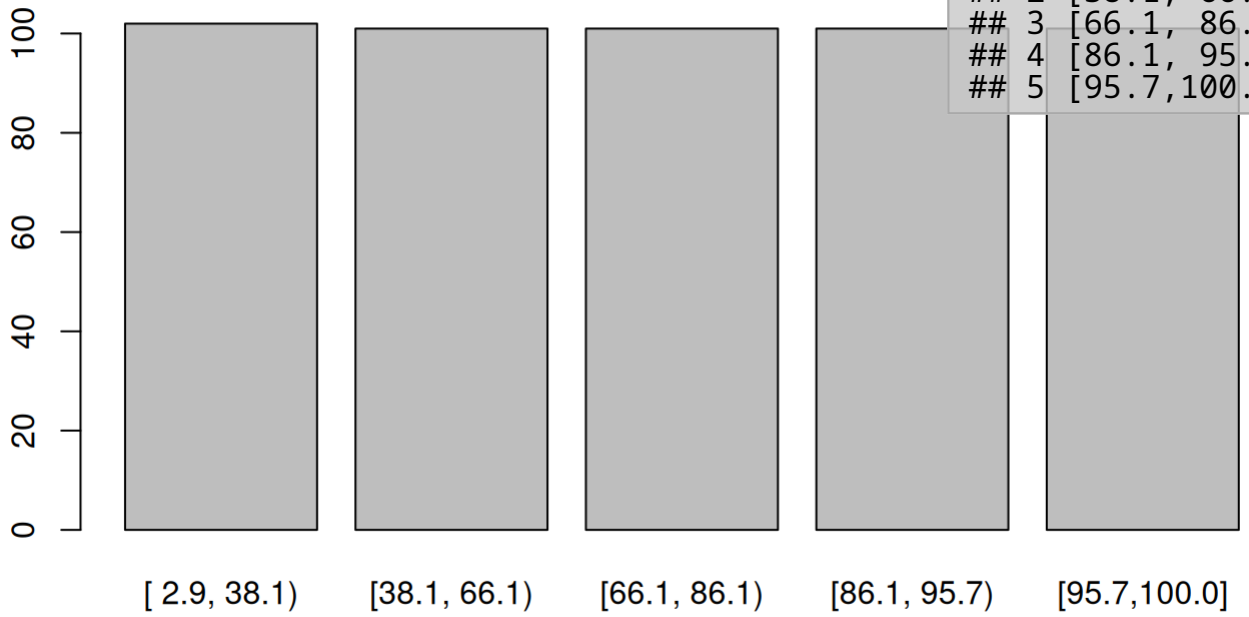
```
t <- table(Boston$newAge)
tibble(Range = names(t), Count = t)
```

```
## # A tibble: 5 × 2
##   Range Count
##   <chr> <table[1d]>
## 1 A1     45
## 2 A2     71
## 3 A3     70
## 4 A4     81
## 5 A5    239
```



Discretization - Equal Frequency

```
Boston$newAge <-
  cut2(Boston$age, g = 5)
barplot(table(Boston$newAge))
```




```
t <- table(Boston$newAge)
tibble(Range = names(t), Count = t)
```

```
## # A tibble: 5 × 2
##   Range         Count
##   <chr>         <table[1d]>
## 1 [ 2.9, 38.1) 102
## 2 [38.1, 66.1) 101
## 3 [66.1, 86.1) 101
## 4 [86.1, 95.7) 101
## 5 [95.7,100.0] 101
```

DIY Task

- Go to http://www.tuik.gov.tr
- Find “Tahıllar ve Diğer Bitkisel Ürünlerin Alan ve Üretim Miktarları” under “Temel İstatistikler”
- There are two tables in this Excel file. Load both as separate data tables (sown_area and production) into R
- Tidy both tables
- Using the `merge` function, merge the two tables into one
  - Table `merged` has 4 columns: Year, Type, Area, Tonnes
- Add column `efficiency` to `merged`
  - `efficiency` is the amount of production per decare
- Scale `efficiency` using *normalization*
- Discretize `efficiency` into 3 bins
  - low, medium, high
  - at breakpoints: (-0.01, 0.04, 0.10, 1.01)
- Display a table to show how many times each product type achieved each efficiency level

```
##
##                low medium high
##   Barley        18      0    0
##   Cotton         4     14    0
##   Maize          0     11    7
##   SugarBeets     0      0   18
##   Sunflower     18      0    0
##   Wheat         18      0    0
```