COMPSCIX 415.2 Homework 4

Vishnu Vardhan

2/23/2018

Q 5.6.7.2

Come up with another approach that will give you the same output as not_cancelled %>% count(dest) and not_cancelled %>% count(tailnum, wt = distance) (without using count()).

Answer

Lets first find out what the expected output is:

```
not_cancelled <- flights %>% filter(!is.na(dep_delay),!is.na(arr_delay))

not_cancelled %>% count(dest)
```

```
## # A tibble: 104 x 2
##     dest     n
##    <chr> <int>
##  1   ABQ   254
##  2   ACK   264
##  3   ALB   418
##  4   ANC     8
##  5   ATL 16837
##  6   AUS  2411
##  7   AVL   261
##  8   BDL   412
##  9   BGR   358
## 10   BHM   269
## # ... with 94 more rows
```

```
not_cancelled %>% count(tailnum, wt = distance)
```

```
## # A tibble: 4,037 x 2
##    tailnum      n
##      <chr>  <dbl>
##  1  D942DN   3418
##  2  N0EGMQ 239143
##  3  N10156 109664
##  4  N102UW  25722
##  5  N103US  24619
##  6  N104UW  24616
##  7  N10575 139903
##  8  N105UW  23618
##  9  N107US  21677
## 10  N108UW  32070
## # ... with 4,027 more rows
```

We can recreate this using group_by and summarse as follow:

```
not_cancelled %>% group_by(dest) %>% summarise(n = n())
```

```
## # A tibble: 104 x 2
##     dest     n
##    <chr> <int>
##  1   ABQ   254
##  2   ACK   264
##  3   ALB   418
##  4   ANC     8
##  5   ATL 16837
##  6   AUS  2411
##  7   AVL   261
##  8   BDL   412
##  9   BGR   358
## 10   BHM   269
## # ... with 94 more rows
```

```
not_cancelled %>% group_by(tailnum) %>%  summarise( n = sum(distance))
```

```
## # A tibble: 4,037 x 2
##    tailnum      n
##      <chr>  <dbl>
##  1  D942DN   3418
##  2  N0EGMQ 239143
##  3  N10156 109664
##  4  N102UW  25722
##  5  N103US  24619
##  6  N104UW  24616
##  7  N10575 139903
##  8  N105UW  23618
##  9  N107US  21677
## 10  N108UW  32070
## # ... with 4,027 more rows
```

Q 5.6.7.4

Look at the number of cancelled flights per day. Is there a pattern? Is the proportion of cancelled flights related to the average delay?

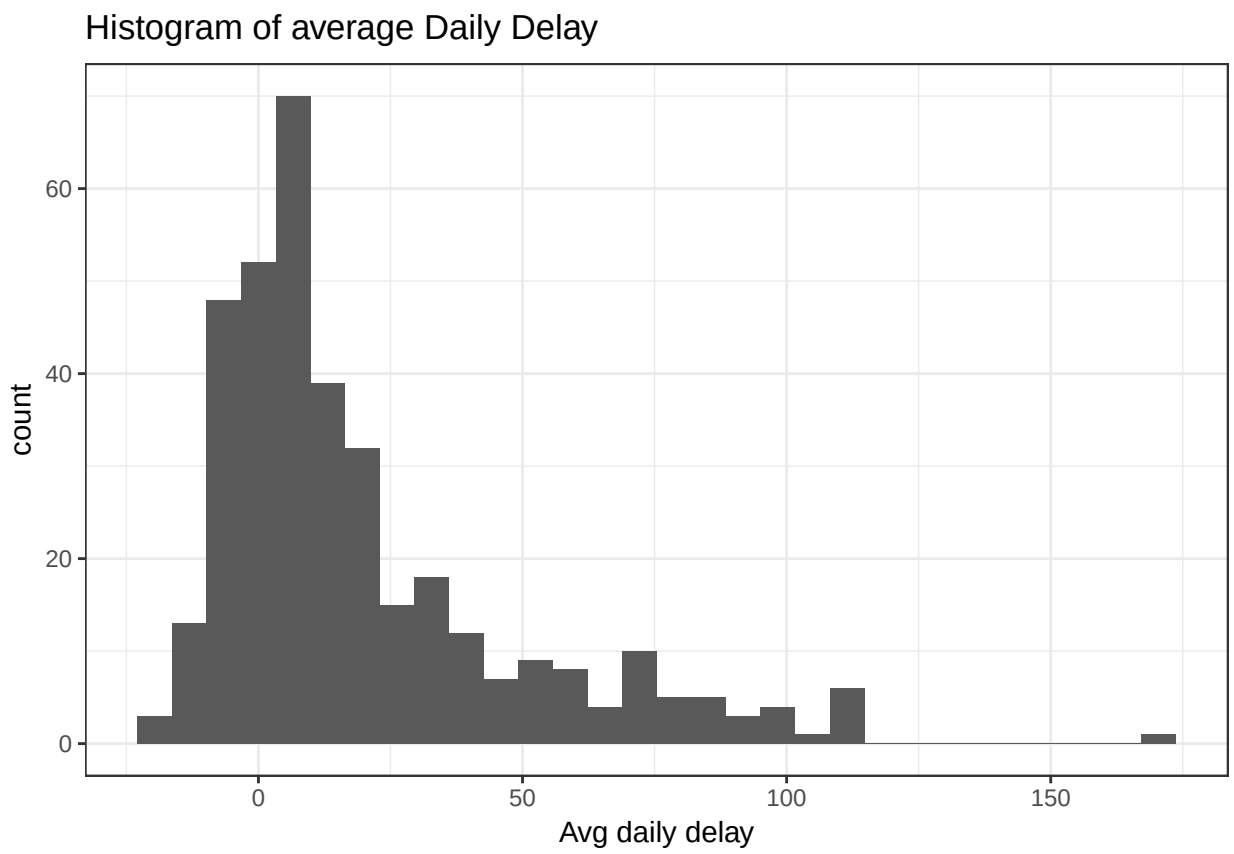
Answer

A histogram shows that most delays seem to be less than 25 minutes. As the delay incrases the number of cancelled flights increase. The effect is much more severe after the 25 minute mark.

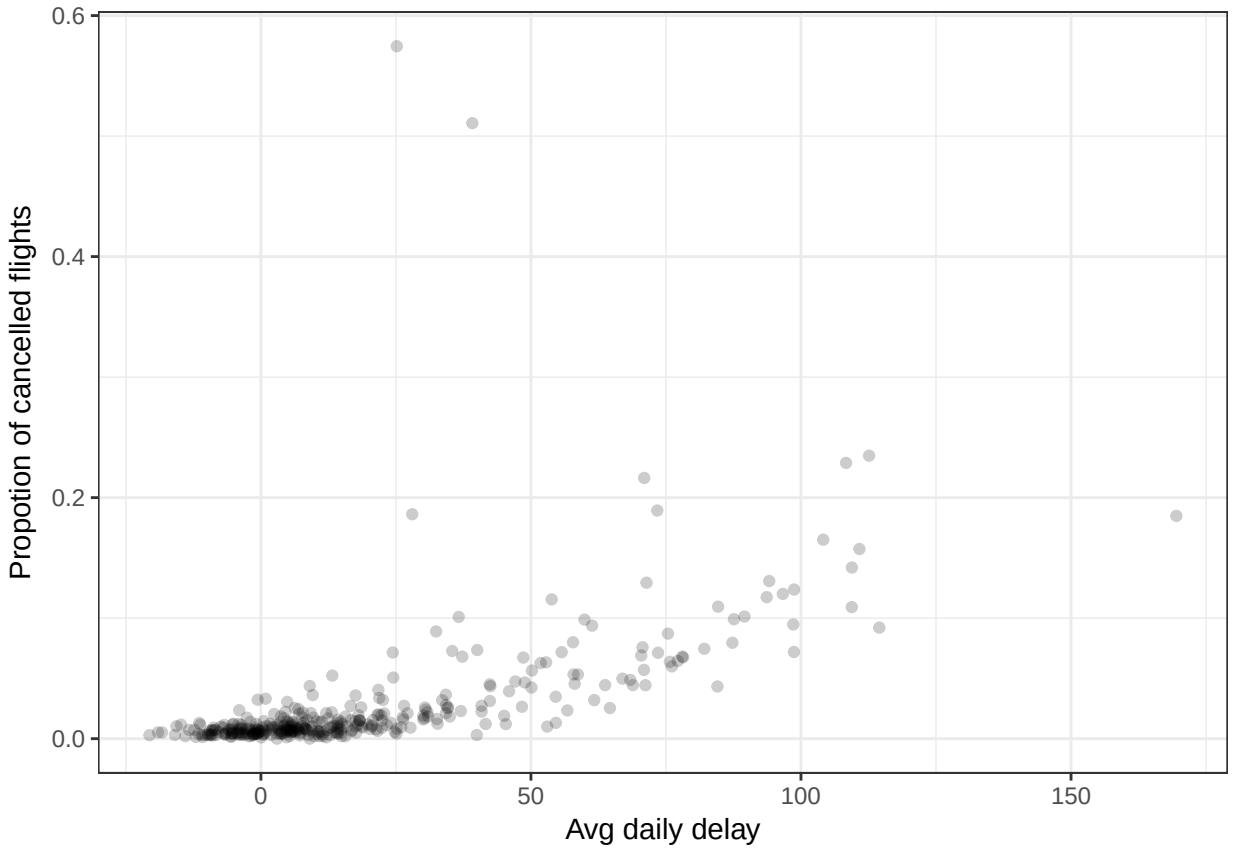
```
d <- flights %>% group_by(year,month,day) %>%
  summarise(
    c_mean     = mean(is.na(dep_delay) | is.na(arr_delay)),
    delay_mean = mean(dep_delay + arr_delay, na.rm = TRUE))

ggplot(data = d) + geom_histogram(aes(delay_mean))  +
  labs(title = "Histogram of average Daily Delay", x = "Avg daily delay") +
  theme_bw()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
ggplot(data = d,mapping = aes(delay_mean,c_mean)) +
  geom_point(alpha=0.2) +
  labs(y = "Propotion of cancelled flights", x = "Avg daily delay") +
  theme_bw()
```



Q 5.6.7.5

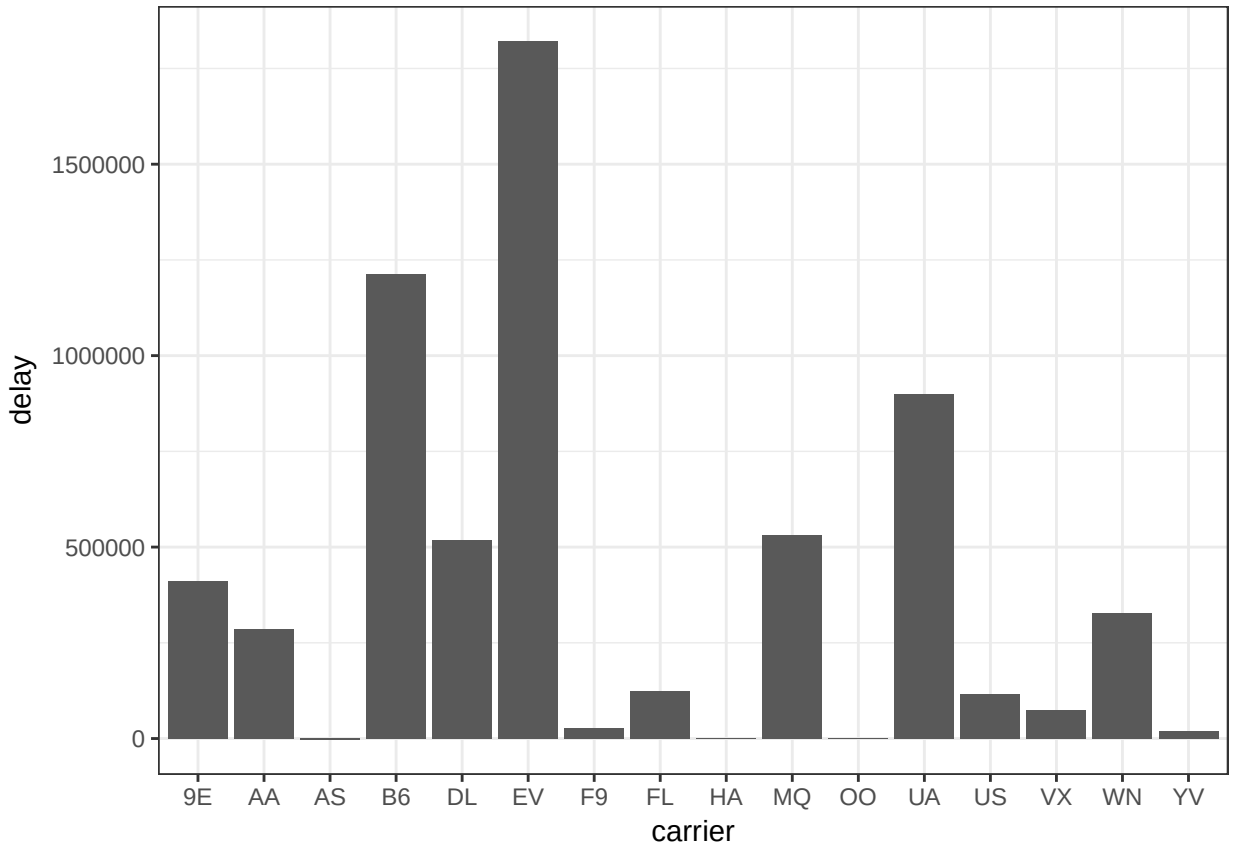
Which carrier has the worst delays? Challenge: can you disentangle the effects of bad airports vs. bad carriers? Why/why not? (Hint: think about flights %>% group_by(carrier, dest) %>% summarise(n()))

Answer

The carrier EV has the worst delays.

```
worst_carrier <- flights %>% group_by(carrier) %>%
  summarise( delay = sum(dep_delay + arr_delay, na.rm = TRUE))


ggplot(data = worst_carrier, mapping = aes(x=carrier, y=delay)) +
  geom_col() +
  theme_bw()
```



As can be seen in the table below, the number of flights into a particular destination can be fairly large. There are 314 such carrier/dest combinations.

I believe the way to test dependence/inter-dependence is to identify if each of these groups is a normal distribution. If it is normal, then there is no relationship between that combination of airport/carrier. Once we identify combinations that are related, then we need to compare the distribution of the combination versus the two populations of airline and carrier.

i.e Lets assume we identify that DL/ATL with 10571 flights is a non-normal distribution with respect to delays.

1. Compre the distribution of delays of all DL flights with that of DL/ATL flights. Similar distribution means Atlanta is not playing any role in delta's delays. Else, Atlanta is playing a role.
2. Compre distribution of delays of all Atlanta flights, with that of all DL/ATL flights. Similar distribution means delta is not playing any role in delays seen at Atlanta. Else, delta is playing a role.

Quick research indicates that we should likely use the Q-Q plot, but i don't understand this enough to plot it for this assignment.

```
flights %>% group_by(carrier,dest) %>%
  summarise(c = n()) %>%
  arrange(desc(c))
```

```
## # A tibble: 314 x 3
## # Groups:   carrier [16]
##    carrier  dest      c
##      <chr> <chr> <int>
##  1      DL   ATL 10571
##  2      US   CLT  8632
```

```
##  3      AA   DFW  7257
##  4      AA   MIA  7234
##  5      UA   ORD  6984
##  6      UA   IAH  6924
##  7      UA   SFO  6819
##  8      B6   FLL  6563
##  9      B6   MCO  6472
## 10      AA   ORD  6059
## # ... with 304 more rows
```

Q 5.6.7.6

What does the sort argument to count() do. When might you use it?

Answer

Lets start with a bunch of experiements with the following code:

flights %>% group_by(carrier,dest) %>% summarise(c = n()) %>% arrange(desc(c)) flights %>% group_by(carrier,dest) %>% count(sort = TRUE) flights %>% count(sort = TRUE) flights %>% group_by(carrier) %>% count(sort = TRUE) flights %>% group_by(carrier,dest) %>% summarise(c = n()) %>% count(sort = TRUE)

The help states that: Count/tally observations by group

'sort = TRUE', changes how the tibble is ordered, and is the same as using 'arrange' with a 'desc' function.

Q 10.5.1

How can you tell if an object is a tibble? (Hint: try printing mtcars, which is a regular data frame).

Answer

Printing a tibble prints a summary, describing the number of rowsXcolumns, followed by the number of columns that print on the screen, and their type, the data for these columns, and an end summary that prints the remaning columns that did not fit on the screen and their respective types.

Reading the chapter reveals that a tibble is, in general, a much more restricted data frame. Reducing functionality vs a raw data.frame, but when combined with functions in the library result in much cleaner, re-usable code.

Q 10.5.2

Compare and contrast the following operations on a data.frame and equivalent tibble. What is different? Why might the default data frame behaviours cause you frustration?

df <- data.frame(abc = 1, xyz = "a") df$x df[, "xyz"] df[, c("abc", "xyz")]

Answer

Trying to do the same thing with tibbles needs the following code

```
t <- tribble( ~abc, ~xyz, 1,"a")
```

Running these commands shows that each is a way to select columns. The essential

The key difference is the partial column match that data.frames allow. Partial column matches, though interesting, make it very difficult to predictability write code when you have large numbers of columns, from disparate sources.

Q 10.5.3

If you have the name of a variable stored in an object, e.g. var <- "mpg", how can you extract the reference variable from a tibble?

Answer

You can use the '[[' operator to access the values.

Example:

```
t <- tribble( ~abc, ~xyz, 1,"a")
var <- "xyz"
t[[var]]
```

```
## [1] "a"
```

Q 10.5.6

What option controls how many additional column names are printed at the footer of a tibble?

Answer

Looking at the package options in the help, shows that the option to control the number of extra columns is:

tibble.max_extra_cols

```
options(tibble.max_extra_cols=1)
```

Q 12.3.3.2

Why does this code fail?

table4a %>% gather(1999, 2000, key = "year", value = "cases") > Error in combine_vars(vars, ind_list): Position must be between 0 and n

Answer

Because table4a is a tibble. Tibble column naming, does not follow r column naming syntax. 1999 does not begin with an alphabet - a requirement for r data frames.

Here is what a valid name looks like in R https://cran.r-project.org/doc/FAQ/R-FAQ.html#What-are-valid-names_003f

enclosing them in backticks makes this work.

This stack overflow question refers to backticks. https://stackoverflow.com/questions/36220823/what-do-backticks-do-in-r

I quote "A pair of backticks is a way to refer to names or combinations of symbols that are otherwise reserved or illegal."

```
table4a %>%  gather(`1999`, `2000`, key = "year", value = "cases")
```

```
## # A tibble: 6 x 3
##       country  year  cases
##         <chr> <chr>  <int>
## 1 Afghanistan  1999    745
## 2      Brazil  1999  37737
## 3       China  1999 212258
## 4 Afghanistan  2000   2666
## 5      Brazil  2000  80488
## 6       China  2000 213766
```

Q 12.3.3.3

Why does spreading this tibble fail? How could you add a new column to fix the problem?

people <- tribble(~name, ~key, ~value, #————————|———|—— “Phillip Woods”, “age”, 45, “Phillip Woods”, “height”, 186, “Phillip Woods”, “age”, 50, “Jessica Cordero”, “age”, 37, “Jessica Cordero”, “height”, 156)

Answer

spreading fails, because there is a duplicate entry for the age of Phillip Woods.

After researching online, including this stackoverflow response https://stackoverflow.com/questions/43259380/spread-with-duplicate-identifiers-using-tidyverse-and/43259735#43259735

I came up with:

```
people <- tribble(
  ~name,             ~key,    ~value,
  #-----------------/--------/------
  "Phillip Woods",   "age",       45,
  "Phillip Woods",   "height",   186,
  "Phillip Woods",   "age",       50,
  "Jessica Cordero", "age",       37,
  "Jessica Cordero", "height",   156
)


people %>% group_by(name,key) %>% mutate( id = row_number()) %>% spread(key,value)
```

```
## # A tibble: 3 x 4
## # Groups:   name [2]
##              name    id   age height
## *           <chr> <int> <dbl>  <dbl>
## 1 Jessica Cordero     1    37    156
## 2   Phillip Woods     1    45    186
## 3   Phillip Woods     2    50     NA
```

Q 12.3.3.4

Tidy the simple tibble below. Do you need to spread or gather it? What are the variables?

preg <- tribble(~pregnant, ~male, ~female, “yes”, NA, 10, “no”, 20, 12)

Answer

This is a gather preoblem, and the variables are gender and some "value"

```
preg <- tribble(
  ~pregnant, ~male, ~female,
  "yes",     NA,    10,
  "no",      20,    12
)

preg %>% gather(male, female, key = "gender", value = "value")
```

```
## # A tibble: 4 x 3
##   pregnant gender value
##      <chr>  <chr> <dbl>
## 1      yes   male    NA
## 2       no   male    20
## 3      yes female    10
## 4       no female    12
```

Q 12.4.3.1

What do the extra and fill arguments do in separate()? Experiment with the various options for the following two toy datasets.

tibble(x = c("a,b,c", "d,e,f,g", "h,i,j")) %>% separate(x, c("one", "two", "three"))

tibble(x = c("a,b,c", "d,e", "f,g,i")) %>% separate(x, c("one", "two", "three"))

Answer

seperate splits the input values into multiple pieces. extra and fill specify the actions to take when there are too many or too few pieces.

By default both just warn.

The additional actions for extra include: - drop and merge

The additional actions for fill include: - identifies where the missing values are - right - missing values on the right - left - missing values on the left - this should not be confused with the 'fill' command. This command does not fill replacements, only where the NA values are displayed

```
# This tibble has extra elements
e <- tibble(x = c("a,b,c", "d,e,f,g", "h,i,j"))
e %>% separate(x, c("one", "two", "three"))
```

```
## Warning: Too many values at 1 locations: 2
```

```
## # A tibble: 3 x 3
##     one   two three
## * <chr> <chr> <chr>
## 1     a     b     c
## 2     d     e     f
## 3     h     i     j
```

```
e %>% separate(x, c("one", "two", "three"), extra = "drop")
```

```
## # A tibble: 3 x 3
##     one   two three
```

```
## * <chr> <chr> <chr>
## 1     a     b     c
## 2     d     e     f
## 3     h     i     j
```

```
e %>% separate(x, c("one", "two", "three"), extra = "merge")
```

```
## # A tibble: 3 x 3
##     one   two three
## * <chr> <chr> <chr>
## 1     a     b     c
## 2     d     e   f,g
## 3     h     i     j
```

```
# Lets deal with fill
e %>% separate(x, c("one", "two", "three","four"))
```

```
## Warning: Too few values at 2 locations: 1, 3
```

```
## # A tibble: 3 x 4
##     one   two three  four
## * <chr> <chr> <chr> <chr>
## 1     a     b     c  <NA>
## 2     d     e     f     g
## 3     h     i     j  <NA>
```

```
e %>% separate(x, c("one", "two", "three","four"),fill = "left")
```

```
## # A tibble: 3 x 4
##     one   two three  four
## * <chr> <chr> <chr> <chr>
## 1  <NA>     a     b     c
## 2     d     e     f     g
## 3  <NA>     h     i     j
```

```
e %>% separate(x, c("one", "two", "three","four"), fill = "right")
```

```
## # A tibble: 3 x 4
##     one   two three  four
## * <chr> <chr> <chr> <chr>
## 1     a     b     c  <NA>
## 2     d     e     f     g
## 3     h     i     j  <NA>
```

Q 12.4.3.2

Both unite() and separate() have a remove argument. What does it do? Why would you set it to FALSE?

Answer

The remove argument removes the orignal row/s that unite and seperate work on. It makes sense, if the unite/seperate conversion is pretty deterministic (e.g years and dates), but if there is uncertainity in the input data, with potential to drop/lose data, it may make sense to keep the original data set arround.

Answer these questions

Follow these steps: - Download the baby_names.txt file from Canvas which is in the Homework 4 assignment section. - Load this file into R correctly and take a glimpse of the output. - Export this file as an rds file and call it 'baby_names.rds'. - Reload the baby_names.rds file and take another glimpse. - Show all of your code and the output. There should be two data import lines of code, one data export line of code, and two glimpses of the data.

Answer

Here is the solution to the question.

```
baby_names <- read.delim("baby_names.txt", header = TRUE, sep = "|")
glimpse(baby_names)
```

```
## Observations: 30,000
## Variables: 5
## $ year <int> 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 188...
## $ sex  <fctr> F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F,...
## $ name <fctr> Mary, Anna, Emma, Elizabeth, Minnie, Margaret, Ida, Alic...
## $ n    <int> 7065, 2604, 2003, 1939, 1746, 1578, 1472, 1414, 1320, 128...
## $ prop <dbl> 0.072384329, 0.026679234, 0.020521700, 0.019865989, 0.017...
```

```
saveRDS( baby_names, "baby_names.rds")
new_baby_names <- readRDS("baby_names.rds")
glimpse(new_baby_names)
```

```
## Observations: 30,000
## Variables: 5
## $ year <int> 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 1880, 188...
## $ sex  <fctr> F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F, F,...
## $ name <fctr> Mary, Anna, Emma, Elizabeth, Minnie, Margaret, Ida, Alic...
## $ n    <int> 7065, 2604, 2003, 1939, 1746, 1578, 1472, 1414, 1320, 128...
## $ prop <dbl> 0.072384329, 0.026679234, 0.020521700, 0.019865989, 0.017...
```

```
identical(baby_names, new_baby_names, ignore.environment = TRUE)
```

```
## [1] TRUE
```