Dplyr workshop

Converting to tibbles

Converting a data frame to a tibble is very simple. Just use as_tibble()!

```
as_tibble(acs12)
```

```
## # A tibble: 2,000 x 13
##    income employment hrs_work race    age gender citizen time_to_work lang
##     <int> <fct>         <int> <fct> <int> <fct>  <fct>          <int> <fct>
##  1  60000 not in la~       40 white    68 female yes               NA engl~
##  2      0 not in la~       NA white    88 male   yes               NA engl~
##  3     NA <NA>             NA white    12 female yes               NA engl~
##  4      0 not in la~       NA white    17 male   yes               NA other
##  5      0 not in la~       NA white    77 female yes               NA other
##  6   1700 employed         40 other    35 female yes               15 other
##  7     NA <NA>             NA white    11 male   yes               NA engl~
##  8     NA <NA>             NA other     7 male   yes               NA engl~
##  9     NA <NA>             NA asian     6 male   yes               NA other
## 10  45000 employed         84 white    27 male   yes               40 engl~
## # ... with 1,990 more rows, and 4 more variables: married <fct>,
## #   edu <fct>, disability <fct>, birth_qrtr <fct>
```

Select

Let's start with the acs12 dataset from the openintro package, which has data from the 2012 US census. Suppose that we want to subset the acs12 to only include four variables: income, gender, edu, and age. Suppose that we also only want to keep complete cases of those variables.

```
#before, we might have done this by using square brackets and na.omit separately
newdata1 = acs12[,c("income","gender","edu", "age")]
newdata2 = na.omit(newdata1)

#now, we can use select (and pipe) to simplify and make this easier to read
newdata = acs12 %>% select(income, gender, edu, age) %>% na.omit()
```

Mutate

Next, we can use mutate() to create new variables that are functions of old variables. For example, suppose that we want to record income in thousands of dollars (instead of in dollars). We can create a new variable called income_thousands using the mutate() function:

```
newdata = newdata %>% mutate(income_thousands = income/1000)
```

Filter

The filter function works in much the same way as subset, except we can reference variable names directly, which makes it look much cleaner. For example, suppose that we want to filter our dataset to only include females who make more than 50 thousand dollars per year.

```r
#calculate the number of people who are female with income less than $50,000
newdata %>% filter(income_thousands<50 & gender=="female") %>% nrow()
```

```
## [1] 734
```

```r
#or, you can use filter() to create a new dataset with specific rows
newdata %>% filter(income_thousands<50 & gender=="female") %>% as_tibble()
```

```
## # A tibble: 734 x 5
##    income gender edu           age income_thousands
##     <int> <fct>  <fct>       <int>            <dbl>
##  1      0 female hs or lower    77              0  
##  2   1700 female hs or lower    35              1.7
##  3   8600 female hs or lower    69              8.6
##  4   4000 female hs or lower    67              4  
##  5  19000 female college        36             19  
##  6   1200 female hs or lower    18              1.2
##  7      0 female college        31              0  
##  8  12000 female hs or lower    32             12  
##  9      0 female grad           37              0  
## 10      0 female hs or lower    47              0  
## # ... with 724 more rows
```

Summarise

The summarise() function does exactly what it sounds like: it allows you to summarize the data by reducing many values down to a single value. For example:

```r
#calculate mean income in the dataset
newdata %>% summarise(mean_salary = mean(income_thousands))
```

```
##   mean_salary
## 1    23.59998
```

This becomes more useful when combined with group_by, which allows you to calculate summary statistics for specific subgroups of the dataset:

```r
#calculate mean income by gender dataset
newdata %>%
  group_by(gender) %>%
  summarise(mean_salary = mean(income_thousands))
```

```
## # A tibble: 2 x 2
##   gender mean_salary
##   <fct>        <dbl>
## 1 male          32.6
## 2 female        14.3
```

```r
#calculate number of people in each gender x education level group with income>$50,000
newdata %>% group_by(gender, edu) %>%
  filter(income_thousands>50) %>%
  summarise(Number = n())
```

```
## # A tibble: 6 x 3
## # Groups:   gender [2]
##   gender edu         Number
```

```
##   <fct>  <fct>        <int>
## 1 male   hs or lower     66
## 2 male   college         69
## 3 male   grad            37
## 4 female hs or lower     15
## 5 female college         25
## 6 female grad            19
```

```r
#calculate mean income by gender and education level
newdata %>%
  group_by(gender, edu) %>%
  summarise(mean_salary = mean(income_thousands))
```

```
## # A tibble: 6 x 3
## # Groups:   gender [2]
##   gender edu         mean_salary
##   <fct>  <fct>             <dbl>
## 1 male   hs or lower       19.4
## 2 male   college           48.7
## 3 male   grad              91.4
## 4 female hs or lower        8.69
## 5 female college           22.6
## 6 female grad              39.4
```

Arrange

Finally, we can use the arrange function to arrange rows in a particular order. For example, we could arrange mean salaries by gender and education level in ascending or descending order with respect to a particular variable:

```r
#ascending order
newdata %>%
  group_by(gender, edu) %>%
  summarise(mean_salary = mean(income_thousands)) %>%
  arrange(mean_salary)
```

```
## # A tibble: 6 x 3
## # Groups:   gender [2]
##   gender edu         mean_salary
##   <fct>  <fct>             <dbl>
## 1 female hs or lower        8.69
## 2 male   hs or lower       19.4
## 3 female college           22.6
## 4 female grad              39.4
## 5 male   college           48.7
## 6 male   grad              91.4
```

```r
#descending order
newdata %>%
  group_by(gender, edu) %>%
  summarise(mean_salary = mean(income_thousands)) %>%
  arrange(desc(mean_salary))
```

```
## # A tibble: 6 x 3
## # Groups:   gender [2]
```

```
##   gender edu         mean_salary
##   <fct>  <fct>             <dbl>
## 1 male   grad              91.4
## 2 male   college           48.7
## 3 female grad              39.4
## 4 female college           22.6
## 5 male   hs or lower       19.4
## 6 female hs or lower        8.69
```

Joins

Dplyr offers a number of different types of joins. There are many online resources to learn more about this, so for the purposes of this tutorial, we will look at one example. Here is one potential resource: https://www.guru99.com/r-dplyr-tutorial.html

Suppose that we want to create a new variable in our dataset called mean_income_agegrp. For each person in the data, this variable tells us the mean earnings of all people who are the same age (in the data). We can do this as follows:

```r
#create a data frame with mean income by age
newdata3 = newdata %>%
  mutate(agef = as.factor(age)) %>%
  group_by(agef) %>%
  summarise(mean_income_agegrp=mean(income)) %>%
  mutate(age=as.numeric(as.character(agef))) %>%
  select(age, mean_income_agegrp)

#look at the first few rows
newdata3[1:3,]
```

```
## # A tibble: 3 x 2
##     age mean_income_agegrp
##   <dbl>              <dbl>
## 1    15                  0
## 2    16                735
## 3    17                325
```

```r
#now let's use a left join to join these values back onto the original dataset by age
newdata = dplyr::left_join(newdata, newdata3, by = "age")

#look at first few rows
newdata[1:3,]
```

```
##   income gender         edu age income_thousands mean_income_agegrp
## 1  60000 female     college  68               60           15705.36
## 2      0   male hs or lower  88                0               0.00
## 3      0   male hs or lower  17                0             325.00
```

Putting it all together

The dplyr functions are most useful in combination with each other. Here are some examples: Let's start with the original acs12 dataset and try to answer some questions:

It's first useful to look at the structure of the dataset:

```r
#str(acs12)
```

1. What is the mean commute time of people who are at least 25 years old and employed, broken down by gender and race subcategories? Follow-up: report commute times in order from shortest to longest. (note: I'm selecting columns and using na.omit first, but you could also include an na.omit parameter in the mean() function and keep all the data):

```r
acs12 %>%
  select(age, gender, race, employment, time_to_work) %>%
  na.omit() %>%
  filter(age >= 25 & employment=="employed") %>%
  group_by(gender, race) %>%
  summarise(mean_time_to_work = mean(time_to_work)) %>%
  arrange(mean_time_to_work)
```

```
## # A tibble: 8 x 3
## # Groups:   gender [2]
##   gender race  mean_time_to_work
##   <fct>  <fct>             <dbl>
## 1 female asian              18.1
## 2 female other              21.6
## 3 female white              24.9
## 4 male   other              26
## 5 male   white              27.7
## 6 male   black              28.0
## 7 female black              32.2
## 8 male   asian              38.4
```

2. What is the mean hourly wage of US citizens by gender (note: there are 52 weeks in a year)?

```r
acs12 %>%
  select(gender, citizen, income, hrs_work) %>%
  na.omit() %>%
  filter(citizen == "yes") %>%
  mutate(weekly_wage = income/52) %>%
  mutate(hourly_wage = weekly_wage/hrs_work) %>%
  group_by(gender) %>%
  summarise(mean_hourly_wage = mean(hourly_wage))
```

```
## # A tibble: 2 x 2
##   gender mean_hourly_wage
##   <fct>             <dbl>
## 1 male               22.9
## 2 female             13.4
```

Practice

We'll use the run10 dataset (sorry for those that have used this a bunch!) from the openintro package:

```
data("run10")
```

1. Start by looking at the structure of the dataset (using str() and/or by typing ?run10 into the Console to get a sense of the available variables).

2. Create a new dataset called run10_2 which only includes the following variables: time, pace, age, gender, and state.
   Use this new dataset for the rest of the questions below:

3. Create a new variable called fivek_split which gives each runner's approximate 5k time. Note that this race is 10 miles, and a 5k is 3.10686 miles.

4. Now, calculate mean 5k split times for each gender group

5. Create a new variable called decade which gives the decade of each person's age. For example, everyone in their 30s would have decade=3, everyone in their 40s would have decade=4, etc. Make this variable a factor variable. Hint: the floor() function might be helfpul to you.

6. Using this new variable, calculate mean pace for females from DC by decade. Which decades have the fastest and slowest mean paces?

7. List all of the state names in the dataset in order from fastest to slowest average finishing time

8. What states are the top 10 male runners from? What states are the top 10 female runners from?

9. Create a new variable called time_hrs, which gives finishing time in terms of hours. Then print median finishing times in hours for each decade group.

10. Create a new dataset called state_data which just has the variable "state" and a new variable called number_from_state which counts the number of people from that state.

11. Now, use a join to append the n_from_state column onto the run10_2 dataset so that everyone in the run10_2 dataset now also has a value for n_from_state (which gives the number of people who ran the race who were from the same state as them)

12. Filter this new dataset so that you only include people from states that have between 50-200 (inclusive) runners from that state. Use this new dataset to calculate mean finishing times (in minutes) for each state. Which of these states had the fastest and slowest finishing times on average?