HW2_ahcooper

Andrew Cooper

9/15/2020

```
library(readr)
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------------
## v ggplot2 3.3.2     v dplyr   1.0.2
## v tibble  3.0.3     v stringr 1.4.0
## v tidyr   1.1.2     v forcats 0.5.0
## v purrr   0.3.4
## -- Conflicts ------------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(knitr)
```

Problem 3

I aim to make version control an integral and necessary component of my future career as a statistician/data analyst. After programming for about 6 years I have realized how much time I have dedicated to sorting through old versions of code or struggling to find the most up-to-date version of a project. I have also become aware of how often I duplicate the same work for multiple projects. I plan on using version control to keep a continuously up-to-date system of all my projects so that I can always grab and use past code whenever I need to.

Problem 4

Part a

base R data-munging

```
# Read in raw sensory data
Sensory_raw <- read_table2("https://www2.isye.gatech.edu/~jeffwu/wuhamadabook/data/Sensory.dat",
     col_types = cols(Item = col_character()), col_names = F, skip = 2)
kable(Sensory_raw[1:5,])
```

X1	X2	X3	X4	X5	X6
1.0	4.3	4.9	3.3	5.3	4.4
4.3	4.5	4.0	5.5	3.3	NA
4.1	5.3	3.4	5.7	4.7	NA

X1	X2	X3	X4	X5	X6
2.0	6.0	5.3	4.5	5.9	4.7
4.9	6.3	4.2	5.5	4.9	NA

We see the issue with the raw data is that measurements for a given item are on multiple rows. To fix this, I take each row of measurements, re-align it such that no columns are empty, and add a new column listing the correct item for each row.

```
Sensory_values <- apply(Sensory_raw, 1, function(x){
  if(any(is.na(x))){
    return(x[!is.na(x)])
  }else{
    return(x[-1])
  }
})
Sensory <- data.frame(cbind(rep(seq(1, 10, 1), each = 3), t(Sensory_values)))
colnames(Sensory) <- c("Item", seq(1, 5, 1))
Sensory <- as_data_frame(Sensory)
```

```
## Warning: `as_data_frame()` is deprecated as of tibble 2.0.0.
## Please use `as_tibble()` instead.
## The signature and semantics have changed, see `?as_tibble`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

tidyverse data-munging

The measurements for each operator are currently spread out over multiple columns. A cleaner version would have these measurements collected into a single column, which I do using the "gather" function from tidyverse.

```
Sensory %>%
  gather(Operator, Value, -Item) %>%
  group_by(Operator) %>%
  summarize(`Average Value` = mean(Value)) %>%
  kable()
```

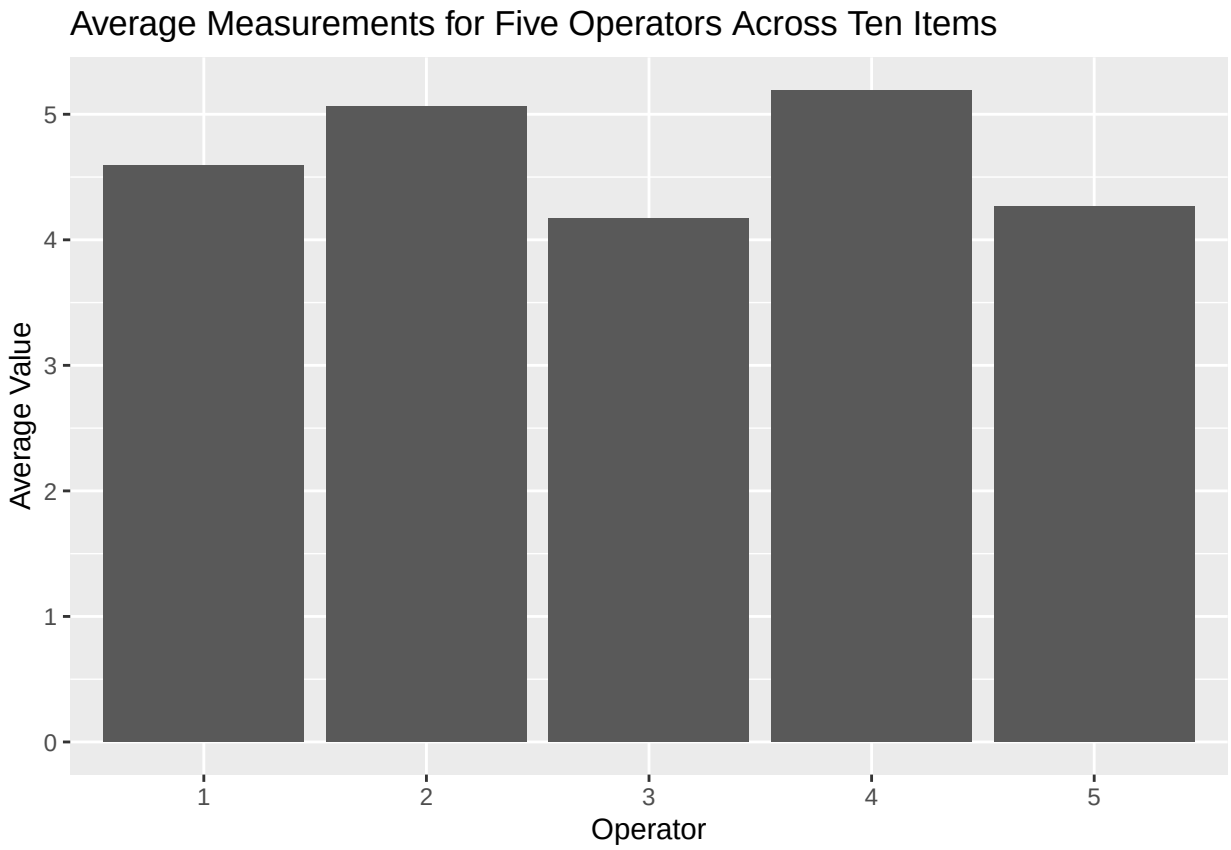
```
## `summarise()` ungrouping output (override with `.groups` argument)
```

Operator	Average Value
1	4.593333
2	5.063333
3	4.166667
4	5.193333
5	4.266667

Plot

Given the nature of the data I decided to create a bar-chart in ggplot, where my summary statistic is the average measurement across all items for a single operator.

```
## `summarise()` ungrouping output (override with `.groups` argument)
```



Part b

base R data-munging

```
Medals_raw <- read_table2("https://www2.isye.gatech.edu/~jeffwu/wuhamadabook/data/LongJumpData.dat")
kable(Medals_raw[1:5,])
```

Year	Long	Jump	Year_1	Long_1	Jump_1	Year_2	Long_2	Jump_2	Year_3	Long_3	Jump_3
-4	249.75	24	293.13	56	308.25	80	336.25	NA	NA	NA	NA
0	282.88	28	304.75	60	319.75	84	336.25	NA	NA	NA	NA
4	289.00	32	300.75	64	317.75	88	343.25	NA	NA	NA	NA
8	294.50	36	317.31	68	350.50	92	342.50	NA	NA	NA	NA
12	299.25	48	308.00	72	324.50	NA	NA	NA	NA	NA	NA

We see the problem with this raw table is the data aren't organized into columns, but follow the pattern of year and maximum long jump. To fix this, I extract the raw data and put it into a vector, then use the pattern to organize the data into two appropriate columns.

```
Medals_values <- as.numeric(c(t(Medals_raw)))
Medals_values <- Medals_values[!is.na(Medals_values)]
Medals <- data.frame(Year = Medals_values[seq(1, length(Medals_values), 2)],
                     Long_Jump = Medals_values[seq(2, length(Medals_values), 2)])
Medals <- as_data_frame(Medals)
```

tidyverse data-munging

We also need to mutate the year variable since it is currently coded. The mutation is simple; we just add 1990 to each value. We also sort the rows by year, which makes sense intuitively.

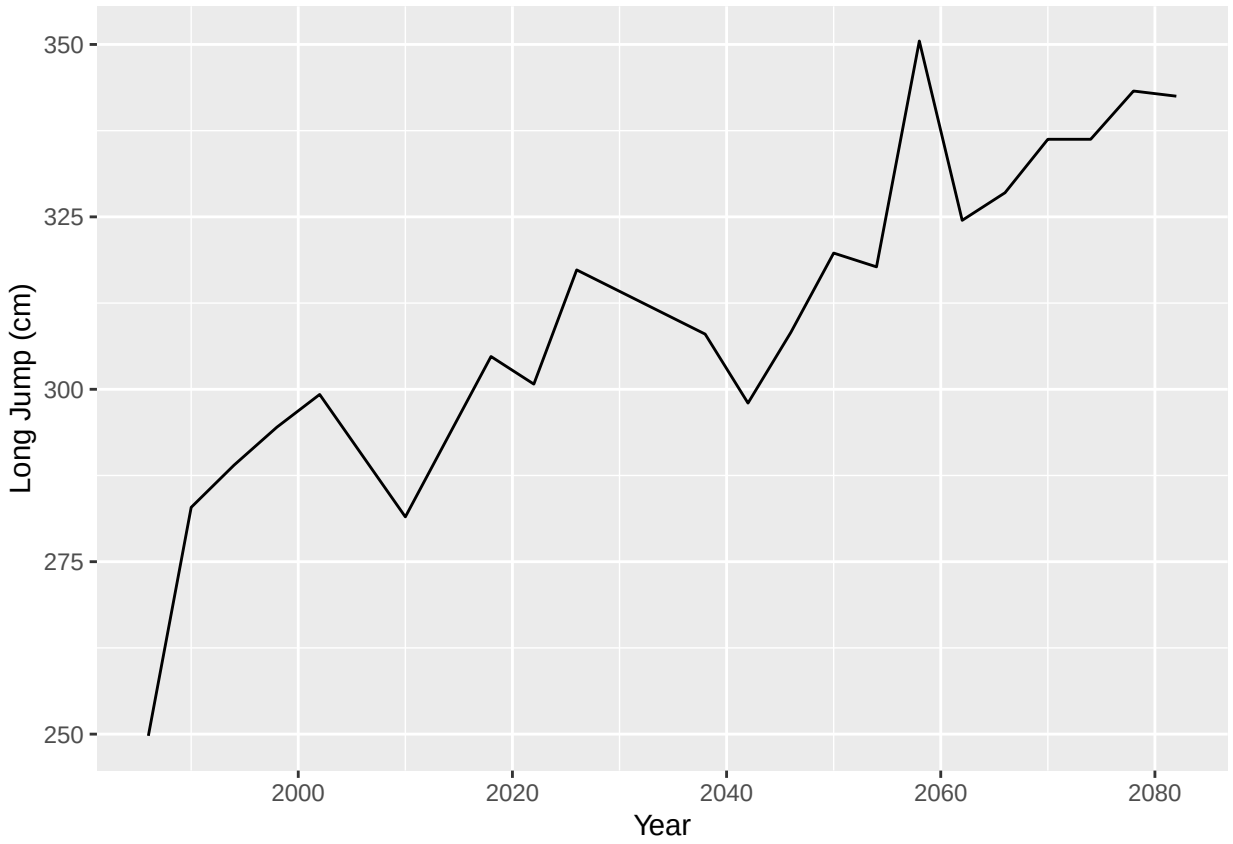
```
Medals <- Medals %>%
  mutate(Year = Year + 1990) %>%
  arrange(Year)
```

Year
Min. :1986
1st Qu.:2011
Median :2040
Mean :2035
3rd Qu.:2061
Max. :2082

Long_Jump
Min. :249.8
1st Qu.:295.4
Median :308.1
Mean :310.3
3rd Qu.:327.5
Max. :350.5

Plot

A line-plot of the long jump data seems appropriate given the time-series nature of the data.



Part c

base R data-munging

```
Body_raw <- read_table2("https://www2.isye.gatech.edu/~jeffwu/wuhamadabook/data/BrainandBodyWeight.dat")
kable(Body_raw[1:5,])
```

Body	Wt	Brain	Wt_1	Body_1	Wt_2	Brain_1	Wt_3	Body_2	Wt_4	Brain_2	Wt_5
3.385	44.5	521.000	655.0	2.50	12.1	NA	NA	NA	NA	NA	NA
0.480	15.5	0.785	3.5	55.50	175.0	NA	NA	NA	NA	NA	NA
1.350	8.1	10.000	115.0	100.00	157.0	NA	NA	NA	NA	NA	NA
465.000	423.0	3.300	25.6	52.16	440.0	NA	NA	NA	NA	NA	NA
36.330	119.5	0.200	5.0	10.55	179.5	NA	NA	NA	NA	NA	NA

We see the problem with the raw data is similar to the problem in part b; that is, the data isn't organized into proper columns but uses a pattern of "body weight", "brain weight" for each observation. Like in part b, I extracted the raw data and put it into a vector, then used the pattern to organize the data into two appropriate columns.

```
Body_values <- as.numeric(c(t(Body_raw)))
Body_values <- Body_values[!is.na(Body_values)]
Body <- data.frame(Brain_Weight = Body_values[seq(1, length(Body_values), 2)],
                   Body_Weight = Body_values[seq(2, length(Body_values), 2)])
Body <- as_data_frame(Body)
```

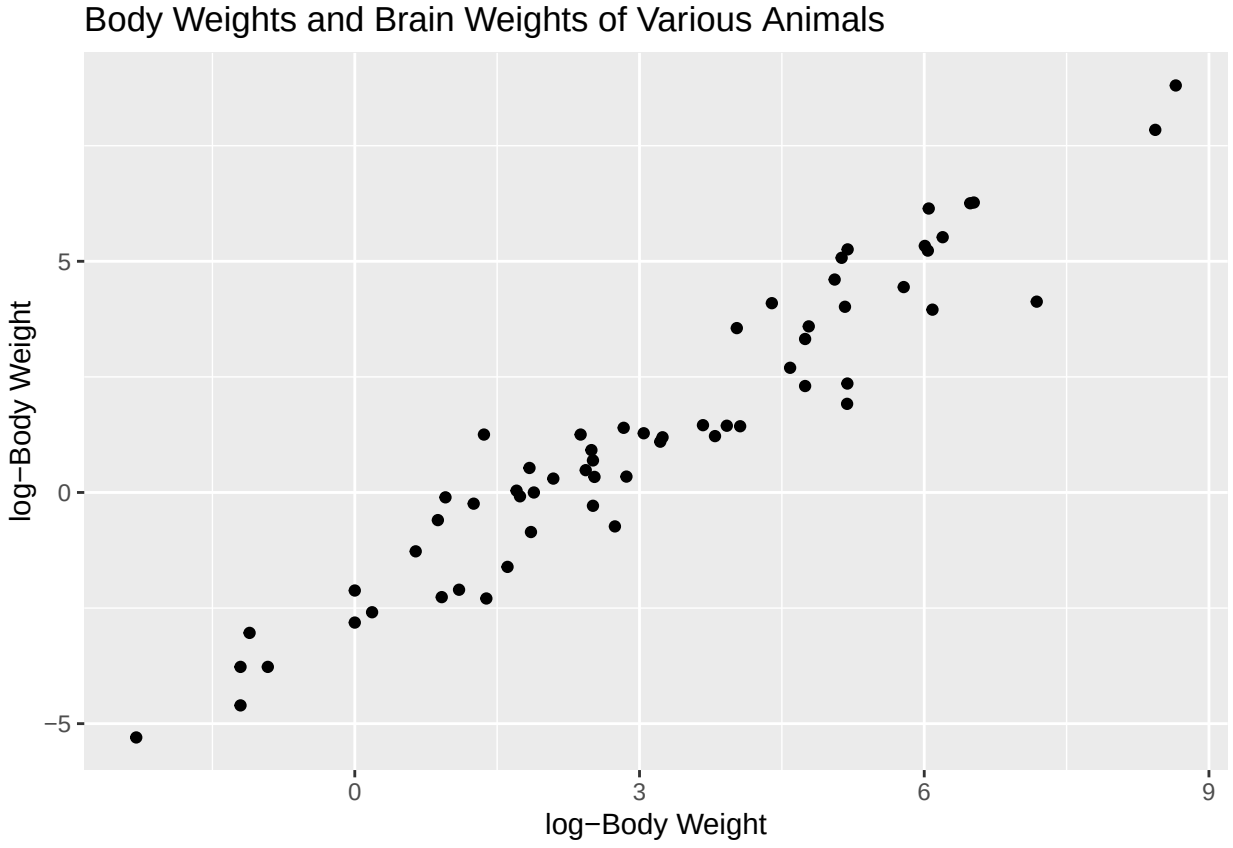
tidyverse data-munging

```
Body %>%
  summarize(`Average Body Weight` = mean(Body_Weight),
            `Average Brain Weight` = mean(Brain_Weight),
            `Body Weight SD` = sd(Body_Weight),
            `Brain Weight SD` = sd(Brain_Weight),
            `Min Body Weight` = min(Body_Weight),
            `Min Brain Weight` = min(Body_Weight),
            `Max Body Weight` = max(Body_Weight),
            `Max Brain Weight` = max(Body_Weight)) %>%
  kable()
```

Average Body Weight	Average Brain Weight	Body Weight SD	Brain Weight SD	Min Body Weight	Min Brain Weight	Max Body Weight	Max Brain Weight
283.1344	198.79	930.2789	899.158	0.1	0.1	5712	5712

Plot

Due to the extremely positive-skewed brain and body weights as we see in the summary table above, it seems appropriate to log-transform both variables in the x-y plot.



We observe a strong positive linear relationship between log-body weight and log-brain weight.

Part d

base r data-munging

```
Tomatoes_raw <- read_table2("https://www2.isye.gatech.edu/~jeffwu/wuhamadabook/data/tomato.dat", skip =
                  col_names = c("Tomato_Type", "1000", "2000", "3000"))
kable(Tomatoes_raw[1:5,])
```

Tomato_Type	1000	2000	3000
Ife#1	16.1,15.3,17.5	16.6,19.2,18.5	20.8,18.0,21.0
PusaEarlyDwarf	8.1,8.6,10.1,	12.7,13.7,11.5	14.4,15.4,13.7
NA	NA	NA	NA
NA	NA	NA	NA
NA	NA	NA	NA

We see the problem with the raw table is that multiple observations are within columns in a comma-delimited format. To fix this, I use apply to iterate through each row and split up these values accordingly.

```
Tomatoes <- data.frame(Tomato_Type = rep(Tomatoes_raw$Tomato_Type, each = 3),
                       apply(Tomatoes_raw[,-1], 2, function(x){as.numeric(unlist(strsplit(x, ",")))}))
Tomatoes <- as_data_frame(Tomatoes)
```

tidyverse data-munging

Currently all the yield measurements for different densities are spread out across multiple columns. It would be tidier to gather these into a single column, which I do in tidyverse.

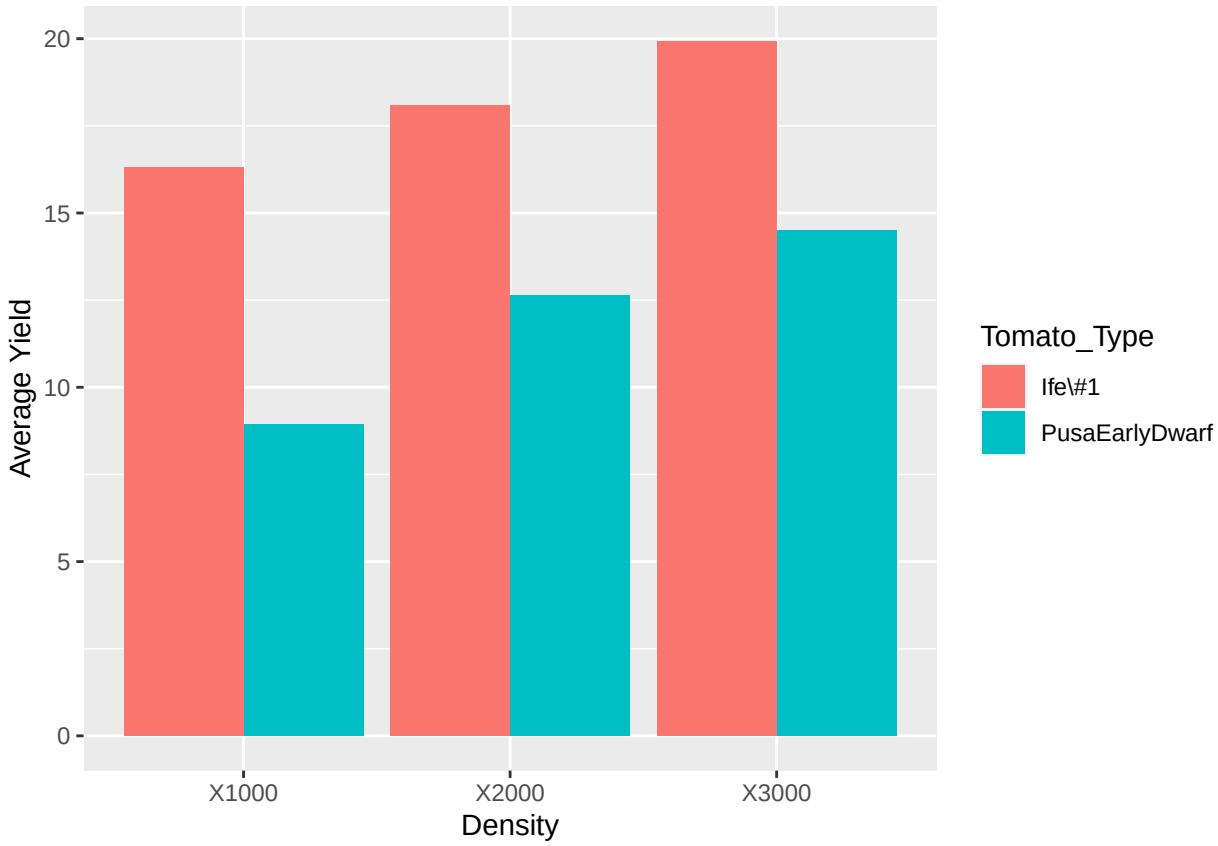
```
Tomatoes <- Tomatoes %>%
  gather(Density, Yield, -Tomato_Type)
```

```
## `summarise()` regrouping output by 'Tomato_Type' (override with `.groups` argument)
```

Tomato_Type	Density	Avg_Yield
Ife#1	X1000	16.300000
Ife#1	X2000	18.100000
Ife#1	X3000	19.933333
PusaEarlyDwarf	X1000	8.933333
PusaEarlyDwarf	X2000	12.633333
PusaEarlyDwarf	X3000	14.500000

Plot

Barplots again seem appropriate here to show the average measured yield for each density and tomato type.


20
15
10
5
0
Average Yield
X1000
X2000
X3000
Density
Tomato_Type
Ife\#1
PusaEarlyDwarf

Problem 5