N741 Lecture 5: EDA with `dplyr` and `ggplot2`

Melinda Higgins & Vicki Hertzberg

February 14, 2018

Packages needed for today's lecture:

- `car`
- `tidyverse` - mainly using `dplyr` and `ggplot2`
- `janitor` *(optional)*

Rmarkdown initial setup options

Create an initial R chunk with `knitr` options to control what is shown or not shown in your final output. Three options are useful to set to FALSE for the final version. These 3 options can be kept as TRUE when debugging initially.

- `echo` - set to TRUE to see your R code in the final document - set to TRUE for homework
- `warning` - set to FALSE so that warnings are hidden
- `message` - set to FALSE to hide messages like when loading packages and such

It is also a good idea to load the packages needed for all of the code run in this RMD (R markdown) document in this first R-code chunk.

```
# knitr options
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(warning = FALSE)
knitr::opts_chunk$set(message = FALSE)

# packages needed for this RMD
library(car)
library(tidyverse)
```

```
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr

## Conflicts with tidy packages ----------------------------------------------

## filter(): dplyr, stats
## lag():    dplyr, stats
## recode(): dplyr, car
## some():   purrr, car
```

Naming R code chunks

It is a good practice to give your R code chunks names. That way when you are debugging, it'll be easier to figure out which section of the document or R-code chunk is failing.

The R-code chunk above is called `setup`.

Working with `Leinhardt` dataset in `car` package

The `car` package has 51 datasets built into it. One of these datasets is the `Leinhardt` dataset. Type `?car::Leinhardt` at the console to see help details on this dataset. This dataset has 105 rows and 4 columns with observations from nations of the world around 1970. The 4 columns are:

- `income`
- `infant`
- `region`
- `oil`

Load the dataset and take a look at it. We can access the dataset directly using the `View()` command and by referencing the dataset using the `package::dataset` syntax.

OR we can also create a new data object using the `<-` assign command and work with this copy of the dataset in memory to work from.

```
dataLeinhardt <- car::Leinhardt
```

Dataset object

Look at the type of object this dataset is and the structure of the data - the types of variables in the dataset.

```
class(dataLeinhardt)
```

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

```
str(dataLeinhardt)
```

```
## 'data.frame':    105 obs. of  4 variables:
##  $ income: int  3426 3350 3346 4751 5029 3312 3403 5040 2009 2298 ...
##  $ infant: num  26.7 23.7 17 16.8 13.5 10.1 12.9 20.4 17.8 25.7 ...
##  $ region: Factor w/ 4 levels "Africa","Americas",..: 3 4 4 2 4 4 4 4 4 4 ...
##  $ oil   : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
```

Summary of Data

One of the easiest ways to summarize the variables in the dataset is to run the `summary()` command.

```
summary(dataLeinhardt)
```

```
##      income           infant           region    oil
##  Min.   :  50.0   Min.   :  9.60   Africa  :34   no :96
##  1st Qu.: 123.0   1st Qu.: 26.20   Americas:23   yes: 9
##  Median : 334.0   Median : 60.60   Asia    :30
##  Mean   : 998.1   Mean   : 89.05   Europe  :18
##  3rd Qu.:1191.0   3rd Qu.:129.40
##  Max.   :5596.0   Max.   :650.00
##                   NA's   :4
```

This is ok when the dataset has only a few variables. However, we will often only want summaries of some of the variables in a dataset. Let's begin using the `dplyr` package to work specific sections of the data we want.

Learn more about `dplyr` at http://dplyr.tidyverse.org/. There is also good overall information at http://stat545.com/topics.html.

Intro to `dplyr`

The `dplyr` package provides 2 key aspects of working with data in `R`:

- the pipe operator `%>%` and
- verb based functions to take action on your data

The pipe operator `%>%` actually comes from the `magrittr` package which is loaded with `dplyr`. The main benefit of the pipe operator `%>%` is that you can directly reference variables in your dataset without having to use the dollar sign `$` selector.

For example, let's make a subset of the data for countries in "Asia" using the `filter()` function from `dplyr`. The R-code below can be read as take the Leinhardt dataset and pipe it into the filter function and find only data for countries in the "Asia" region.

```
dataAsia <- dataLeinhardt %>%
  filter(region == "Asia")
```

Additionally, you can string together a series of commands by simply piping the result from the previous command into the next command. Let's add the `summary()` function to get the summary statistics for the variables but only for the Asian countries.

```
dataLeinhardt %>%
  filter(region == "Asia") %>%
  summary()
```

```
##      income           infant           region    oil
##  Min.   :  71.0   Min.   : 10.20   Africa  : 0   no :26
##  1st Qu.: 100.5   1st Qu.: 21.90   Americas: 0   yes: 4
##  Median : 245.5   Median : 50.00   Asia    :30
##  Mean   : 638.9   Mean   : 96.17   Europe  : 0
##  3rd Qu.: 539.2   3rd Qu.:112.15
##  Max.   :3723.0   Max.   :650.00
##                   NA's   :3
```

Choose variables

Let's only look at `income` and `infant` and get the summary stats. Use the `select()` function from `dplyr`.

```
dataLeinhardt %>%
  select(income, infant) %>%
  summary()
```

```
##      income           infant
##  Min.   :  50.0   Min.   :  9.60
##  1st Qu.: 123.0   1st Qu.: 26.20
##  Median : 334.0   Median : 60.60
##  Mean   : 998.1   Mean   : 89.05
##  3rd Qu.:1191.0   3rd Qu.:129.40
##  Max.   :5596.0   Max.   :650.00
##                   NA's   :4
```

Get specific statistics on the variables you want

Let's get the mean and standard devaition for the `income` variable.

```
dataLeinhardt %>%
  summarise(meanIncome = mean(income),
            sdIncome = sd(income))
```

```
##   meanIncome sdIncome
## 1   998.0667 1416.714
```

and we can get the number of observations using the `n()` command in `dplyr`.

```
dataLeinhardt %>%
  summarise(nObs = n())
```

```
##   nObs
## 1  105
```

Get summary stats

We can get summary statistics for multiple statistics and multiple variables by using the `summarise_all()` function in `dplyr`. This function is used in conjunction with the `funs()` command to list the functions you want along with any options needed.

```
dataLeinhardt %>%
  select(income, infant) %>%
  summarise_all(funs(mean, sd))
```

```
##   income_mean infant_mean income_sd infant_sd
## 1    998.0667          NA  1416.714       NaN
```

Notice the `mean` and `sd` for `infant` yields `NA` and `NaN` due to the missing data in the `infant` variable. So, we need to add the option `na.rm=TRUE` to remove missing values before computing the requested statistic.

```
# add na.rm=TRUE as option to remove
# NAs for the infant variable.
dataLeinhardt %>%
  select(income, infant) %>%
  summarise_all(funs(mean, sd), na.rm=TRUE)
```

```
##   income_mean infant_mean income_sd infant_sd
## 1    998.0667    89.04752  1416.714  90.80171
```

Complete Cases

Notice that there are 4 countries that do not have data for infant mortality - there are 4 `NA's` for the `infant` variable. Suppose we want to determine how many observations have complete data across all variables in this dataset. Let's look at the `complete.cases()` function. The result of this function is a vector of TRUE and FALSE for each row/observation that has no missing data across all columns/variables.

Then use the `table()` function to see the tally of the number of TRUE's and FALSE's.

```
completeObs <- complete.cases(dataLeinhardt)
table(completeObs)
```

```
## completeObs
## FALSE  TRUE
##     4   101
```

We can then use this vector of TRUE and FALSE to keep only the complete data. Here are 2 ways to do this.

```
dataComplete <- dataLeinhardt %>%
  na.omit()

dataComplete <- dataLeinhardt %>%
  filter(complete.cases(.))
```

Count the amount of missing data

The function `is.na()` is a good way to determine how much missing data there is for any given variable. This again creates a LOGICAL vector of TRUE's and FALSE's to indicate if data for the variable specified is missing for that observation or not.

It turns out that you can run functions like `mean()` and `sum()` on LOGICAL vectors. If you run `sum()` on this vector you get the number of missing values. If you run `mean()` you get the proportion of missing data (i.e. the count/total number of observations).

```
missingInfant <- is.na(dataLeinhardt$infant)
sum(missingInfant)
```

```
## [1] 4
```

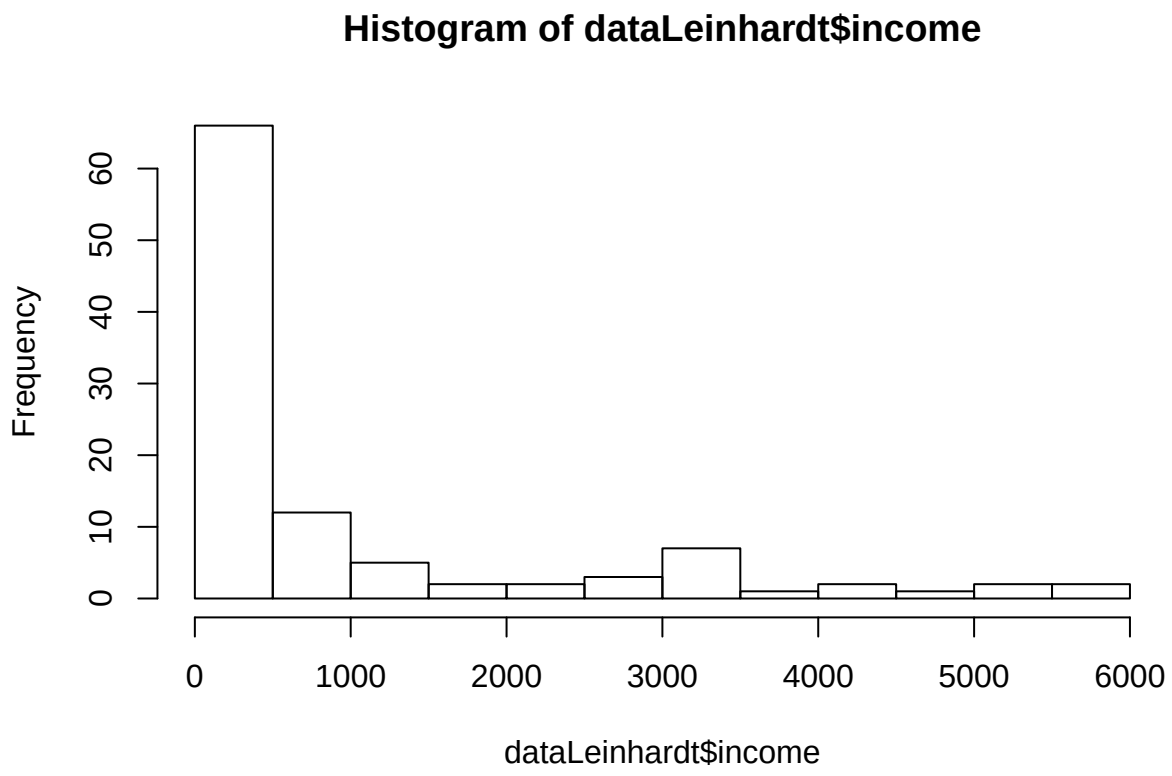
```
mean(missingInfant)
```

```
## [1] 0.03809524
```

Graphics with ggplot2

Let's look at the distributions of `income` and `infant` mortality. Here is a basic histogram of `income`.

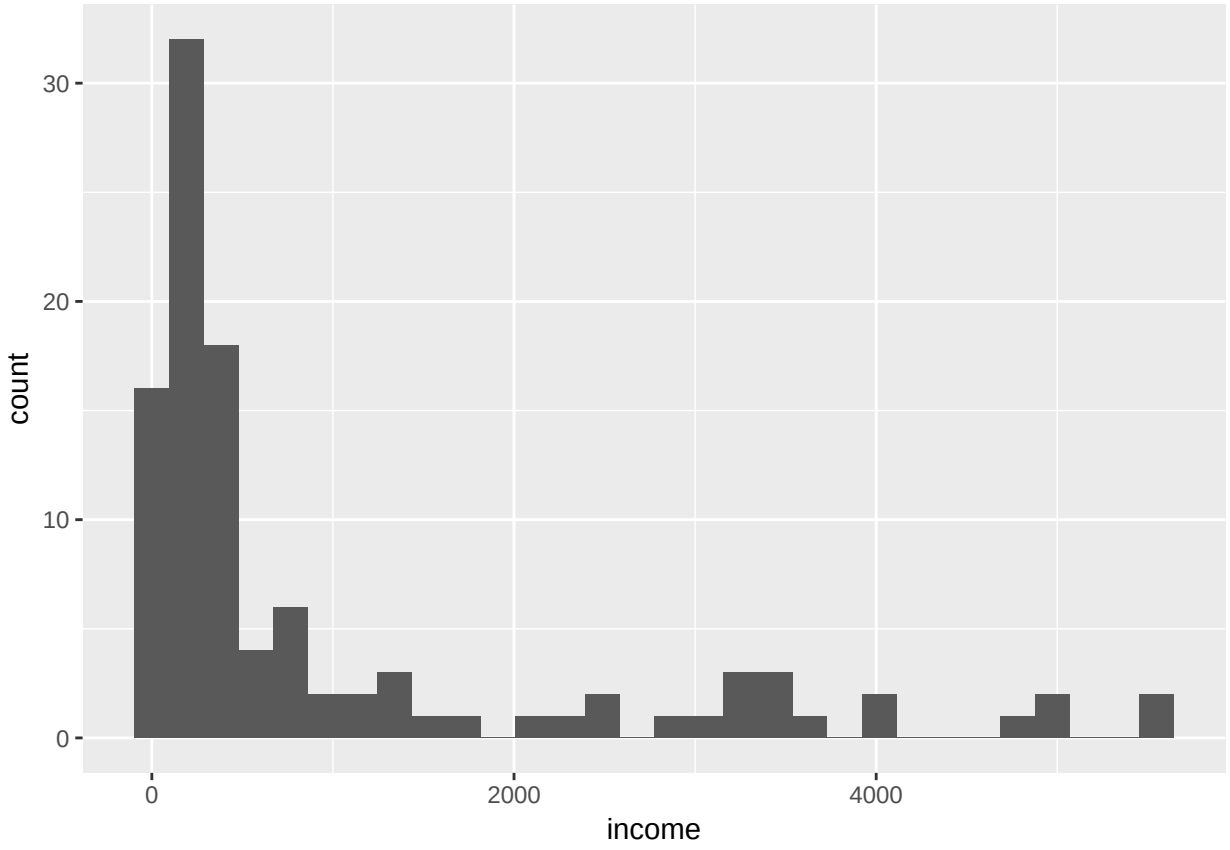
```
hist(dataLeinhardt$income)
```



Instead let's use a `ggplot2` approach. First we create a plot with `ggplot()` with the "aesthetics" we need defined. The "aesthetic" we need here is the `income` variable.

Then we add the geometric object layer - these are called `geom`'s. The one we want here is `geom_histogram()`.

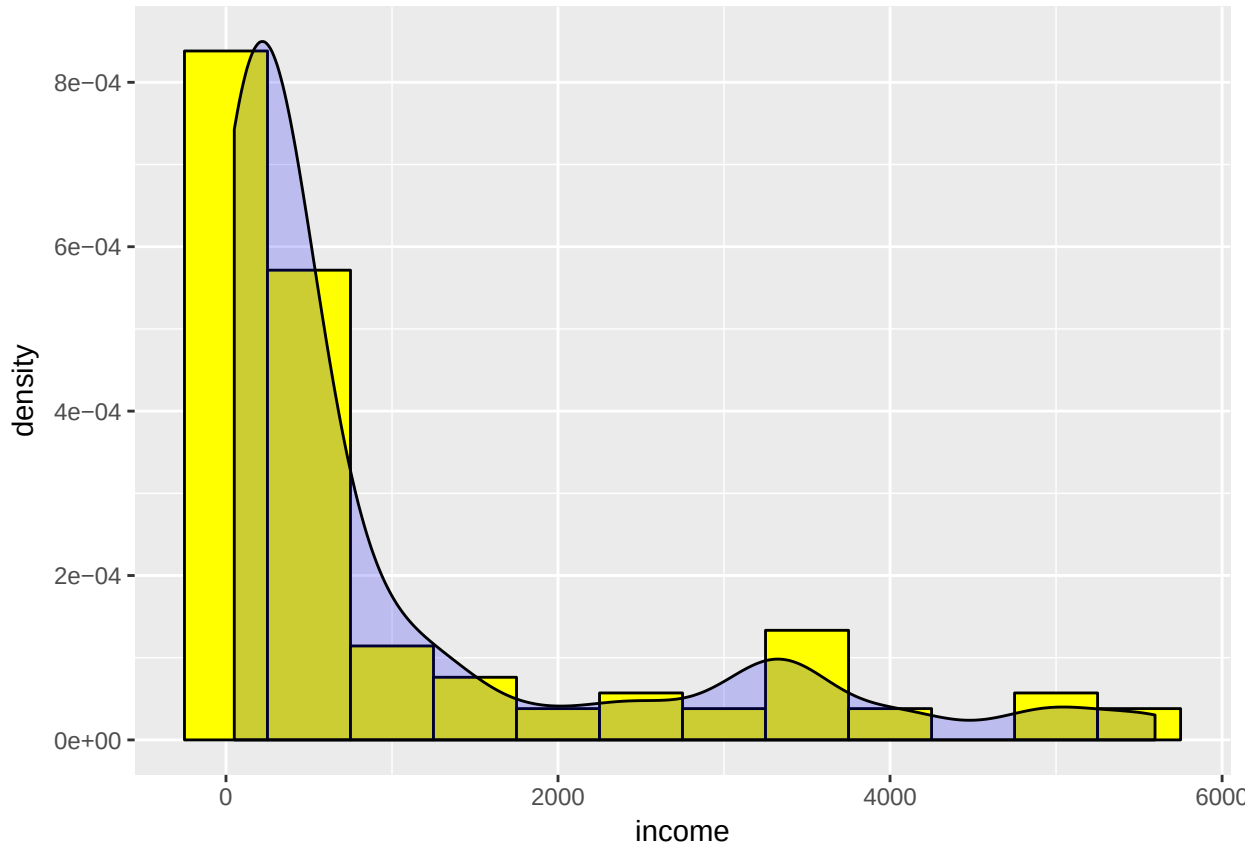
```
ggplot(dataLeinhardt, aes(income)) +
  geom_histogram()
```



A great website for learning `ggplot2` and seeing worked example on how to create various graphics is the "Cookbook for R" website at http://www.cookbook-r.com/Graphs/.

For example, here is a nice histogram with a density curve overlaid with color - see the 4th histogram plot and associated code at http://www.cookbook-r.com/Graphs/Plotting_distributions_(ggplot2)/. Let's adapt this code for our histogram of `income`. Basically, change out the dataset name, the variable listed in `aes()` and change the `binwidth` to something more appropriate for this `income` variable. Feel free to play with changing the colors.

```
# Histogram overlaid with kernel density curve
ggplot(dataLeinhardt, aes(x=income)) +
    geom_histogram(aes(y=..density..),
                   binwidth=500,
                   colour="black", fill="yellow") +
    geom_density(alpha=.2, fill="blue")
```

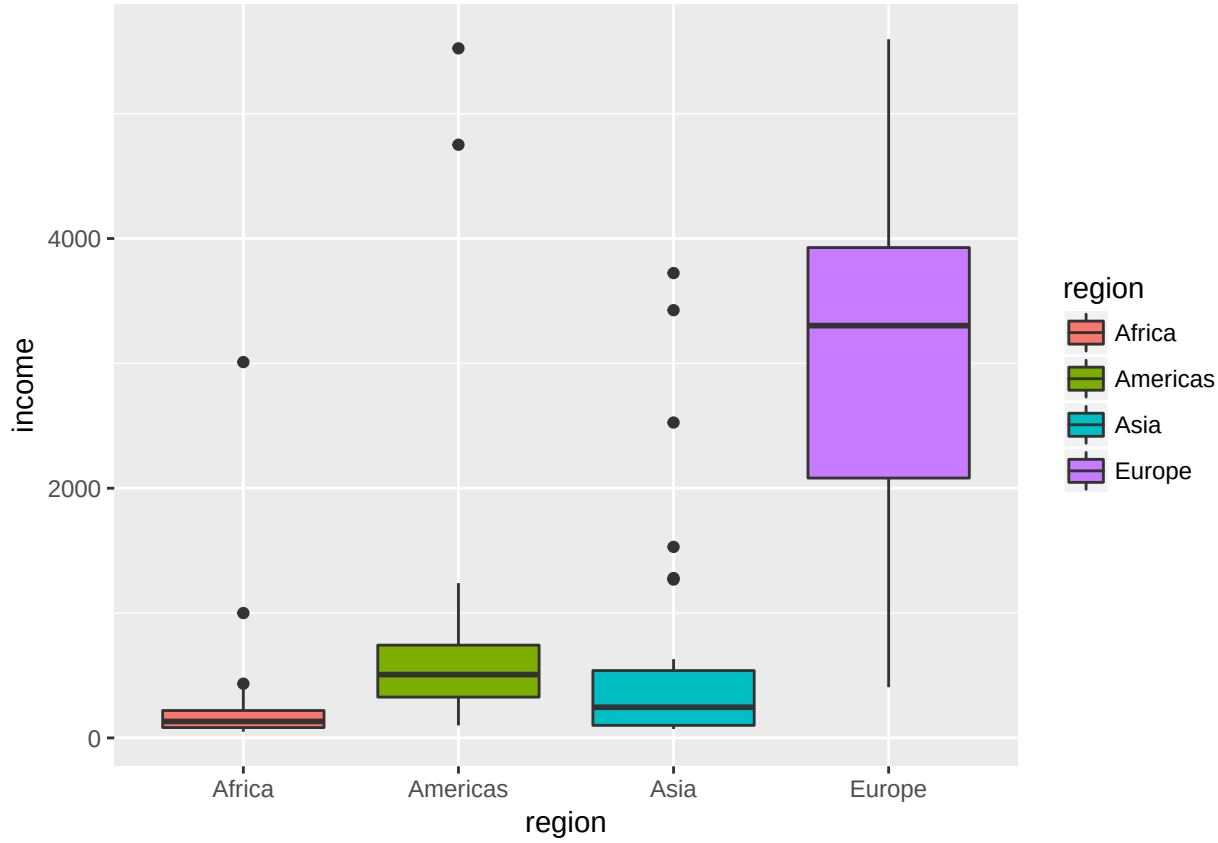


Boxplots by groups

What if we want to look at the income distributions by region. We can do that using the `geom_boxplot()` and updating the `aes()` list.

Again, see examples at http://www.cookbook-r.com/Graphs/Plotting_distributions_(ggplot2)/.

```
# A basic box with the conditions colored
ggplot(dataLeinhardt,
       aes(x=region, y=income, fill=region)) +
  geom_boxplot()
```



Transform, Create new variables

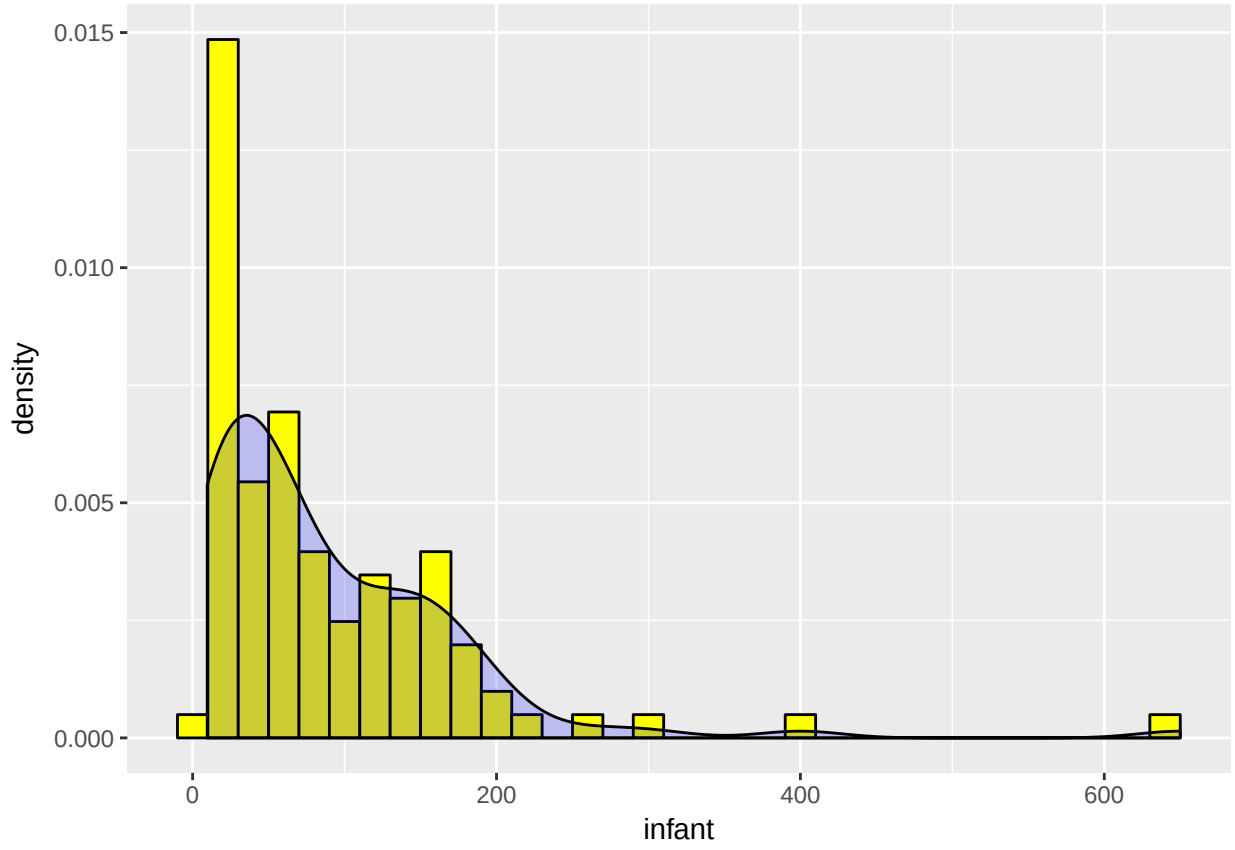
The infant mortality rates are very right skewed (longer tail to the right). A possible mathematical transformation of right skewness is to do a log transformation. Let's create a new variable in the dataset that is the log of the infant mortality rate using the `log()` function from base `R` and the `mutate()` function from `dplyr`.

```
dataLeinhardt <- dataLeinhardt %>%
  mutate(logInfant = log(infant))
```

Compare these 2 variables using histograms. Notice that the bandwidths are different.

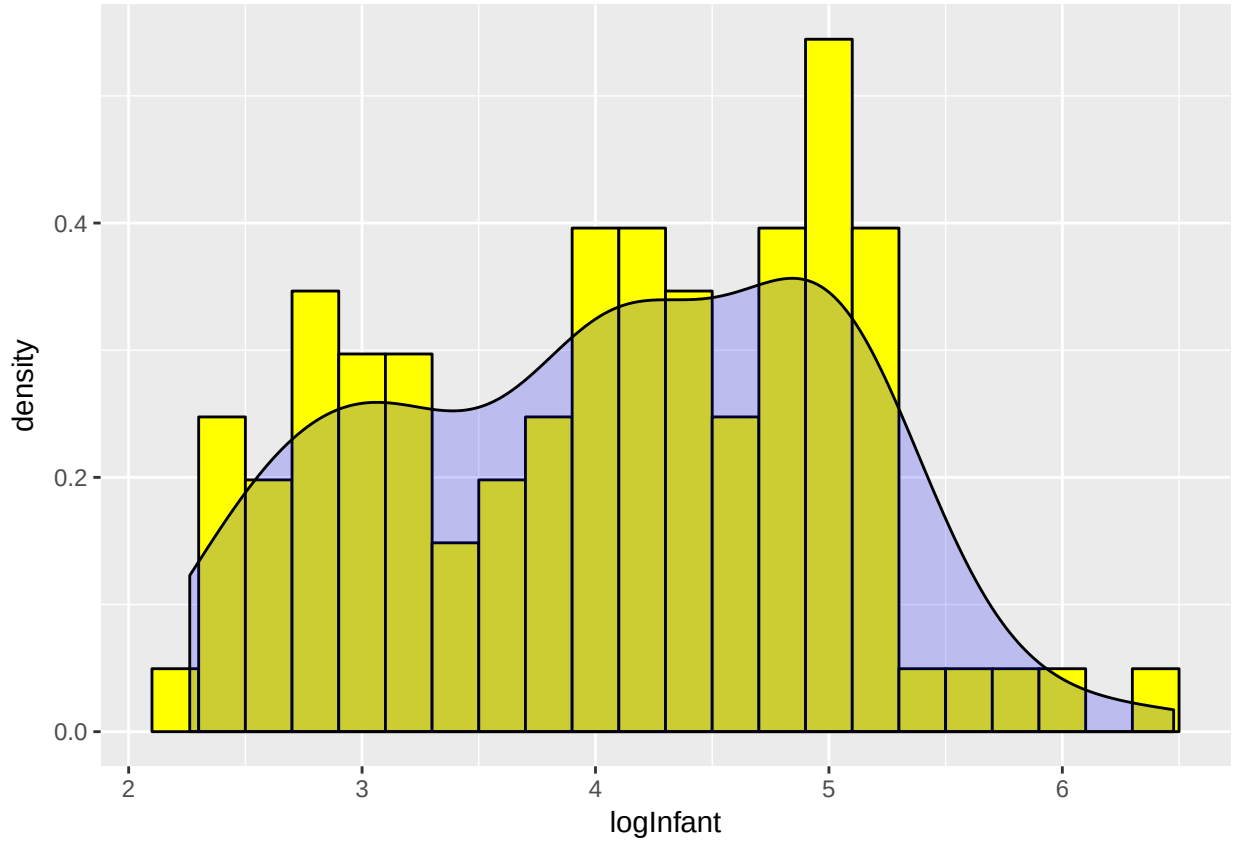
Original `infant` variable

```
ggplot(dataLeinhardt, aes(x=infant)) +
    geom_histogram(aes(y=..density..),
                   binwidth=20,
                   colour="black", fill="yellow") +
    geom_density(alpha=.2, fill="blue")
```



Log of `income` variable

```
ggplot(dataLeinhardt, aes(x=logInfant)) +
    geom_histogram(aes(y=..density..),
                   binwidth=.2,
                   colour="black", fill="yellow") +
    geom_density(alpha=.2, fill="blue")
```



Recode data

Suppose we wanted to break the infant mortality rates into low, moderate and high categories. For example, let's create the following categories based on these infant mortality rates.

Infant Mortality Categories

Category	Infant Mortality (rate/1000 live births)
Low	<50
Moderate	50 to <100
High	100 or more

To do this we'll create a new variable using the `mutate()` function in `dplyr` AND we'll also use the `if_else()` function in `dplyr`.

```
dataLeinhardt <- dataLeinhardt %>%
  mutate(infantcat = if_else(infant<50, "1. Low",
                     if_else(infant<100,
                             "2. Moderate",
                             "3. High",
                             "4. missing"),
                     "4. missing"))
```

Count the number of countries that fall into each category

```
dataLeinhardt %>%
  count(infantcat)
```

```
## # A tibble: 4 x 2
##     infantcat     n
##         <chr> <int>
## 1      1. Low    41
## 2 2. Moderate    23
## 3     3. High    37
## 4  4. missing     4
```

[OPTIONAL] Here is a nice way to get a formatted table of counts and percents using the `tabyl()` function from the `janitor` package.

```
# optional - a nice way to get a formatted
# table of the counts for infant categories
library(janitor)
dataLeinhardt %>%
  janitor::tabyl(infantcat)
```

```
##     infantcat  n    percent
## 1      1. Low 41 0.39047619
## 2 2. Moderate 23 0.21904762
## 3     3. High 37 0.35238095
## 4  4. missing  4 0.03809524
```

Run again adding the `knitr::kable()` function when using INSIDE of an RMD document.

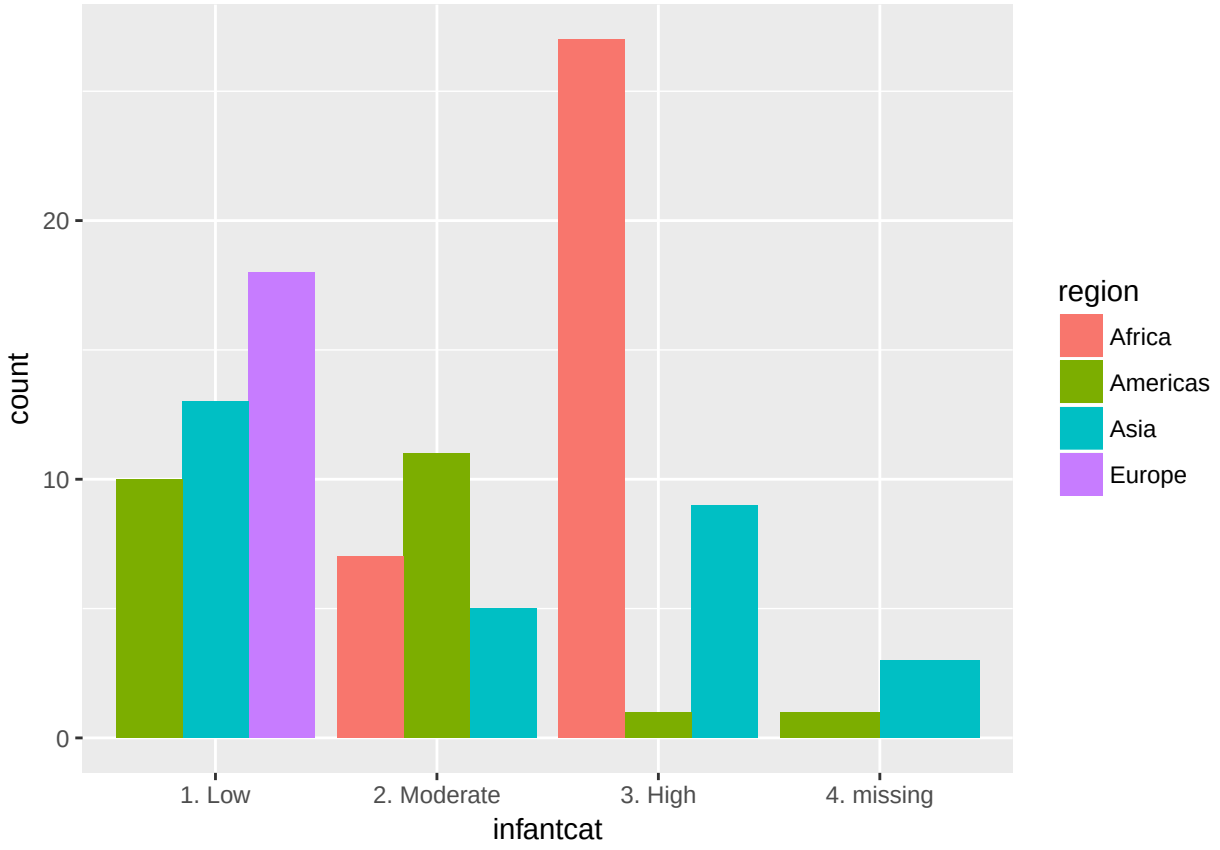
```
dataLeinhardt %>%
  janitor::tabyl(infantcat) %>%
  knitr::kable()
```

infantcat	n	percent
1. Low	41	0.3904762
2. Moderate	23	0.2190476
3. High	37	0.3523810
4. missing	4	0.0380952

Clustered bar chart

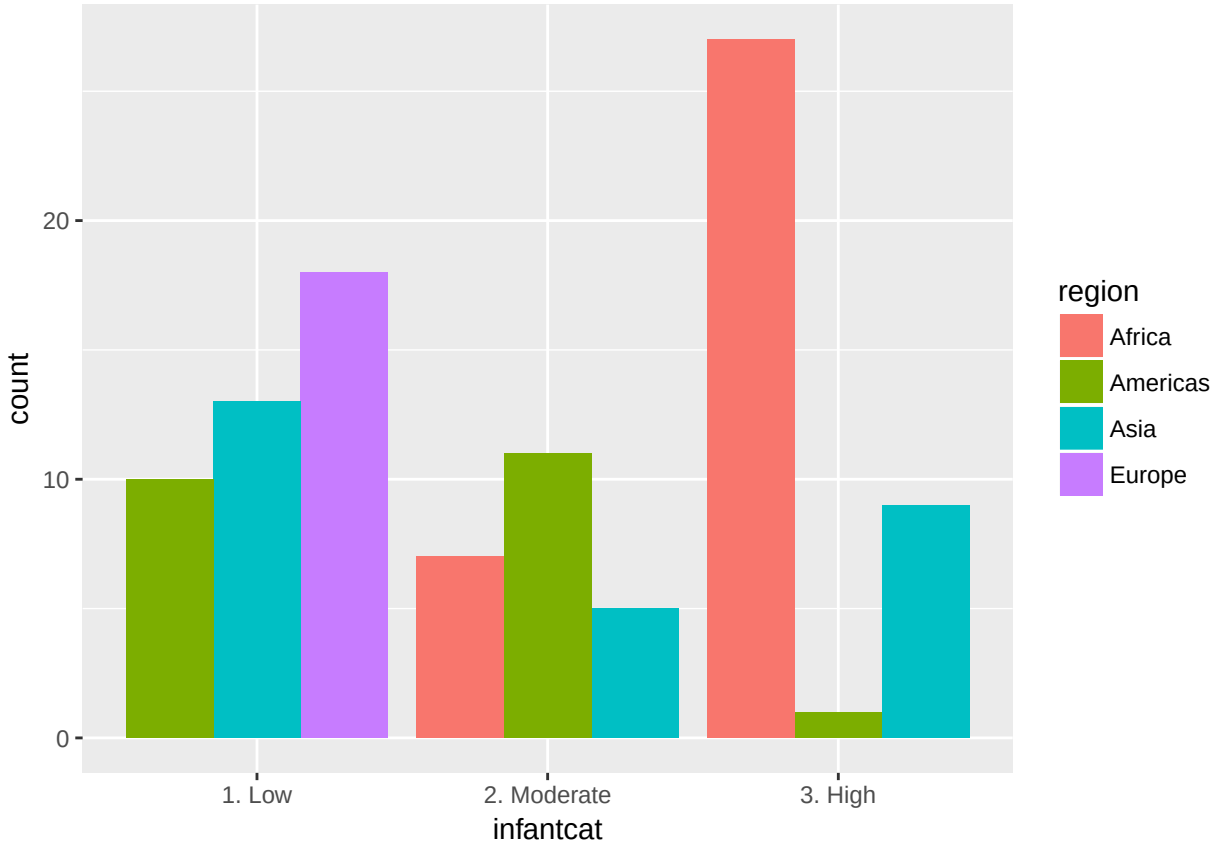
Let's look at these infant mortality rate categories by region by making a clustered bar chart. some examples can be seen at http://www.cookbook-r.com/Graphs/Bar_and_line_graphs_(ggplot2)/#with-x-axis-treated-as-categorical. Scroll to the bottom of the page.

```
# cluster barchart of infant categories by region
dataLeinhardt %>% ggplot(aes(x=infantcat, fill=region)) +
  geom_bar(position = "dodge")
```



Make the chart again removing the countries with missing infant mortality rates. Use the `is.na()` function and put an exclamation point `!` in front to say find all without missing data, find when NOT TRUE.

```
dataLeinhardt %>%
  filter(!is.na(infant)) %>%
  ggplot(aes(x=infantcat, fill=region)) +
  geom_bar(position = "dodge")
```



Other things to try

1. Go back to the first R-code chunk and change `echo=TRUE` to `echo=FALSE` and see what happens to the output.
2. Install the `Rcmdr` package and then load it by running `library(Rcmdr)` in the Console window - try out using the R Commander package. Learn more at http://www.rcommander.com/.
3. Kahoot Quiz