SUMMARY STATISTICS

Applied math for data science (DSC 482/MTH 445) Fall 2022

Table of Contents

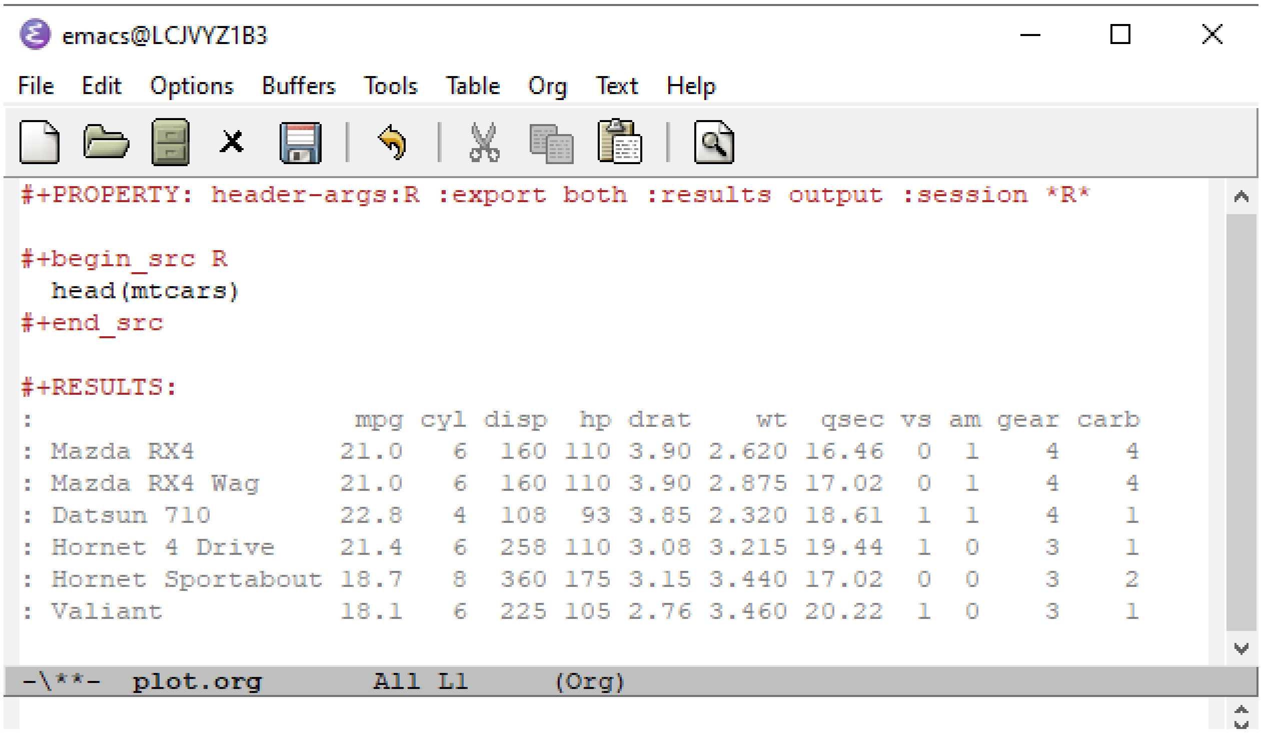
Figure 1: poster for "The Terminal List" (Amazon, 2022-).

- Centrality: Mean, Median, Mode
- Counts, Percentages, and Proportions
- Quantiles, Percentiles, and 5-number-summary
- Spread: Variance, Standard Deviation and Interquartile Range
- 2 Practice exercises and 1 exercise session

1 Measures of centrality

- Explain large collections of data
- Describe where numeric observations are centered
- Centrality measures suggest symmetries where none exist

2 Preparations to code along



```
#+PROPERTY: header-args:R :export both :results output :session *R*

#+begin_src R
  head(mtcars)
#+end_src

#+RESULTS:
:                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
: Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
: Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
: Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
: Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
: Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
: Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1

-\**-  plot.org       All L1      (Org)
```

- **Open** a new Org-mode file `stats.org` in Emacs
- **Add** this line at the top of the file `plot.org`:

  ```
  #+PROPERTY: header-args:R :results output :session *R*
  ```

- **Activate** the code by putting your cursor on the line and entering `C-c C-c`. You should see the message `Local setup has been refreshed` in the minibuffer at the bottom of the editor.
- When you execute your first R code block, you'll be asked where you want the session named `*R*` to run: **enter** the path to `plot.org`
- For plots, **use** the header `:results graphics file :file plot.png`
- When you leave Emacs, you'll be warned that the session `*R*` is active: you can **ignore** this warning

3 Mean or arithmetic average

- For a set of n labeled numeric measurements, the sample mean is the arithmetic average over all

  $$\bar{x} = \frac{(x_1 + x_2 + \ldots + x_n)}{n} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

  measurements:

- If you observe 8 points 2, 4.4, 3, 3, 2, 2.2, 2, 4, the mean is:

```
x <- c(2,4.4,3,3,2,2.2,2,4) # store observations in vector
x_mean <- sum(x)/length(x)  # arithmetic mean
x_mean  # print mean
```

4 Median or middle magnitude

- Sort your observations by magnitude
- For an odd number of observations: take the middle value
- For an even number of observations: average two middle values

$$\bar{m}_x = \begin{cases} x_i^{(\frac{n+1}{2})}, & \text{if } n \text{ is odd} \\ \left(x_i^{(\frac{n}{2})} + x_j^{(\frac{n}{2}+1)}\right)/2, & \text{if } n \text{ is even} \end{cases}$$

Where the upper index denotes the order statistics: $x_i^{(t)}$ is the t-th smallest observation regardless of the observation index i.

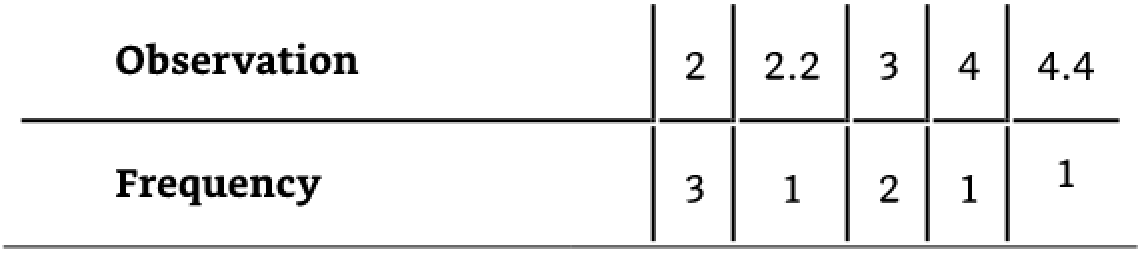
- If you observe 8 points 2, 4.4, 3, 3, 2, 2.2, 2, 4, you have n/2=4.

```
x <- c(2,4.4,3,3,2,2.2,2,4) # store observations in vector
sorted_x <- sort(x)  # sort observations
sorted_x
                                          # median by hand:
(sorted_x[length(x)/2] + sorted_x[length(x)/2+1])/2
```

5 Mode or most common observation

- Used with numeric-discrete data than numeric-continuous
- Used when discussing probability *density* functions
- Collection of numeric measurements may have no or > 1 mode

- If you observe 8 points 2, 4.4, 3, 3, 2, 2.2, 2, 4, you can tabulate the frequency of each measurement:



Observation	2	2.2	3	4	4.4
Frequency	3	1	2	1	1

6 Mean and median with built-in functions

- Create a new Org-mode file

- Create a R code block with header: `R :results output :session`
- Store the eight observations as a numeric vector `xdata`:

  {2, 4.4, 3, 3, 2, 2.2, 2, 4}

```
xdata <- c(2,4.4,3,3,2,2.2,2,4) # store observations in vector
xdata
```

- Compute the `mean` and the `median` and store them in variables.

```
x.bar <- mean(xdata)
x.bar
```

```
m.bar <- median(xdata)
m.bar
```

7 Mode with contingency `table`

- To find a mode, compute the contingency `table` for `xdata`.

```
xtab <- table(xdata)
xtab
```

- To identify the most frequent values automatically, use `range` which reports `min` and `max` of `xdata`.

```
sort(xdata) # sort vector values
min(xdata)
max(xdata)
range(xdata) # return min and max value
```

- Applying these functions to a `table` operates on the frequencies:

```
xtab # object that stores the contingency table
max(xtab) # max frequency in the table
```

- Finally, use a logical index vector to get the mode:

```
d.bar <- xtab[xtab == max(xtab)]
d.bar  # prints the value and the frequency
```

8 Practice: measures of centrality

1. Calculate the mean and median weights of the chicks in `chickwts`
2. Identify the value and frequency of the most common earthquake magnitude in `quakes`

8.1 SOLUTION

1. Calculate the mean and median weights of the chicks in `chickwts`

```
str(chickwts)
mean(chickwts$weight)
median(chickwts$weight)
```

```
'data.frame':	71 obs. of  2 variables:
 $ weight: num  179 160 136 227 217 168 108 124 143 140 ...
 $ feed  : Factor w/ 6 levels "casein","horsebean",..: 2 2 2 2 2 2 2 2 2 2 ...
[1] 261.3099
[1] 258
```

Answer: the average chick weight 261 grams, and the chicken in the middle weighed 258 grams.

2. Identify the value and frequency of the most common earthquake magnitude in `quakes`

```
str(quakes)
Qtab <- table(quakes$mag)
Qtab[Qtab == max(Qtab)]
```

```
'data.frame':	1000 obs. of  5 variables:
 $ lat     : num  -20.4 -20.6 -26 -18 -20.4 ...
 $ long    : num  182 181 184 182 182 ...
 $ depth   : int  562 650 42 626 649 195 82 194 211 622 ...
 $ mag     : num  4.8 4.2 5.4 4.1 4 4 4.8 4.4 4.7 4.3 ...
 $ stations: int  41 15 43 19 11 12 43 15 35 19 ...
4.5
107
```

Answer: the most common earthquakes were 107 occurrences of magnitude 4.5.

9 Missing or undefined values

- Many of the standard stats functions in R will not run if the data set contains missing (`NA`) or undefined (`NaN`) values.

```
mean(c(1,4,NA))
mean(c(1,4,NaN))
```

- To prevent inclusion of these special values, switch on the `na.rm` attribute.

```
mean( c(1,4,NA), na.rm = TRUE)
mean( c(1,4,NaN), na.rm = TRUE)
```

- Anything that calculates a numeric statistic based on a numeric vector carries this attribute: `sum`, `mean`, `median`, `max`, `min`, `range`.

10 Practice: missing values

1. Look at the `Pima.tr` dataset in the `MASS` package.
2. Use `summary` to find out how many values are missing in the measurements of the body mass index.
3. What is the sample mean of the body mass index values?

10.1 Solution

```
library(MASS)  # load MASS package
str(Pima.tr2)  # look at structure of Pima.tr2
summary(Pima.tr2) # summarize the stats of the data set
mean(Pima.tr2$bmi) # this mean cannot be computed: NA
mean(Pima.tr2$bmi, na.rm=TRUE) # the 3 NAs have been removed
```

11 excuRsion: category subsets with `tapply` (Matloff)

- The built-in `ToothGrowth` data set contains the numeric variable `len` (length of a tooth), and the categorical variable `supp` with two levels, `OJ` and `VC` for "Orange juice" and "Vitamin C".

```
str(ToothGrowth)
```

- Let's say we want to know the mean length `ToothGrowth$len` for each of the two `levels`.

- The `tapply` function allows us to split the vector `X = ToothGrowth$len` in two groups according to the values of `INDEX = Toothgrowth$supp`, and then apply the function `FUN = mean`.

```
tapply(X=ToothGrowth$len, INDEX=ToothGrowth$supp, FUN=mean)
```

- `[X]` Check out `help(tapply)` - remember to enter system commands in the R console **(why is `help` a system command?)**

12 Practice: mean weight of chicks by feed type

- `[ ]`

  If you want to find the mean weight of chicks grouped by feed type, you could use `mean` on each specific subset - how would this look like?

```
mean(chickwts$weight[chickwts$feed == "casein"])
mean(chickwts$weight[chickwts$feed == "horsebean"])
mean(chickwts$weight[chickwts$feed == "linseed"])
mean(chickwts$weight[chickwts$feed == "meatmeal"])
mean(chickwts$weight[chickwts$feed == "soybean"])
mean(chickwts$weight[chickwts$feed == "sunflower"])
```

- `[ ]`

  Instead, use `tapply(X, INDEX, FUN)` to calculate these values using just one line of code! Remember: `X` is the vector, `INDEX` is the splitting category (`factor` level), and `FUN` is the function.

```
tapply(
  X = chickwts$weight,
  INDEX = chickwts$feed,
  FUN = mean)
```

13 Counting chicks

- Sometimes it's useful to summarize non-numerical data, e.g. the number of observations that fall in a particular category
- *Counts* or *frequencies* are summary statistics of categorical data

- Again you can use the contingency `table` command for frequencies - e.g. for the feed types in the `chickwts` data set.

```
table(chickwts$feed)
```

14 Visualizing contingency tables

- `[ ]` **How would you visualize this table?** Think about the data and about generic plotting in R.

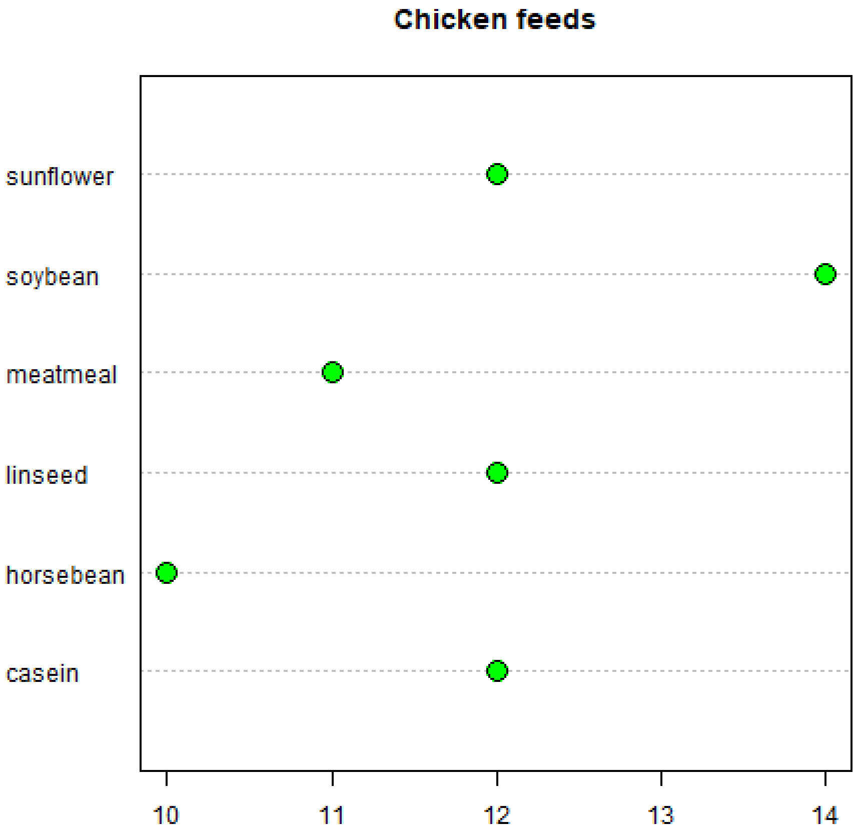
14.1 Solution

- Plug the table into `plot` - it's a generic graphic function and it does have a `plot.table` method.

```
plot(x=table(chickwts$feed),
     main="Chicken feeds",
     ylab="Frequencies")
```

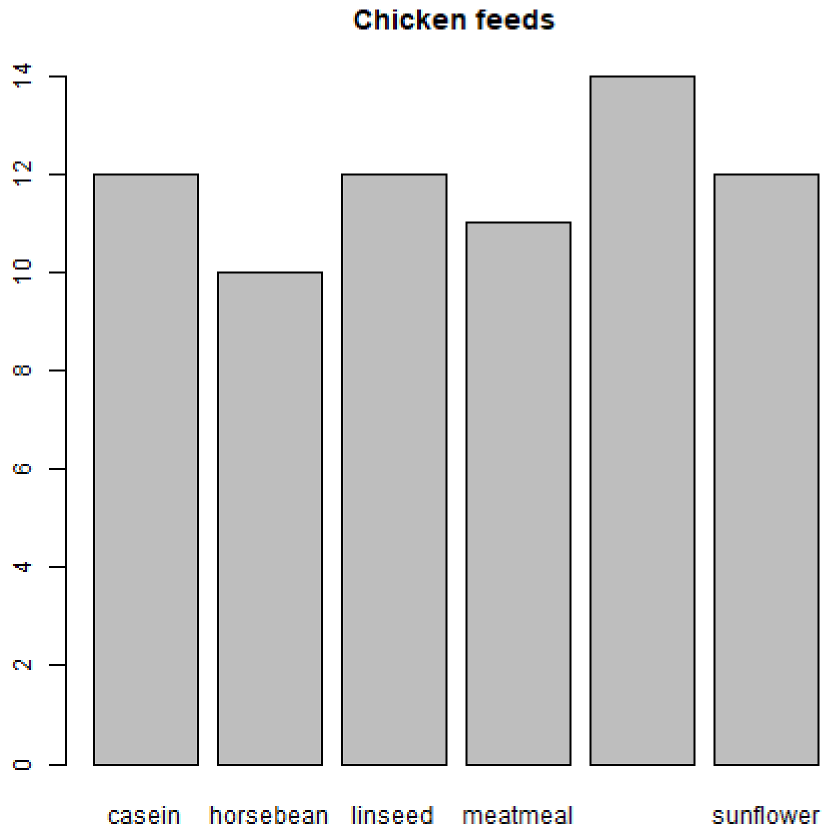
- "Thin frequencies" like this are better represented as dots in a so-called dot plot, with the `dotchart` function (is it generic?):

```
dotchart(x=table(chickwts$feed),
         main="Chicken feeds",
         pch=21, bg="green", pt.cex=2)
```

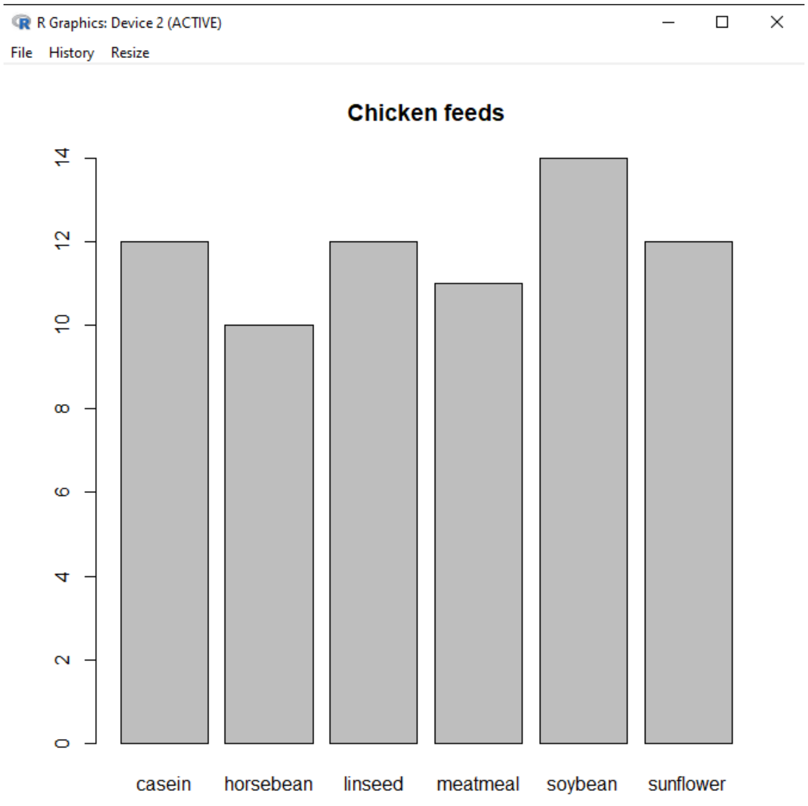


- Since the table contains just categories, you could also try and plug them directly into `plot`, without going through `table`:

```
plot(x=chickwts$feed,
     main="Chicken feeds")
```



- **Bonus (10 pts):** find out why the last graph does not show the `level soybean`? When executed in the R console, the command works:



15 Proportions with `table`

- More information from counts includes the proportion of observations that fall into each category.
- Proportions represent the fraction of observations in each category, usually as a decimal number between 0 and 1, and they add up to 1.

- For numeric vectors like our sample vector, there is a function, `prop.table`.

```
x <- c(2,4.4,3,3,2,2.2,2,4) # store observations in vector
prop.table(x)  # prop.table works for numeric vectors
sum(prop.table(x)) # proportions add up to 1
```

- For category vectors, or factors with levels, you need to divide the count by the overall sample size, which is `nrow(chickwts)`.

```
table(chickwts$feed) / nrow(chickwts)
table(chickwts$feed)
sum(table(chickwts$feed)) # counts add up to total no. of records
nrow(chickwts) # number of rows in the data set
```

16 Proportions with logical flag vectors

- You do not always need `table` - the sum over a logical flag vector is just as good because such a vector of `TRUE` and `FALSE` is coded as a vector of `0` and `1`.
- Example: `chickwts$feed == "soybean"` lists all chicks fed wih `soybean` as `TRUE` (or 1):

```
chickwts$feed == "soybean"
```

- For example, to find the proportion of chicks fed `soybean`:

```
sum(chickwts$feed == "soybean") / nrow(chickwts)
```

- This is equivalent to averaging over the logical flag vector:

```
mean(chickwts$feed == "soybean")
```

- You can use this approach to calculate the proportion of entities in groups. E.g. the proportion of chicks fed `soybean` or `horsebean`:

```
mean(chickwts$feed == "soybean" | chickwts$feed == "horsebean")
```

- This computation uses the following logical argument vector:

```
chickwts$feed == "soybean" | chickwts$feed == "horsebean"
```

17 Proportions with `tapply`

- You can also use `tapply` with the `FUN` argument to be an anonymous (non-named) function that computes the mean for each `feed` level

```
prop <- tapply(
  X = chickwts$weight,  # object that can be split by factor levels
  INDEX = chickwts$feed, # list of factors
  FUN = function(x) length(x)/nrow(chickwts)) # function to be applied
                                        # to factors
prop
sum(prop)
```

- Here, the anonymous function is defined with a dummy argument `x`.

18 Rounding numeric data with `round`

- The `round` function rounds numeric data output to a certain number of decimal places. It has only two arguments, input data and digits.

```
round(
  table(chickwts$feed) / nrow(chickwts), # input data
  digits = 3) # output digits
```

19 Percentages vs. proportion

- Percentage and proportion represent the same thing.
- They differ in scale - percentage is proportion multipled by 100.

- The percentage of chicks on a soybean diet is approximately 19.7%

```
round(
  x = mean(chickwts$feed == "soybean") * 100,
  digits = 1)
```

- Proportions always lie in [0,1] while percentages lie in [0,100].
- Statisticians prefer percentages when discussing percentiles, and proportions when discussing probabilities.

20 Exercises (for home)

- Download practice file from GitHub and save as Org-mode file
- Submit completed Org-mode file to Canvas by Thursday, 22-Sept, 8 am

21 TODO Glossary: concepts

TERM	MEANING

22 TODO Glossary: code

CODE	MEANING

23 References

- DataCamp (n.d.). Introduction to Statistics. URL: datacamp.com.
- Davies TD (2016). Book of R. NoStarch Press. URL: nostarch.com
- Matloff N (2022). fasteR. URL: github.com/matloff/fasteR

Author: MARCUS BIRKENKRAHE

Created: 2022-09-22 Thu 17:59