COMPSCIX 415.2 Homework 4

Rajat Jain

July 2nd, 2018

Contents

Section 5.6.7 Exercises 1
Section 10.5 Exercises 6
Section 12.3.3 Exercises 8
Section 12.4.3 Exercises 10
Additional Question 12

Section 5.6.7 Exercises

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()`).**

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

# Equivalent of not_cancelled %>% count(dest)
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
```

```
# Equivalent of not_cancelled %>% count(tailnum, wt = distance)
not_cancelled %>%
  group_by(tailnum) %>%
  summarise(
    n = sum(distance)
    )
```

```
## # A tibble: 4,037 x 2
```

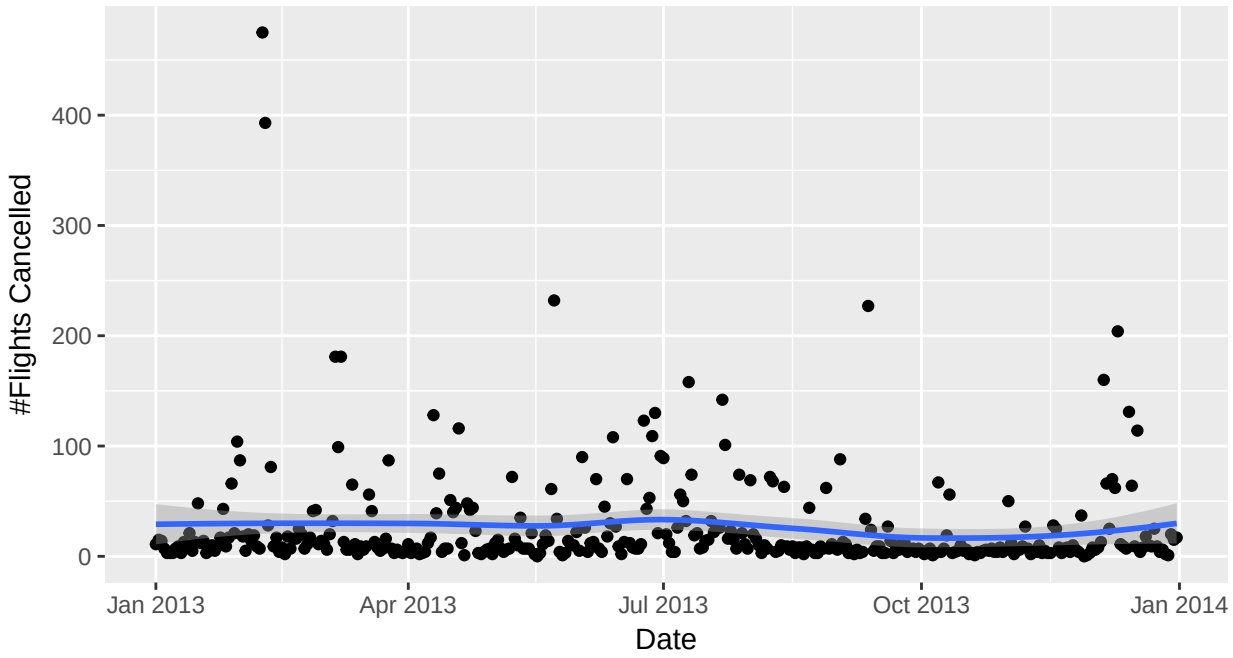
```
##    tailnum       n
##    <chr>     <dbl>
##  1 D942DN    3418.
##  2 N0EGMQ  239143.
##  3 N10156  109664.
##  4 N102UW   25722.
##  5 N103US   24619.
##  6 N104UW   24616.
##  7 N10575  139903.
##  8 N105UW   23618.
##  9 N107US   21677.
## 10 N108UW   32070.
## # ... with 4,027 more rows
```

We can get the same result as `count()` by using `group_by()` with `summarise()` as hsown above.

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?**

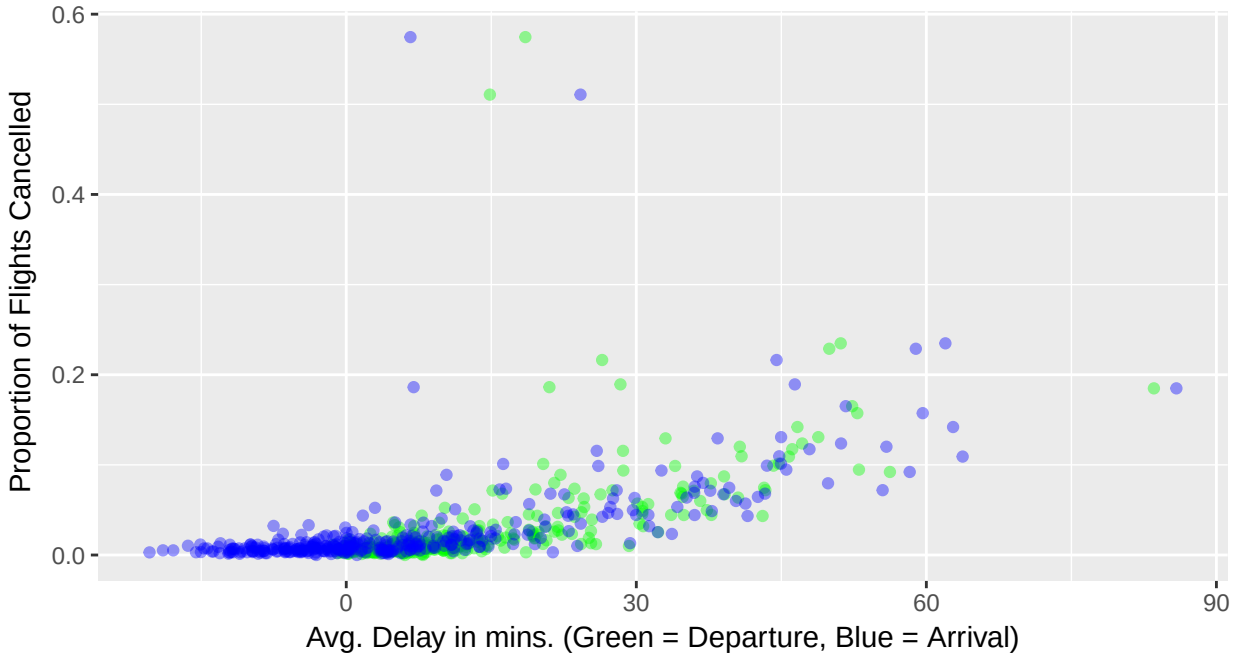
```
cancelled <- flights %>%
  mutate(
    date = make_datetime(year, month, day),
    is_cancelled = (is.na(dep_delay) | is.na(arr_delay))
  ) %>%
  group_by(date) %>%
  summarise(
    num_cancelled  = sum(is_cancelled, na.rm=TRUE),
    prop_cancelled = mean(is_cancelled, na.rm=TRUE),
    avg_dep_delay  = mean(dep_delay, na.rm=TRUE),
    avg_arr_delay  = mean(arr_delay, na.rm=TRUE)
  )

# Flights cancelled per day.
ggplot(data = cancelled, aes(x=date, y=num_cancelled)) +
  geom_point() + geom_smooth() +
  xlab("Date") + ylab("#Flights Cancelled")
```



There is no pattern in cancelled flights per day. The smooth line through the plot is a fairly flat line with very low variance. There are however, some days with extremely high number of cancelled flight. This may be due to factors like weather.

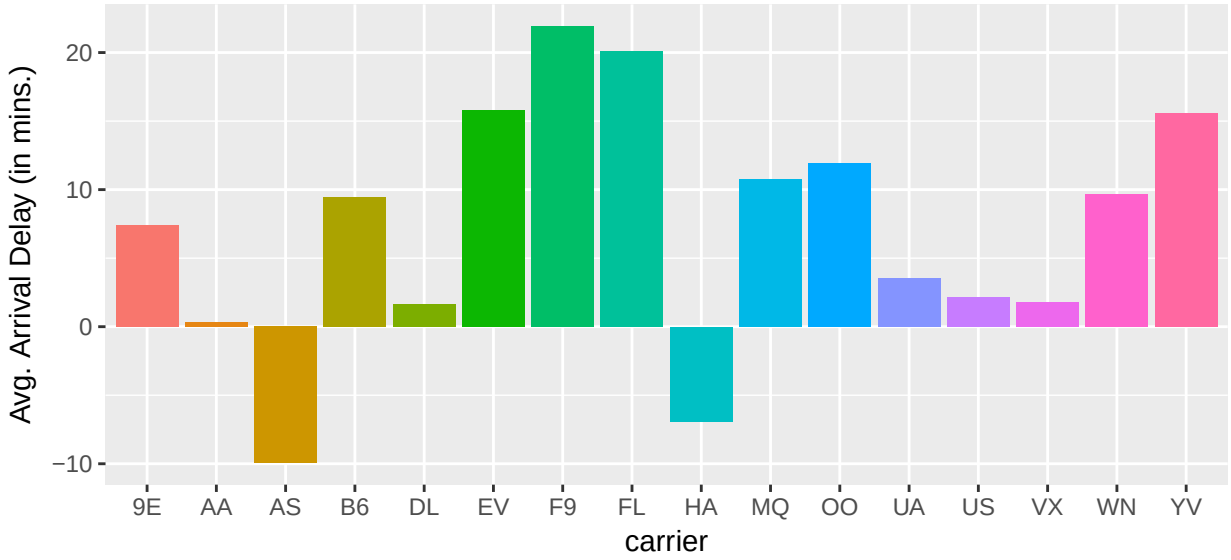
```
# Proportion of cancelled flights related to the average delay.
ggplot(data = cancelled, aes(y = prop_cancelled)) +
  geom_point(aes(x = avg_dep_delay), color = "green", alpha = "0.4") +
  geom_point(aes(x = avg_arr_delay), color = "blue", alpha = "0.4") +
  xlab("Avg. Delay in mins. (Green = Departure, Blue = Arrival)") +
  ylab(" Proportion of Flights Cancelled")
```



The proportion of cancelled flights also doesn't seem to be related to the average departure or arrival delay as shown by the plot above.

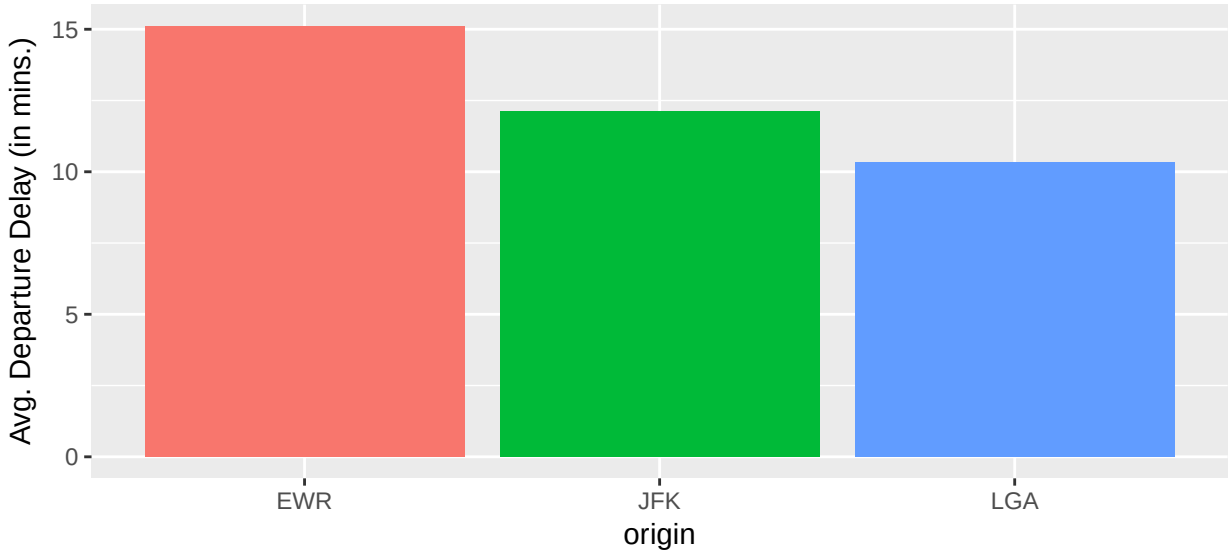
5. **Which carrier has the worst delays? Challenge: can you disentangle the effects of bad airports vs. bad carriers? Why/why not? (Hint: think about `flights %>% group_by(carrier, dest) %>% summarise(n())`)**

```
flights %>%
  group_by(carrier) %>%
  # Effective delay is delay in reaching destination.
  summarise(avg_delay = mean(arr_delay, na.rm = TRUE)) %>%
  ggplot(aes(x = carrier, y = avg_delay, fill = carrier)) +
  geom_col(show.legend = FALSE) + ylab("Avg. Arrival Delay (in mins.)")
```



Frontier Airlines (`F9`) has the worst delays averaging over 20 mins of delay in arrival per flight.

```
flights %>%
  group_by(origin) %>%
  summarise(avg_delay = mean(dep_delay, na.rm = TRUE)) %>%
  ggplot(aes(x = origin, y = avg_delay, fill = origin)) +
  geom_col(show.legend = FALSE) + ylab("Avg. Departure Delay (in mins.)")
```



There are many destination airports, some with too few flights. It is very difficult to visualize and to disentangle the effects of those vs. bad carriers. Howerver, if we look at the departure delays by origin airport, it is clear that the flights originating from `EWR` certainly have higher departure delays which can be correlated to higher arrival delays as well.

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

The `sort` argument to `count()` sorts the output by descending order of n. We may use this when we are interested in looking at the top most frequent items. For example, the code below counts the number of flights by `carrier`. By using `sort = TRUE`, we get to see the carriers with the most number of flights at the top.

```
flights %>%
  count(carrier, sort = TRUE)
```

```
## # A tibble: 16 x 2
##    carrier     n
##    <chr>   <int>
##  1 UA      58665
##  2 B6      54635
##  3 EV      54173
##  4 DL      48110
##  5 AA      32729
##  6 MQ      26397
##  7 US      20536
##  8 9E      18460
##  9 WN      12275
## 10 VX       5162
## 11 FL       3260
## 12 AS        714
## 13 F9        685
## 14 YV        601
## 15 HA        342
## 16 OO         32
```

Section 10.5 Exercises

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

On printing a tibble only as many columns are printed as can fit on the screen and only first 10 rows are printed whereas a regular data frame is printed completely. Alternatively, we can also tell by running `class()` function on the object as shown in the example below.

```
class(mtcars)
```

```
## [1] "data.frame"
```

```
class(mpg)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

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?**

```
# Given data frame.
df <- data.frame(abc = 1, xyz = "a")
# Create a tibble from it.
tbl <- as_tibble(df)

# First Operation
df$x
```

```
## [1] a
## Levels: a
```

```
tbl$x
```

```
## Warning: Unknown or uninitialised column: 'x'.
```

```
## NULL
```

In this operation we are trying to extract the values from column `x` of the object which does not exist. However, in the data frame operation a value (of column `xyz`) is returned. This could cause unpredictable behaviors and frustration in a large project. The same operation on a tibble object returns `NULL` value and throws a Warning.

```
# Second Operation
df[, "xyz"]
```

```
## [1] a
## Levels: a
```

```
tbl[, "xyz"]
```

```
## # A tibble: 1 x 1
##   xyz
##   <fct>
## 1 a
```

```
# Third Operation
df[, c("abc", "xyz")]
```

```
##   abc xyz
```

```
## 1   1   a
```

```
tbl[, c("abc", "xyz")]
```

```
## # A tibble: 1 x 2
##     abc xyz
##   <dbl> <fct>
## 1    1. a
```

In this case, both data frame and tibble return similar outputs.

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?**

Consider the following example.

```
my_tbl <- tribble(
  ~make,      ~mpg,
  #----------/-----
  "Ford",     18,
  "Honda",    24,
  "Toyota",   23,
  "Mercedes", 14
)

var <- "mpg"

#1. Using double brackets.
my_tbl[[var]]
```

```
## [1] 18 24 23 14
```

```
#2. Using select verb from tidyverse package.
select(my_tbl, var)
```

```
## # A tibble: 4 x 1
##     mpg
##   <dbl>
## 1   18.
## 2   24.
## 3   23.
## 4   14.
```

We can use any of the two methods demonstrated above.

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

The option `n_extra` controls how many additional column names are printed at the footer of a tibble. (Courtesy: `?print.tbl_df`).

Section 12.3.3 Exercises

2. **Why does this code fail?**

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

The above code fails because the column names 1999 & 2000 are numbers. In general, R wants the column names to start with an alphabet. In this case it tries to interpret them as column indexes instead of column names.

```
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
```

The code works however, by applying a work-around for this problem using backticks on the column names.

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
)
```

Spreading this tibble fails because the name "Phillip Woods" is repeated with different values of "age" on Line# 1 & 3. To fix this, we could add another column which helps in uniquely identify the two rows. For example:

```
people <- tribble(
  ~name,             ~key,    ~value, ~id,
  #-----------------/--------/-------/----
  "Phillip Woods",   "age",       45,  1,
  "Phillip Woods",   "height",   186,  1,
  "Phillip Woods",   "age",       50,  2,
  "Jessica Cordero", "age",       37,  3,
  "Jessica Cordero", "height",   156,  3
)
spread(people, key = key, value = value)
```

```
## # A tibble: 3 x 4
##   name               id   age height
##   <chr>           <dbl> <dbl>  <dbl>
```

```
## 1 Jessica Cordero    3.   37.   156.
## 2 Phillip Woods      1.   45.   186.
## 3 Phillip Woods      2.   50.    NA
```

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
)
```

We need to gather it. Columns `male` and `female` will be gathered under column name `sex`.

```
gather(preg, sex, count, male, female)
```

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

Section 12.4.3 Exercises

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"))
```

```
## Warning: Expected 3 pieces. Additional pieces discarded in 1 rows [2].
```

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

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

```
## Warning: Expected 3 pieces. Missing pieces filled with `NA` in 1 rows [2].
```

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

`extra` controls what happens when there are too many pieces to separate. There are three valid options:

- "warn" (the default): emit a warning and drop extra values
- "drop": drop any extra values without a warning
- "merge": only splits at most length(into) times (Courtesy: `?separate`)

```
tibble(x = c("a,b,c", "d,e,f,g", "h,i,j")) %>%
  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
```

`fill` controls what happens when there are not enough pieces. There are three valid options:

- "warn" (the default): emit a warning and fill from the right
- "right": fill with missing values on the right
- "left": fill with missing values on the left (Courtesy: `?separate`)

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

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

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

If `remove` argument is set to `TRUE`, it removes input columns from output data frame. It is set to `TRUE` by default. We can set it to `FALSE` when we want to preserve the original columns and create new columns using either `unite()` or `separate()`. Here are a few examples which illuatrate this behavior:

```
people <- tibble(name = c("JAIN, Rajat", "DOW, John", "DOW, Jane"))
people
```

```
## # A tibble: 3 x 1
##   name
##   <chr>
## 1 JAIN, Rajat
## 2 DOW, John
## 3 DOW, Jane
```

```
# Separate - Default behavior.
separate(people, name, c("last_name", "first_name"), sep = ", ")
```

```
## # A tibble: 3 x 2
##   last_name first_name
##   <chr>     <chr>
## 1 JAIN      Rajat
## 2 DOW       John
## 3 DOW       Jane
```

```
# Separate - With remove set to FALSE.
people_new <- separate(people, name, c("last_name", "first_name"), sep = ", ", remove = FALSE)
people_new
```

```
## # A tibble: 3 x 3
##   name        last_name first_name
##   <chr>       <chr>     <chr>
## 1 JAIN, Rajat JAIN      Rajat
## 2 DOW, John   DOW        John
## 3 DOW, Jane   DOW        Jane
```

```
# Unite - Default behavior.
unite(people_new, name_formatted, first_name, last_name, sep = " ")
```

```
## # A tibble: 3 x 2
##   name        name_formatted
##   <chr>       <chr>
## 1 JAIN, Rajat Rajat JAIN
## 2 DOW, John   John DOW
## 3 DOW, Jane   Jane DOW
```

```
# Unite - With remove set to FALSE.
unite(people_new, name_formatted, first_name, last_name, sep = " ", remove = FALSE)
```

```
## # A tibble: 3 x 4
##   name        name_formatted last_name first_name
##   <chr>       <chr>          <chr>     <chr>
## 1 JAIN, Rajat Rajat JAIN     JAIN      Rajat
## 2 DOW, John   John DOW       DOW       John
## 3 DOW, Jane   Jane DOW       DOW       Jane
```

Additional Question

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 a csv file and call it 'baby_names.csv'.**
- **Reload the baby_names.csv 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.**

```
baby_names <- read_delim(file = "baby_names.txt", delim="|")
```

```
## Parsed with column specification:
## cols(
##   year = col_integer(),
##   sex = col_character(),
##   name = col_character(),
##   n = col_integer(),
##   prop = col_double()
## )
```

```
glimpse(baby_names)
```

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

```
write_csv(baby_names, path = "baby_names.csv")

baby_names_2 <- read_csv(file = "baby_names.csv")
```

```
## Parsed with column specification:
## cols(
##   year = col_integer(),
##   sex = col_character(),
##   name = col_character(),
##   n = col_integer(),
##   prop = col_double()
## )
```

```
glimpse(baby_names)
```

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