Data Wrangling in R

Learning Objectives

1. View specific row(s) and/or column(s) of a data frame
2. Select observations by condition(s)
3. Change column name(s)
4. Find and remove missing data
5. Summarizing variable(s)
6. Summarizing variable by groups
7. Create a new variable based on existing variable(s)
8. Combine data frames and vectors
9. Export data frame in R to a .csv file
10. Sort data frame by column values

We will use the dataset ClassDataPrevious.csv as a working example. The dataset was collected from an introductory statistics class at UVa from a previous semester. Download the dataset from Collab and read it into R

```
Data<-read.csv("ClassDataPrevious.csv", header=TRUE)
dim(Data)
```

```
## [1] 298   8
```

```
colnames(Data)
```

```
## [1] "Year"     "Sleep"    "Sport"    "Courses"  "Major"    "Age"      "Computer"
## [8] "Lunch"
```

There are 298 rows (observations) and 8 columns (variables) in the data frame. The variables are:

1. `Year`: the year the student is in
2. `Sleep`: how much sleep the student averages a night (in hours)
3. `Sport`: the student's favorite sport
4. `Courses`: how many courses the student is taking in the semester
5. `Major`: the student's major
6. `Age`: the student's age (in years)
7. `Computer`: the operating system the student uses (Mac or PC)
8. `Lunch`: how much the student usually spends on lunch (in dollars)

1. View specific row(s) and/or column(s) of a data frame

We can view specific rows and/or columns of any data frame using the square brackets `[]`, for example

```
Data[1,2] ##row index first, then column index
```

```
## [1] 8
```

The row index is listed first, then the column index, in the square brackets. This means the first student sleeps 8 hours a night. We can also view multiple rows and columns, for example

```
Data[c(1,3,4),c(1,5,8)]
```

```
##     Year                              Major Lunch
## 1 Second                           Commerce    11
## 3 Second Cognitive science and psychology    10
## 4  First                           Pre-Comm     4
```

to view the 1st, 5th, and 8th variables for observations 1, 3, and 4.

There are several ways to view a specific column. For example, the view the 1st column (which is the variable called `Year`)

```
Data$Year ##or
Data[,1] ##or
Data[,-c(2:8)]
```

Note the comma separates the indices for the row and column. An empty value before the comma means we want all the rows, and then the specific column. To view multiple columns, for example the first four columns

```
Data[,1:4]
Data[,c(1,2,3,4)]
```

To view the values of certain rows, we can use

```
Data[c(1,3),]
```

to view the values for observations 1 and 3. An empty value after the comma means we want all the columns for those specific rows.

2. Select observations by condition(s)

We may want to only analyze certain subsets of our data, based on some conditions. For example, we only want to analyze students whose favorite sport is soccer. The `which()` function in R helps us find the indices associated with a condition being met. For example

```
which(Data$Sport=="Soccer")
```

```
##  [1]   3  20  25  26  31  32  33  38  44  46  48  50  51  64  67  71  87  92  98
## [20]  99 118 122 124 126 128 133 136 137 143 146 153 159 165 174 197 198 207 211
## [39] 214 226 234 241 255 259 260 266 274 278 281 283 294 295
```

informs us which rows belong to observations whose favorite sport is soccer, i.e. the 3rd, 20th, 25th (and so on) students. We can create a new data frame that contains only students whose favorite sport is soccer.

```
SoccerPeeps<-Data[which(Data$Sport=="Soccer"),]
dim(SoccerPeeps)
```

```
## [1] 52  8
```

We are extracting the rows which satisfy the condition, favorite sport being soccer, and storing these rows into a new data frame called `SoccerPeeps`. We can see that this new data frame has 52 observations.

Suppose we want to have a data frame that satisfies two conditions: that the favorite sport is soccer and they are 2nd years at UVa, we can type

```
SoccerPeeps_2nd<-Data[which(Data$Sport=="Soccer" & Data$Year=="Second"),]
dim(SoccerPeeps_2nd)
```

```
## [1] 25  8
```

This new data frame `SoccerPeeps_2nd` has 25 observations.

We can also set conditions based on numeric variables, for example, we want students who sleep more than eight hours a night

```
Sleepy<-Data[which(Data$Sleep>8),]
```

We can also create a data frame that contains students who satisfy at least one out of two conditions: the favorite sport is soccer or they sleep more than 8 hours a night

```
Sleepy_or_Soccer<-Data[which(Data$Sport=="Soccer" | Data$Sleep>8),]
```

3. Change column name(s)

For some datasets, the names of the columns are complicated or do not make sense. We should always give descriptive names to columns that make sense. For this dataset, the names are self-explanatory so we do not really need to change them. As an example, suppose we want to change the name of the 7th column from `Computer` to `Comp`

```
names(Data)[7]<-"Comp"
```

To change the names of multiples columns (for example, the 1st and 7th columns), type

```
names(Data)[c(1,7)]<-c("Yr","Computer")
```

4. Find and remove missing data

There are a few ways to locate missing data. Using the `is.na()` function directly on a data frame produces a lot of output that can be messy to view.

```
is.na(Data)
```

On the other hand, using the `complete.cases()` function is more pleasing to view

```
Data[!complete.cases(Data),]
```

```
##         Yr Sleep      Sport Courses                                    Major
## 103 Second    NA Basketball       7 psychology and youth and social innovation
## 206 Second     8       None       4                          Cognitive Science
##     Age Computer Lunch
## 103  19      Mac    10
## 206  19      Mac    NA
```

The code above will extract rows that are not complete cases, in other words, rows that have missing entries. The output informs us observation 103 has a missing value for `Sleep`, and observation 206 has a missing value for `Lunch`.

If you want to remove observations with a missing value, you can use one of the following two lines of code to create new data frames with the rows with missing values removed:

```
Data_nomiss<-na.omit(Data) ##or
Data_nomiss2<-Data[complete.cases(Data),]
```

A word of caution: these lines of code will remove the entire row as long as at least a column has missing entries. As noted earlier, observation 103 has a missing value for only the `Sleep` variable. But this observation still provides information on the other variables, which are now removed.

5. Summarizing variable(s)

Very often, we want to obtain some characteristics of our data. A common way to summarize a numerical variable is to find its mean. We have four numerical variables in our data frame, which are in columns 2, 4, 6, and 8. To find the mean of all four numerical variables, we can use the `apply()` function:

```
apply(Data[,c(2,4,6,8)],2,mean)
```

```
##     Sleep   Courses       Age     Lunch
##        NA  5.016779 19.573826        NA
```

```
apply(Data[,c(2,4,6,8)],2,mean,na.rm=T)
```

```
##      Sleep    Courses        Age      Lunch
## 155.559259   5.016779  19.573826 156.594175
```

Notice that due to the missing values, the first line has `NA` for some of the variables. The second line includes an optional argument, `na.rm=T`, which will remove the observations with an `NA` value for the variable from the calculation of the mean.

There are at least 3 arguments that are supplied to the `apply()` function:

1. The first argument is an data frame containing all the variables which we want to find the mean of. In this case, we want columns 2, 4, 6, and 8 of the data frame `Data`.
2. The second argument takes on the value `1` or `2`. Since we want to find the mean of columns, rather than rows, we type `2`. If want to mean of a row, we will type `1`.
3. The third argument specifies the name of the function you want to apply to the columns of the supplied data frame. In this case, we want the mean. We can change this to find the median, standard deviation, etc, of these numeric variables if we want to.

We notice the means for some of the variables are suspiciously high, so looking at the medians will be more informative.

```
apply(Data[,c(2,4,6,8)],2,median,na.rm=T)
```

```
##   Sleep Courses     Age   Lunch
##     7.5     5.0    19.0     9.0
```

6. Summarizing variable by groups

Sometimes we want to summarize a variable by groups. Suppose we want to find the median amount of sleep separately for 1st years, 2nd years, 3rd years, and 4th years get. We can use the `tapply()` function

```
tapply(Data$Sleep,Data$Yr,median,na.rm=T)
```

```
##  First Fourth Second  Third
##    8.0    7.0    7.5    7.0
```

This informs us the median amount of sleep first years get is 8 hours a night; for fourth years the median amount is 7 hours a night.

There are at least 3 arguments that are supplied to the `tapply()` function:

1. The first argument contains the vector which we want to summarize.
2. The second argument contains the factor which we use to subset our data. In this example, we want to subset according to `Yr`.
3. The third argument is the function which we want to apply to each subset of our data.
4. The fourth argument is optional, in this case, we want to remove observations with missing values from the calculation of the mean.

Notice the output orders the factor levels in alphabetical order. For our context, it is better to rearrange the levels to First, Second, Third, Fourth using the `factor()` function:

```
Data$Yr<-factor(Data$Yr, levels=c("First","Second","Third","Fourth"))

levels(Data$Yr)
```

```
## [1] "First"  "Second" "Third"  "Fourth"
```

```
tapply(Data$Sleep,Data$Yr,median,na.rm=T) ##much nicer
```

```
##  First Second  Third Fourth
##    8.0    7.5    7.0    7.0
```

This output makes a lot more sense for this context.

If we want to summarize a variable on groups formed by more than one variable, we need to adjust the second argument in the `tapply()` function by creating a list. Suppose we want to find the median sleep hour based on the `Yr` and the preferred operating system of the observations,

```
tapply(Data$Sleep,list(Data$Yr,Data$Computer),median,na.rm=T)
```

```
##           Mac   PC
## First  NA 8.0 7.50
## Second  7 7.5 7.50
## Third  NA 7.5 7.00
## Fourth NA 7.0 7.25
```

Interestingly, it looks like there were observations who did not specify which operating system they use, hence the extra column in the output.

7. Create a new variable based on existing variable(s)

Depending on the context of our analysis, we may need to create new variables based on existing variables. There are a few variations of this task, based on the type of variable you want to create, and the type of variable it is based on.

Create a numeric variable based on another numeric variable

The variable `Sleep` is in number of hours. Suppose we need to convert the values of `Sleep` to number of minutes, we can simply perform the following mathematical operation

```
Sleep_mins<-Data$Sleep * 60
```

and store the transformed variable into a vector called `Sleep_mins`.

Create a binary variable based on a numeric variable

Suppose we want to create a binary variable (categorical variable with two levels), called `deprived`. An observation will obtain a value of "yes" if they sleep for less than 7 hours a night, and "no" otherwise. The `ifelse()` function is useful in creating binary variables

```
deprived<-ifelse(Data$Sleep<7, "yes", "no")
```

There are 3 arguments associated with the `ifelse()` function:

1. The first argument is the condition that we wish to use.
2. The second argument is the value of the observation if the condition is true.
3. The third argument is the value of the observation if the condition if false.

Create a categorical variable based on a numeric variable

Suppose we want to create a categorical variable based on the number of courses a student takes. We will call this new variable `CourseLoad`, which takes on the following values:

- `light` if 3 courses or less,
- `regular` if 4 or 5 courses,
- `heavy` if more than 5 courses .

The `cut()` function is used in this situation

```
CourseLoad<-cut(Data$Courses, breaks = c(-Inf, 3, 5, Inf),
                labels = c("light", "regular", "heavy"))
```

There are three arguments that are applied to the `cut()` function:

1. The first argument is the vector which you are basing the new variable on.
2. The argument breaks lists how you want to set the intervals associated with `Data$Courses`. In this case, we are creating three intervals: one from $(-\infty, 3]$, another from $(3, 5]$, and the last interval from $(5, \infty]$.
3. The argument labels gives the label for `CourseLoad` associated with each interval.

Collapse levels

Sometimes, a categorical variable has more levels than we need for our analysis, and we want to collapse some levels. For example, the variable `Yr` has four levels: First, Second, Third, and Fourth. Perhaps we are more interested in comparing upperclassmen and underclassmen, so we want to collapse First and Second years into underclassmen, and Third and Fourth years into upperclassmen.

```
levels(Data$Yr)
```

```
## [1] "First"  "Second" "Third"  "Fourth"
```

```
new.levels<-c("und", "und", "up","up")
Year2<-factor(new.levels[Data$Yr])
levels(Year2)
```

```
## [1] "und" "up"
```

The levels associated with the variable `Yr` are ordered as First, Second, Third, Fourth. The character vector new.levels has `under` as the first two characters, and `upper` as the last two characters to correspond to the original levels in the variable `Yr`. The new variable is called `Year2`.

8. Combine data frames

We have created four new variables, `Sleep_mins`, `deprived`, `CourseLoad`, and `Year2`, based on previously existing variables. Since these variables are all based on the same observations, we can combine them with an existing data frame using the `data.frame()` function.

```
Data<-data.frame(Data,Sleep_mins,deprived,CourseLoad,Year2)
head(Data)
```

```
##       Yr Sleep      Sport Courses                           Major Age Computer
## 1 Second     8 Basketball       6                        Commerce  19      Mac
## 2 Second     7     Tennis       5                       Psychology  19      Mac
## 3 Second     8     Soccer       5 Cognitive science and psychology  21      Mac
## 4  First     9 Basketball       5                        Pre-Comm  19      Mac
## 5 Second     4 Basketball       6                      Statistics   19       PC
## 6  Third     7       None       4                       Psychology  20       PC
##   Lunch Sleep_mins deprived CourseLoad Year2
## 1    11        480       no      heavy   und
```

```
## 2    10        420       no    regular   und
## 3    10        480       no    regular   und
## 4     4        540       no    regular   und
## 5     0        240      yes      heavy   und
## 6    11        420       no    regular    up
```

Notice that since we listed the four new variables after `Data` in the `data.frame()` function, they appear after the original columns in the data frame.

Alternatively, we can use the `cbind()` function which gives the same data frame

```
Data2<-cbind(Data,Sleep_mins,deprived,CourseLoad,Year2)
```

If you are combining data frames which have different observations but the same columns, we can merge them using `rbind()`

```
dat1<-Data[1:3,1:3]
dat3<-Data[6:8,1:3]
res.dat2<-rbind(dat1,dat3)
head(res.dat2)
```

```
##       Yr Sleep      Sport
## 1 Second     8 Basketball
## 2 Second     7     Tennis
## 3 Second     8     Soccer
## 6  Third     7       None
## 7 Second     7 Basketball
## 8  First     7 Basketball
```

9. Export data frame in R to a .csv file

To export a data frame to a .csv file, type

```
write.csv(Data, file="newdata.csv", row.names = TRUE)
```

A file newdata.csv will be created in the working directory.

10. Sort data frame by column values

To sort your data frame in ascending order by `Age`,

```
Data_by_age<-Data[order(Data$Age),]
```

To sort in descending order by `Age`,

```
Data_by_age_des<-Data[order(-Data$Age),]
```

To sort in ascending order by `Age` first, then by `Sleep`,

```
Data_by_age_sleep<-Data[order(Data$Age, Data$Sleep),]
```