Case Study 1

Sonia Mazzi - Data Science Campus - ONS

18/09/2018

Data Science Campus

CASE STUDY. UC Berkeley gender bias post-graduate admissions data

Berkeley
UNIVERSITY OF CALIFORNIA

Data Science Campus

- Post-graduate admission figures for the auttumn of 1973 at the University of California, Berkeley.
- Men applying for post-graduate studies were more likely than women to be admitted.
- It was argued that the difference was so large that it was unlikely to be due to chance.

	Applicants	Admitted
Men	8442	44%
Women	4321	35%

- Data is presented in an informative way.
- However, this is an example of “messy data”.

	Applicants	Admitted
Men	8442	44%
Women	4321	35%

- An observational unit is an applicant.
- Measured variables are gender (male, female) and admission status (admitted, not admitted).
- Since we are not interested in any other chracteristics of the observational units we may aggregate cases by Gender and Admission Status.

Original, messy data is

	Applicants	Admitted
Men	8442	44%
Women	4321	35%

- Present the same information in a different and tidy way.

Gender	Admission_status	Number
male	admitted	3714
male	rejected	4728
female	admitted	1512
female	rejected	2809

	Applicants	Admitted
Men	8442	44%
Women	4321	35%

- The data above has been aggregated through departments.
- Different genders have different preferences of departments.
- Different departments have different admission policies.

- Graduate admissions by department (6 largest departments)

Department	Applied_men	Admitted_men	Applied_women	Admitted_women
A	825	62%	108	82%
B	560	63%	25	68%
C	325	37%	593	34%
D	417	33%	375	35%
E	191	28%	393	24%
F	373	6%	341	7%

- Create a tibble with the information above.

```
Berk_messy2 = read_table2("DataFiles/BerkeleyData.txt")
```

```
## Parsed with column specification:
## cols(
##   Department = col_character(),
##   Applied_men = col_integer(),
##   Admitted_men_p = col_character(),
##   Applied_women = col_integer(),
##   Admitted_women_p = col_character()
## )
```

```
Berk_messy2
```

```
## # A tibble: 6 x 5
##   Department Applied_men Admitted_men_p Applied_women Admitted_women_p
##   <chr>            <int> <chr>                  <int> <chr>
## 1 A                  825 62%                      108 82%
## 2 B                  560 63%                       25 68%
## 3 C                  325 37%                      593 34%
## 4 D                  417 33%                      375 35%
## 5 E                  191 28%                      393 24%
## 6 F                  373 6%                       341 7%
```

- The data in `Berk_messy2`, does not comply with the principles of tidy data.
- The variables in this study are Department (fixed), Gender (fixed), Number of applicants (measured) and Admission Status (measured).
- So, there is some manipulation to do before we can obtain a data array with 4 columns.

```
glimpse(Berk_messy2)
```

```
## Observations: 6
## Variables: 5
## $ Department       <chr> "A", "B", "C", "D", "E", "F"
## $ Applied_men      <int> 825, 560, 325, 417, 191, 373
## $ Admitted_men_p   <chr> "62%", "63%", "37%", "33%", "28%", "6%"
## $ Applied_women    <int> 108, 25, 593, 375, 393, 341
## $ Admitted_women_p <chr> "82%", "68%", "34%", "35%", "24%", "7%"
```

- Transform the information in percentages to admitted and non-admitted status numbers.
- The symbol "%" needs to be eliminated in order to be able to compute the number of admitted and non-admitted applicants.
- Use the functions `str_replace_all` and `sapply()` to apply the function to columns 3 and 5 in one call.

The function ‘str_replace_all()‘ in the package ‘stringr‘ replaces all instances of an existing pattern in a character vector, ‘pat_exist‘, with another, ‘pat_replace‘.
Usage: ‘str_replace_all(Vector, pat_exist, pat_replace)‘

The function ‘sapply()‘ applies a function to several columns in one call.
Usage: ‘sapply(Columns, Function, FunctionArgs)‘

```r
#remove percentage sign from columns 3 and 5
aux = sapply(Berk_messy2[,c(3,5)], str_replace_all, "%", "")

Berk_messy2[,c(3,5)] = aux

#make columns 3 and 5 numeric
Berk_messy2[,c(3,5)] = sapply(Berk_messy2[,c(3,5)], as.numeric)
```

```
Berk_messy2
```

```
## # A tibble: 6 x 5
##   Department Applied_men Admitted_men_p Applied_women Admitted_women_p
##   <chr>            <int>          <dbl>         <int>            <dbl>
## 1 A                  825             62           108               82
## 2 B                  560             63            25               68
## 3 C                  325             37           593               34
## 4 D                  417             33           375               35
## 5 E                  191             28           393               24
## 6 F                  373              6           341                7
```

- Convert percentages into admitted and not-admitted numbers.
- Want whole numbers, integers, so we use the function `round()` with the argument `digits=0`

The function ‘mutate()‘ of the ‘dplyr‘ package computes and appends one or more new columns to a tibble.
Usage: ‘mutate(theTibble, NewCol = ...)‘

```
Berk_messy2 =
mutate(Berk_messy2, Admitted_M = round(Admitted_men_p * Applied_men/100))
```

```
Berk_messy2 =
mutate(Berk_messy2, Admitted_F = round(Admitted_women_p * Applied_women/100))
```

```
Berk_messy2 = mutate(Berk_messy2, NotAdmitted_M = Applied_men - Admitted_M)
```

```
Berk_messy2 = mutate(Berk_messy2, NotAdmitted_F = Applied_women - Admitted_F)
```

```
glimpse(Berk_messy2)
```

```
## Observations: 6
## Variables: 9
## $ Department       <chr> "A", "B", "C", "D", "E", "F"
## $ Applied_men      <int> 825, 560, 325, 417, 191, 373
## $ Admitted_men_p   <dbl> 62, 63, 37, 33, 28, 6
## $ Applied_women    <int> 108, 25, 593, 375, 393, 341
## $ Admitted_women_p <dbl> 82, 68, 34, 35, 24, 7
## $ Admitted_M       <dbl> 512, 353, 120, 138, 53, 22
## $ Admitted_F       <dbl> 89, 17, 202, 131, 94, 24
## $ NotAdmitted_M    <dbl> 313, 207, 205, 279, 138, 351
## $ NotAdmitted_F    <dbl> 19, 8, 391, 244, 299, 317
```

```r
# remove the percentages columns
Berk_messy3 = select(Berk_messy2, -c(3,5))
```

```r
glimpse(Berk_messy3)
```

```
## Observations: 6
## Variables: 7
## $ Department    <chr> "A", "B", "C", "D", "E", "F"
## $ Applied_men   <int> 825, 560, 325, 417, 191, 373
## $ Applied_women <int> 108, 25, 593, 375, 393, 341
## $ Admitted_M    <dbl> 512, 353, 120, 138, 53, 22
## $ Admitted_F    <dbl> 89, 17, 202, 131, 94, 24
## $ NotAdmitted_M <dbl> 313, 207, 205, 279, 138, 351
## $ NotAdmitted_F <dbl> 19, 8, 391, 244, 299, 317
```

```r
# remove the total number of applications columns
Berk_messy3 = select(Berk_messy3, -c(2,3))
```

```r
glimpse(Berk_messy3)
```

```
## Observations: 6
## Variables: 5
## $ Department    <chr> "A", "B", "C", "D", "E", "F"
## $ Admitted_M    <dbl> 512, 353, 120, 138, 53, 22
## $ Admitted_F    <dbl> 89, 17, 202, 131, 94, 24
## $ NotAdmitted_M <dbl> 313, 207, 205, 279, 138, 351
## $ NotAdmitted_F <dbl> 19, 8, 391, 244, 299, 317
```

```
glimpse(Berk_messy3)
```

```
## Observations: 6
## Variables: 5
## $ Department    <chr> "A", "B", "C", "D", "E", "F"
## $ Admitted_M    <dbl> 512, 353, 120, 138, 53, 22
## $ Admitted_F    <dbl> 89, 17, 202, 131, 94, 24
## $ NotAdmitted_M <dbl> 313, 207, 205, 279, 138, 351
## $ NotAdmitted_F <dbl> 19, 8, 391, 244, 299, 317
```

- We have managed to tidy the data a bit, but we are not there yet.
- Recall that the variables in this study are: gender, department, admission status and numbers observed.
- We would like the data to be stored in four columns.
- As it is, the data combines two variables, gender and admission status in single columns.

Let us use the package `tidyr` to manipulate the data further.

The column names `Admitted_F`, `Admitted_M`, `NotAdmitted_F`, `NotAdmitted_M` are not variable names but values of two variables: admission status and gender.

- First create a column named `admitted_status_gender` with values `Admitted_F`, `Admitted_M`, `NotAdmitted_F`, `NotAdmitted_M`,
- and another column, `number`, with the resulting values for each category.

```
Berk_messy4 =
  gather(Berk_messy3, "admitted_status_gender", "number", -1)
```

```
glimpse(Berk_messy4)
```

```
## Observations: 24
## Variables: 3
## $ Department             <chr> "A", "B", "C", "D", "E", "F", "A", "B",...
## $ admitted_status_gender <chr> "Admitted_M", "Admitted_M", "Admitted_M...
## $ number                 <dbl> 512, 353, 120, 138, 53, 22, 89, 17, 202...
```

- Note that `admitted_status_gender` actually combines two variables: admission status and gender.

Data Science Campus

So we must separate this column into two. We use the `separate()` function of the `tidyr` package.

```
Berk_tidy2 = separate(Berk_messy4, admitted_status_gender,
                      c("AdmissionStatus","Gender"), sep = "_")
```

```
glimpse(Berk_tidy2)
```

```
## Observations: 24
## Variables: 4
## $ Department      <chr> "A", "B", "C", "D", "E", "F", "A", "B", "C", "...
## $ AdmissionStatus <chr> "Admitted", "Admitted", "Admitted", "Admitted"...
## $ Gender          <chr> "M", "M", "M", "M", "M", "M", "F", "F", "F", "...
## $ number          <dbl> 512, 353, 120, 138, 53, 22, 89, 17, 202, 131, ...
```

The function ‘separate()‘ of the ‘tidyr‘ package separates one column into several columns.
Usage: ‘separate(theTibble, colToSep, c("newcol1", ...))‘

Data Science Campus

Finally, we re-order columns and sort rows by the first column, Department, using the `arrange()` function.

```
# rearrange columns, fixed variables first
Berk_tidy2 = Berk_tidy2[,c(1,3,2,4)]
```

```
# order rows by Department
Berk_tidy2 = arrange(Berk_tidy2, Department)
```

```
Berk_tidy2
```

```
## # A tibble: 24 x 4
##    Department Gender AdmissionStatus number
##    <chr>      <chr>  <chr>             <dbl>
##  1 A          M      Admitted            512
##  2 A          F      Admitted             89
##  3 A          M      NotAdmitted         313
##  4 A          F      NotAdmitted          19
##  5 B          M      Admitted            353
##  6 B          F      Admitted             17
##  7 B          M      NotAdmitted         207
##  8 B          F      NotAdmitted           8
##  9 C          M      Admitted            120
## 10 C          F      Admitted            202
## # ... with 14 more rows
```

The data is now tidy and ready for exploratory analysis.

Data Science Campus

We visualise this data set of categorical, non-ordinal data using a mosaic plot. Mosaic plots are useful for visualizing proportions in more than 2 dimensions. We use the package `vcd` (not in the Tidyverse).

First, we display the data as a contingency table

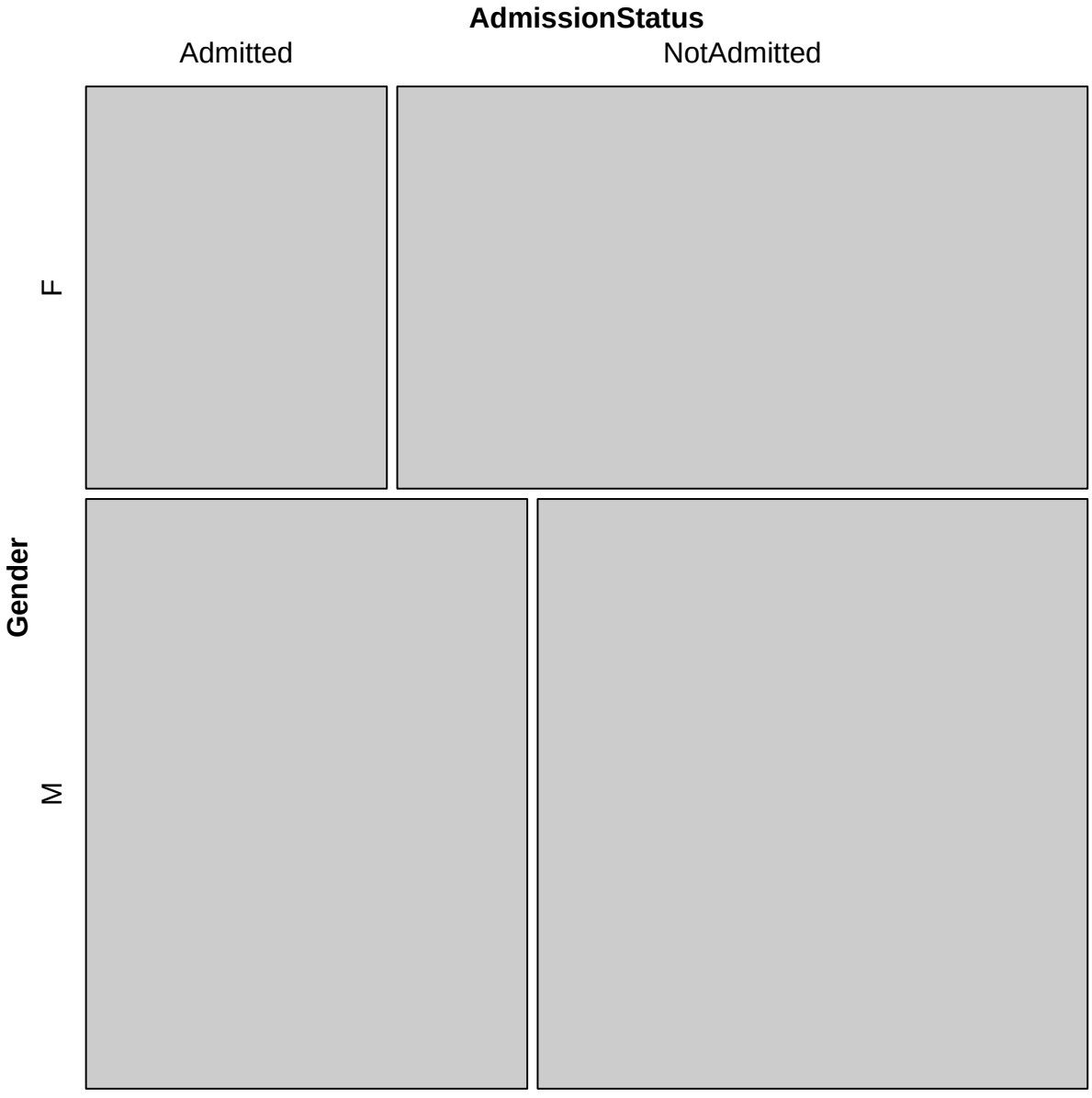
```
# create a contingency table
ct1 =
  xtabs(number ~ Gender + AdmissionStatus, data = Berk_tidy2)
```

```
ct1
```

```
##        AdmissionStatus
## Gender Admitted NotAdmitted
##      F      557         1278
##      M     1198         1493
```

Now we display it as a mosaic plot

```
mosaic(ct1)
```

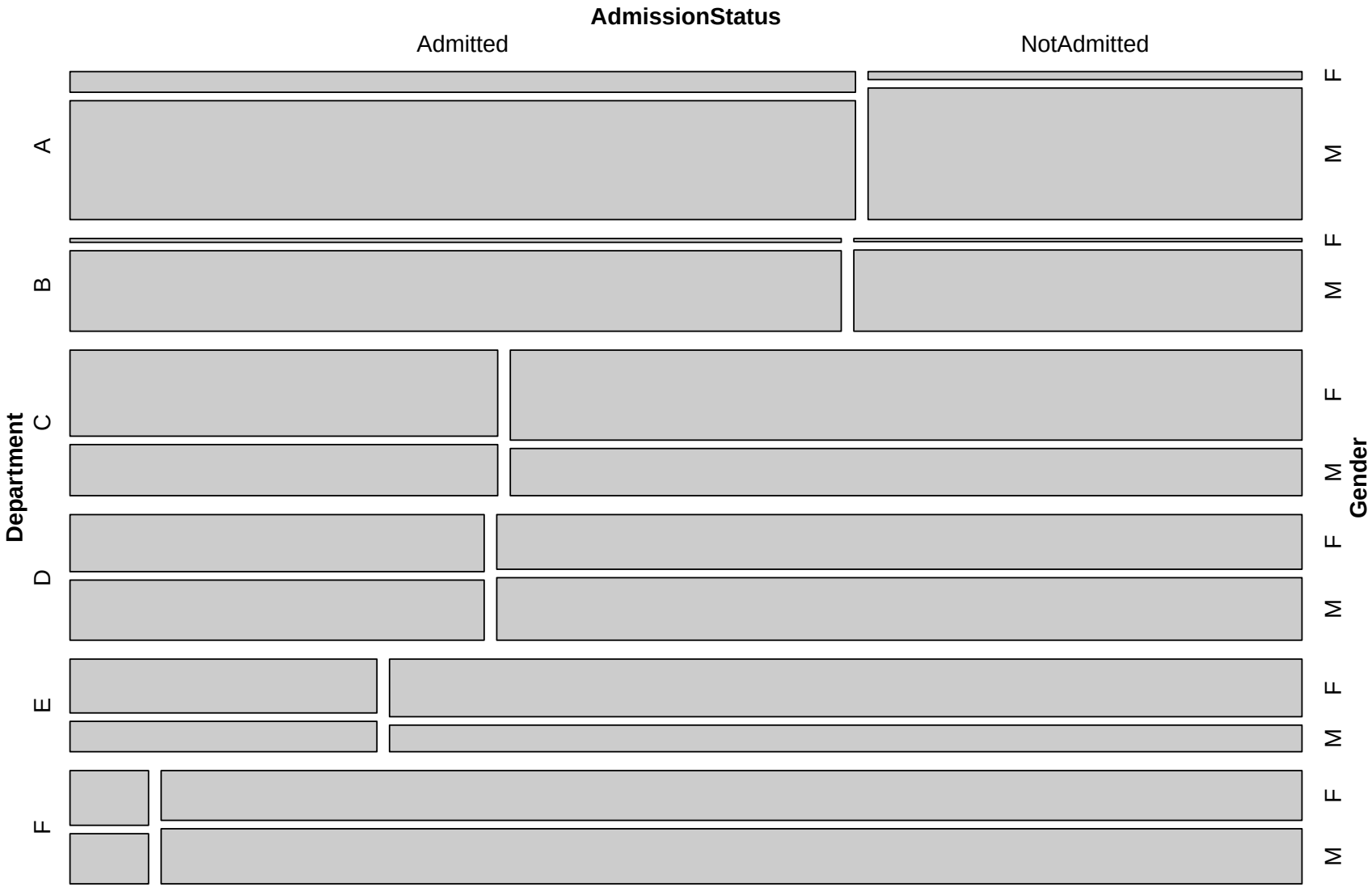


Data Science Campus

Next we visualise the data in a similar way but by department or academic unit.

```
# create a contingency table
aux2 =
  xtabs(number ~ Department + AdmissionStatus + Gender,
        data = Berk_tidy2)
```

```
mosaic(aux2)
```



- One of the perils when studying associations between a variable of interest and a set of explanatory variables is overfitting.
- Too many explanatory variables we may explain very well the observed values of the variable of interest but nothing else and so our study will have little predictive value.
- Problems also occur when relevant explanatory variables are ignored.
- It is possible that when one ignores a relevant variable one observes an effect and when the variable is considered the opposite effect is observed.
- This is called Simpson's paradox.
- What we have explored with the Berkeley garduate admissions data is one of the best-known examples of Simpson's paradox.