COMPSCIX 415.2 Homework 4

Sanatan Das

February 24, 2018

Load packages

```
library(tidyverse)
library(nycflights13)
```

5.6.7 Exercises

QUESTION 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 2:

```
# collect all the data for not cancelled flights
not_cancelled <- flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay))

#provided query
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
```

```
# We can do the same operation without using count() as below.
# Explanation: first we do group by by destination column
# operation on all not_cancelled flights.
# After that we summarize n() on that result.
not_cancelled %>%
  group_by(dest) %>%
  summarize(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
```

```r
# provided query
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
```

```r
# We can do the same operation without using count() as below.
# Explanation: First we do a group by by tailnum column
# and then summarize on sum of the distance.
# NA records are removed.
not_cancelled %>%
  group_by(tailnum) %>%
  summarize(n = sum(distance, na.rm = TRUE))
```

```
## # 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
```

QUESTION 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 4:

The above question refers to "per day", does it mean grouping by day column in the data set or group by calendar day. If it is group by calendar day, it would require grouping by year, month, and day. We will try to analyze both ways.

```
# analysis on grouping by day column
flights %>%
  filter(is.na(dep_delay)) %>%
  count(day)
```

```
## # A tibble: 31 x 2
##      day     n
##    <int> <int>
##  1     1   246
##  2     2   250
##  3     3   109
##  4     4    82
##  5     5   226
##  6     6   296
##  7     7   318
##  8     8   921
##  9     9   593
## 10    10   535
## # ... with 21 more rows
```

```
flights %>%
  group_by(day) %>%
  summarize(prop_canceled = sum(is.na(dep_delay)) / n(),
            avg_delay = mean(dep_delay, na.rm = TRUE))
```

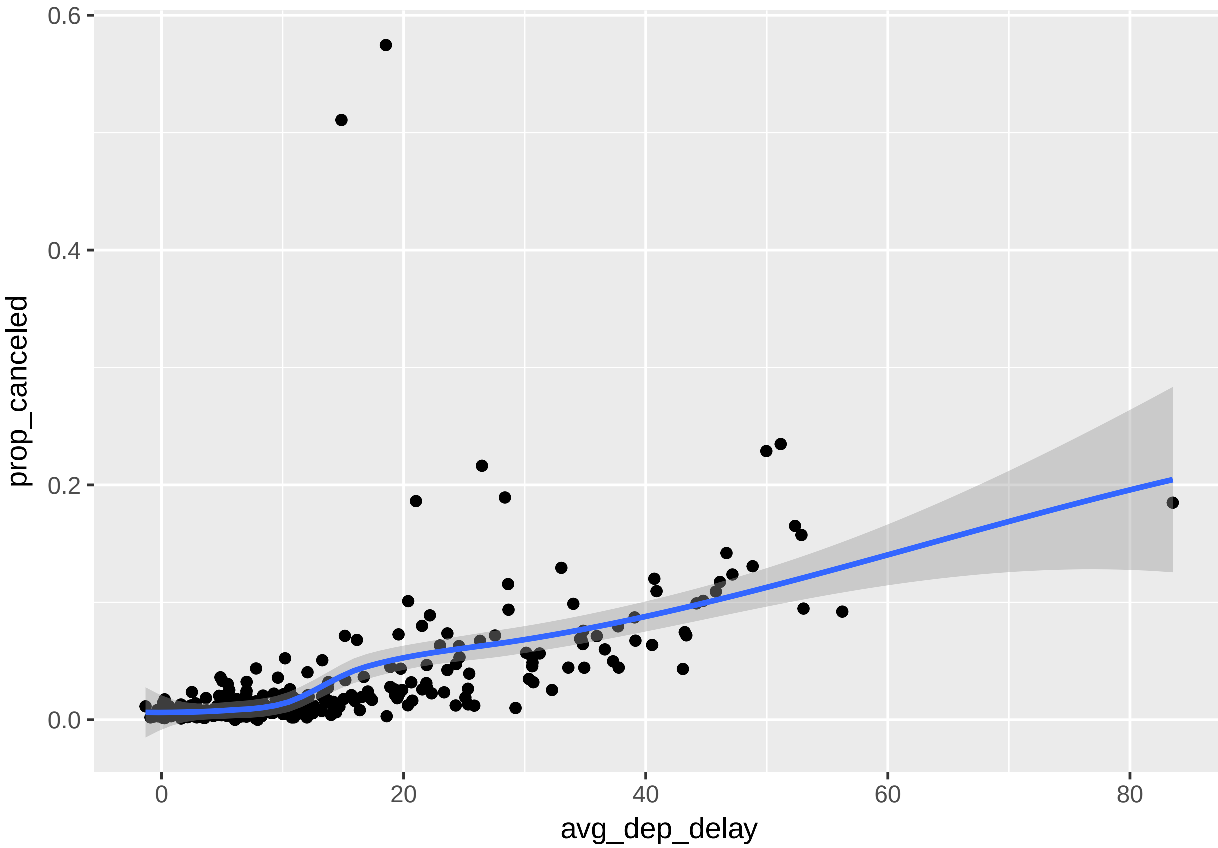
```
## # A tibble: 31 x 3
##      day prop_canceled avg_delay
##    <int>         <dbl>     <dbl>
##  1     1       0.0223      14.2
##  2     2       0.0231      14.1
##  3     3       0.00972     10.8
##  4     4       0.00741      5.79
##  5     5       0.0208       7.82
##  6     6       0.0268       6.99
##  7     7       0.0289      14.3
##  8     8       0.0817      21.8
##  9     9       0.0546      14.6
## 10    10       0.0477      18.3
## # ... with 21 more rows
```

```
# analysis on group by calendar day

canceled_delayed <-
  flights %>%
  mutate(canceled = (is.na(arr_delay) | is.na(dep_delay))) %>%
  group_by(year, month, day) %>%
  summarise(prop_canceled = mean(canceled),
            avg_dep_delay = mean(dep_delay, na.rm = TRUE))

ggplot(canceled_delayed, aes(x = avg_dep_delay, prop_canceled)) +
  geom_point() +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess'
```



QUESTION 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 5:

```r
# worst carrier
flights %>%
  group_by(carrier) %>%
  summarize(mean_delay = mean(arr_delay, na.rm = TRUE)) %>%
  arrange(desc(mean_delay))
```

```
## # A tibble: 16 x 2
##    carrier mean_delay
##    <chr>        <dbl>
##  1 F9          21.9
##  2 FL          20.1
##  3 EV          15.8
##  4 YV          15.6
##  5 OO          11.9
##  6 MQ          10.8
##  7 WN           9.65
##  8 B6           9.46
##  9 9E           7.38
## 10 UA           3.56
## 11 US           2.13
```

```
## 12 VX            1.76
## 13 DL            1.64
## 14 AA            0.364
## 15 HA          - 6.92
## 16 AS          - 9.93
```

From the above result, F9 is the worst carrier.

```r
# worst carrier
flights %>%
  group_by(carrier) %>%
  summarize(mean_delay = mean(arr_delay, na.rm = TRUE)) %>%
  arrange(desc(mean_delay))
```

```
## # A tibble: 16 x 2
##    carrier mean_delay
##    <chr>        <dbl>
##  1 F9          21.9
##  2 FL          20.1
##  3 EV          15.8
##  4 YV          15.6
##  5 OO          11.9
##  6 MQ          10.8
##  7 WN           9.65
##  8 B6           9.46
##  9 9E           7.38
## 10 UA           3.56
## 11 US           2.13
## 12 VX           1.76
## 13 DL           1.64
## 14 AA           0.364
## 15 HA          - 6.92
## 16 AS          - 9.93
```

```r
# challenge: bad airports vs. bad carriers
flights %>%
  group_by(carrier, dest) %>%
  summarize(mean_delay = mean(arr_delay, na.rm = TRUE)) %>%
  group_by(carrier) %>%
  summarize(mean_delay_mad = mad(mean_delay, na.rm = TRUE)) %>%
  arrange(desc(mean_delay_mad))
```

```
## # A tibble: 16 x 2
##    carrier mean_delay_mad
##    <chr>            <dbl>
##  1 VX               12.4
##  2 OO               10.5
##  3 YV                8.97
##  4 9E                8.20
##  5 EV                7.09
##  6 DL                7.00
##  7 UA                5.04
##  8 US                5.03
##  9 B6                5.00
## 10 WN                4.51
## 11 AA                3.31
```

```
## 12 MQ                   2.88
## 13 FL                   1.55
## 14 AS                   0
## 15 F9                   0
## 16 HA                   0
```

For the challenge, I calculated the median absolute deviation of average arrival delay by carrier and destination. Higher values indicate a larger spread in the average delays across destinations, meaning these carriers experienced more variation in average delays - for some destinations these carriers experienced longer delays, whereas some destinations arrivals were closer to on time. Lower values mean the carrier experienced similar delays across destinations. It does not mean these carriers were on time. It means tht they were more consistent. Comparing this table to the first table of average arrival delays could disentangle the effect of bad carriers vs. bad airports.

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

The sort argument will sort the results of count() in descending order of n. We use this if we plan to arrange() the results after completing the count.

10.5 Exercises

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

ANSWER 1:

```r
# data frame
print(mtcars)
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
```

```
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```
# tibble
print(as_tibble(mtcars))
```

```
## # A tibble: 32 x 11
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##  * <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1  21.0  6.00   160 110    3.90  2.62  16.5  0     1.00  4.00  4.00
##  2  21.0  6.00   160 110    3.90  2.88  17.0  0     1.00  4.00  4.00
##  3  22.8  4.00   108  93.0  3.85  2.32  18.6  1.00  1.00  4.00  1.00
##  4  21.4  6.00   258 110    3.08  3.22  19.4  1.00  0     3.00  1.00
##  5  18.7  8.00   360 175    3.15  3.44  17.0  0     0     3.00  2.00
##  6  18.1  6.00   225 105    2.76  3.46  20.2  1.00  0     3.00  1.00
##  7  14.3  8.00   360 245    3.21  3.57  15.8  0     0     3.00  4.00
##  8  24.4  4.00   147  62.0  3.69  3.19  20.0  1.00  0     4.00  2.00
##  9  22.8  4.00   141  95.0  3.92  3.15  22.9  1.00  0     4.00  2.00
## 10  19.2  6.00   168 123    3.92  3.44  18.3  1.00  0     4.00  4.00
## # ... with 22 more rows
```

A data frame will print the entire contents. A tibble will only print (by default) the first 10 rows and as many columns as will fit in the console.

QUESTION 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?

ANSWER 2:

As data frame

```
# data frame
df <- data.frame(abc = 1, xyz = "a")
df$x
```

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

```
df[, "xyz"]
```

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

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

```
##   abc xyz
## 1   1   a
```

As tibble

```
# tibble
df <- tibble(abc = 1, xyz = "a")
```

```r
df$x
```

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

```
## NULL
```

```r
df[, "xyz"]
```

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

```r
df[, c("abc", "xyz")]
```

```
## # A tibble: 1 x 2
##     abc xyz
##   <dbl> <chr>
## 1  1.00 a
```

Here are two observations:

- Tibbles never do partial matching; data frames do.
- Subsetting tibbles using [[will always return a tibble; subsetting data frames using [[can potentially return a vector.

QUESTION 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 3:

```r
# For an example, let's store hwy in a variable and extract from mpg tibble.
var <- "hwy"
mpg[[var]]
```

```
##   [1] 29 29 31 30 26 26 27 26 25 28 27 25 25 25 25 24 25 23 20 15 20 17 17
##  [24] 26 23 26 25 24 19 14 15 17 27 30 26 29 26 24 24 22 22 24 24 17 22 21
##  [47] 23 23 19 18 17 17 19 19 12 17 15 17 17 12 17 16 18 15 16 12 17 17 16
##  [70] 12 15 16 17 15 17 17 18 17 19 17 19 19 17 17 17 16 16 17 15 17 26 25
##  [93] 26 24 21 22 23 22 20 33 32 32 29 32 34 36 36 29 26 27 30 31 26 26 28
## [116] 26 29 28 27 24 24 24 22 19 20 17 12 19 18 14 15 18 18 15 17 16 18 17
## [139] 19 19 17 29 27 31 32 27 26 26 25 25 17 17 20 18 26 26 27 28 25 25 24
## [162] 27 25 26 23 26 26 26 26 25 27 25 27 20 20 19 17 20 17 29 27 31 31 26
## [185] 26 28 27 29 31 31 26 26 27 30 33 35 37 35 15 18 20 20 22 17 19 18 20
## [208] 29 26 29 29 24 44 29 26 29 29 29 29 23 24 44 41 29 26 28 29 29 29 28
## [231] 29 26 26 26
```

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

ANSWER 6: *options(tibble.max_extra_cols = n)* is used for that. If the number of columns can not fit in the console, in that case it is useful (googled it).

12.3.3 Exercises

QUESTION 2: Why does this code fail?

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

ANSWER 2: The columns to gather are specified with dplyr::select() style notation. The columns "1999" and "2000" are non-syntactic names (because they don't start with a letter) so we have to surround them in backticks.The error message is:

Quitting from lines 206-211 (homework_4_das_sanatan.Rmd) Error in inds_combine(.vars, ind_list) : Position must be between 0 and n Calls: … gather.data.frame -> unname -> -> inds_combine Execution halted

The correct way of doing it as below.

```
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
```

QUESTION 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 3:

```
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
)

# Take a glimpse at people

glimpse(people)
```

```
## Observations: 5
## Variables: 3
## $ name  <chr> "Phillip Woods", "Phillip Woods", "Phillip Woods", "Jess...
## $ key   <chr> "age", "height", "age", "age", "height"
## $ value <dbl> 45, 186, 50, 37, 156
```

```
#spread(people, key, value)
```

If we run spread, we see the below error message.

Error: Duplicate identifiers for rows (1, 3) Execution halted

Spreading the data frame fails because there are two rows with "age" for "Phillip Woods". We would need to add another column with an indicator for the number observation it is as below.

```r
people <- tribble(
  ~name,             ~key,    ~value, ~obs,
  #-----------------|--------|------|------
  "Phillip Woods",   "age",       45, 1,
  "Phillip Woods",   "height",   186, 1,
  "Phillip Woods",   "age",       50, 2,
  "Jessica Cordero", "age",       37, 1,
  "Jessica Cordero", "height",   156, 1
)

spread(people, key, value)
```

```
## # A tibble: 3 x 4
##   name              obs   age height
##   <chr>           <dbl> <dbl>  <dbl>
## 1 Jessica Cordero  1.00  37.0    156
## 2 Phillip Woods    1.00  45.0    186
## 3 Phillip Woods    2.00  50.0     NA
```

QUESTION 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 4:

We have to gather it. The variables can be:

- pregnant: logical ("yes", "no")
- female: logical
- count: integer

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

gather(preg, sex, count, male, female) %>%
  mutate(pregnant = pregnant == "yes",
         female = sex == "female") %>%
  select(-sex)
```

```
## # A tibble: 4 x 3
##   pregnant count female
##   <lgl>    <dbl> <lgl>
## 1 T         NA   F
## 2 F         20.0 F
## 3 T         10.0 T
## 4 F         12.0 T
```

It makes easier if we convert the pregnant and female from character vectors to logical.

12.4.3 Exercises

QUESTION 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 1:

```r
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
```

```r
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

If sep is a character vector, this controls what happens when there are too many pieces. 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

fill

If sep is a character vector, this 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

By default separate drops the extra values with a warning.

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

```r
# This produces the same result as above, dropping extra values, but without the warning.

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
```

In this, the extra values are not split, so "f,g" appears in column three.

In this, one of the entries for column, "d,e", has too few elements. The default for fill is similar to separate; it fills with missing values but emits a warning. In this, row 2 of column "three", is NA.

```r
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
```

Alternative options for fill are "right", to fill with missing values from the right, but without a warning

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

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

The option fill = "left" also fills with missing values without a warning, but this time from the left side. Now, column "one" of row 2 will be missing, and the other values in that row are shifted over.

```r
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
```

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

ANSWER 2:

remove argument

If TRUE, remove input columns from output data frame.

We would set it to FALSE if we want to create a new variable, but keep the old one.

Additional Questions

Import baby_names.txt and glimpse it

```
# Read from baby_names.txt file
baby_names <- read_csv("C:/view/opt/apps/git/R/compscix-415-2-assignments/baby_names.txt")
```

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

```
# glimpse baby_names
glimpse(baby_names)
```

```
## Observations: 30,000
## Variables: 1
## $ `year|sex|name|n|prop` <chr> "1880|F|Mary|7065|0.0723843285111266", ...
```

```
# Write to baby_names.rds
saveRDS(baby_names, "baby_names.rds")

# load baby_names.rds
new_baby_names = readRDS("C:/view/opt/apps/git/R/compscix-415-2-assignments/baby_names.rds")

# glimpse new_baby_names
glimpse(new_baby_names)
```

```
## Observations: 30,000
## Variables: 1
## $ `year|sex|name|n|prop` <chr> "1880|F|Mary|7065|0.0723843285111266", ...
```

END OF HW4 ASSIGNMENT