Aggregation

`Data Aggregation:` The process of taking some data and putting it into a form that lends itself easily to summary, e.g. replace groups of observations with summary statistics.

`Data Restructuring:` Change the structure of the data so that its new form is more convenient for a specific purpose (usually analysis).

R has many ways to do either. Before you start writing code to do some of these activities check around to see what functions exist. Chances are there are some good tools available. We will explore many of them in this session.

Command	Purpose
table, xtabs	Create Contingency Tables
tapply, split	Summarize a continuous variable by a grouping variables
aggregate	Summarize continuous variable(s) by grouping variables
dplyr	Split, Apply, Combine

Counting

One of the most basic forms of aggregation is to put things into tables for easy counting. This is usually done with categorical variables to obtain "count" information. We can then do things like Chi-Square tests.

```
letters[1:10]                    # Built in letters char vector
```

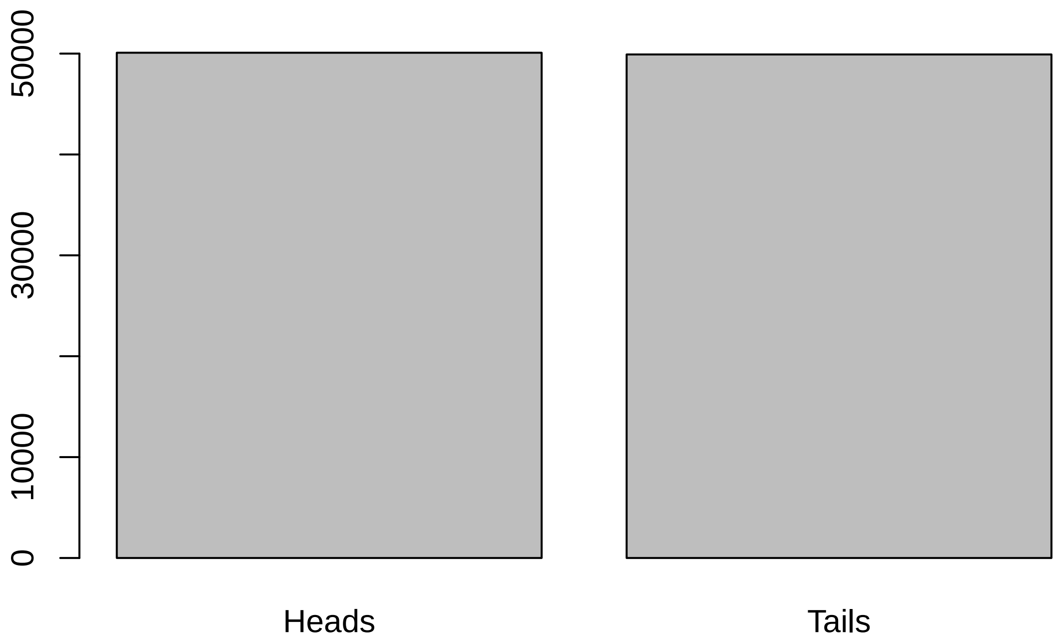
```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```
my.sample <- sample(letters[1:10],50,replace=TRUE)

table(my.sample)
```

```
## my.sample
## a b c d e f g h i j
## 8 6 5 4 4 6 4 6 2 5
```

We can "flip" a coin 100,000 times to see if its a fair coin

```
n <- 100000
coin_flips <- sample(c("Heads","Tails"),100000,T)

# Make a table
coin_tab <- table(coin_flips)
barplot(coin_tab)
```

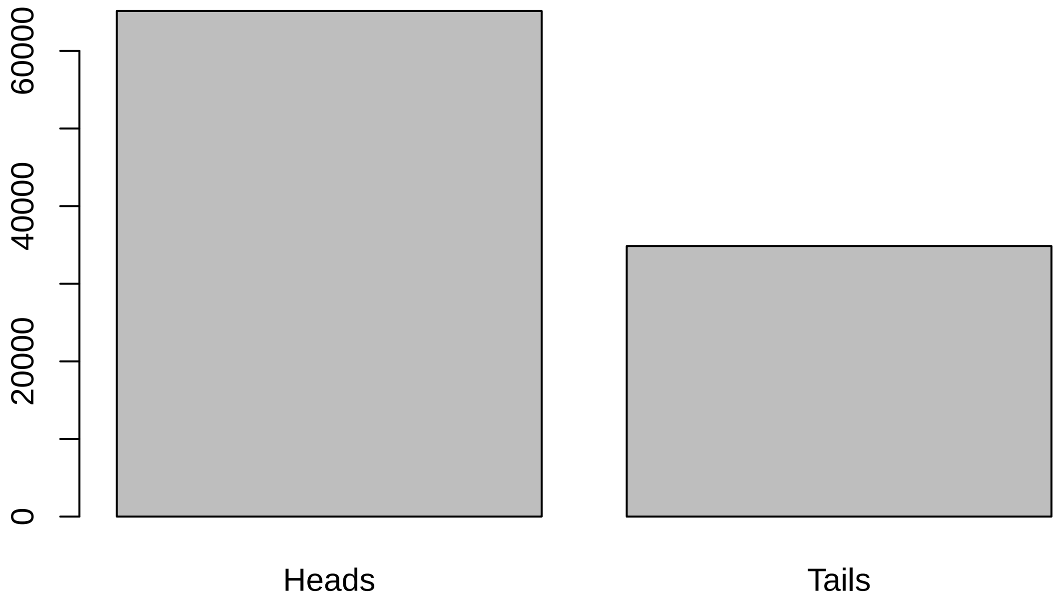



```
# Fair coin?
chisq.test(coin_tab)
```

```
##
##  Chi-squared test for given probabilities
##
## data:  coin_tab
## X-squared = 0.29584, df = 1, p-value = 0.5865
```

Un Fair Coin ?

```
n <- 100000
coin_flips <- sample(c("Heads","Tails"),100000,T,prob=c(.65,.35))

# Make a table
coin_tab <- table(coin_flips)
barplot(coin_tab)
```




```r
# Fair coin?
chisq.test(coin_tab)
```

```
##
##  Chi-squared test for given probabilities
##
## data:  coin_tab
## X-squared = 9179.7, df = 1, p-value < 2.2e-16
```

However, we usually create tables out of a data frame that contains some categories or factors. Using the popular mtcars data set let's find out how many 4,6,8 and cylinder cars there are for each category of transmission (auto or manual).

```r
table(transmission=mtcars$am,cylinder=mtcars$cyl)
```

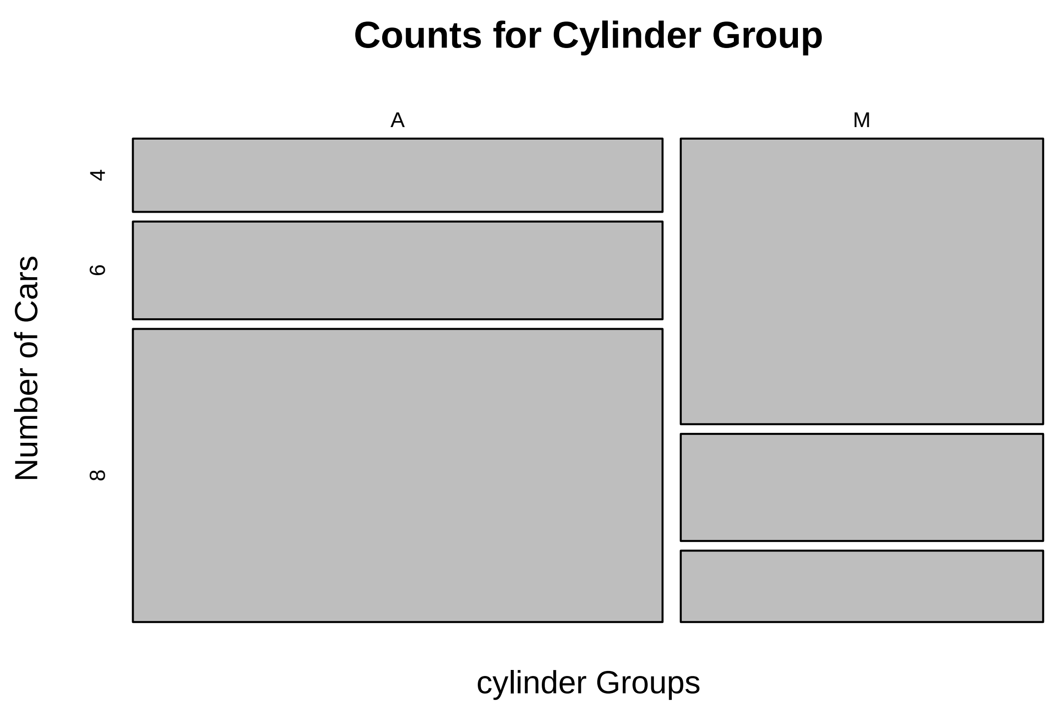
```
##             cylinder
## transmission  4  6  8
##            0  3  4 12
##            1  8  3  2
```

We might want to make factors more obvious

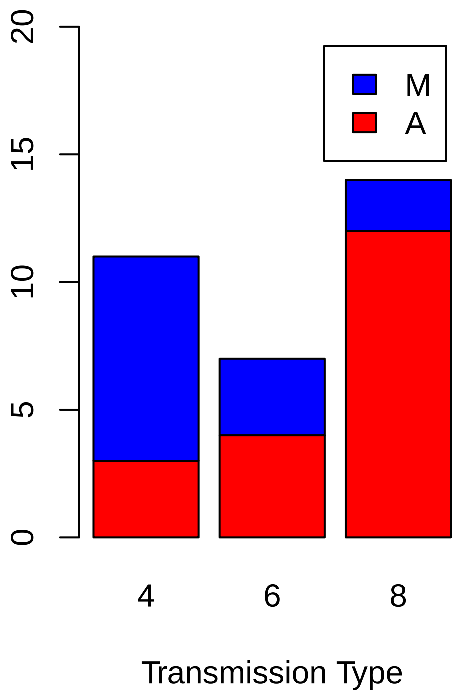
```r
mtcars <- transform(mtcars,am=factor(am,labels=c("A","M")))
table(mtcars$am,mtcars$cyl)
```

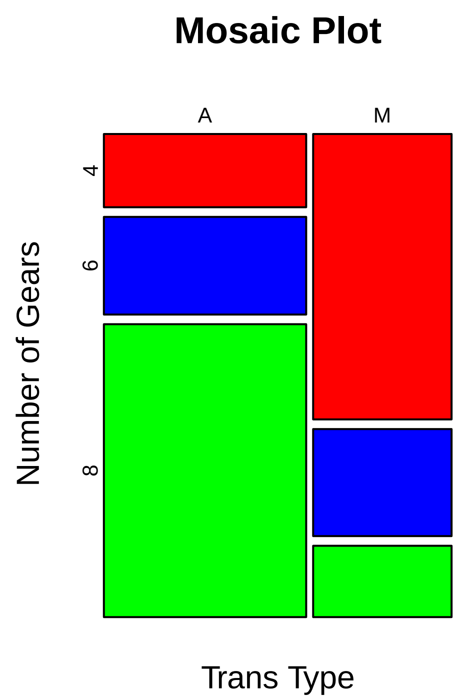
```
##
##      4  6  8
##   A  3  4 12
##   M  8  3  2
```

```r
plot(table(mtcars$am,mtcars$cyl),main="Counts for Cylinder Group",
        ylab="Number of Cars",xlab="cylinder Groups")
```




```
par(mfrow=c(1,2))

my.table <- table(mtcars$am,mtcars$cyl)

barplot(my.table,legend=T,col=c("red","blue","green"),
          xlab="Transmission Type",ylim=c(0,20))


# Here we get a Mosaic Plot

plot(my.table,col=c("red","blue","green"),main="Mosaic Plot",
      xlab="Trans Type", ylab="Number of Gears")
```





Remember that we can take continuous variables and turn them into categories.

```
my.cut <- cut(mtcars$mpg,breaks=4,labels=c("Bad","Not Good","Good","Great"))

my.cut
```

```
##  [1] Not Good Not Good Good     Not Good Not Good Not Good Bad      Good
##  [9] Good     Not Good Not Good Not Good Not Good Bad      Bad      Bad
## [17] Bad      Great    Great    Great    Not Good Bad      Bad      Bad
## [25] Not Good Good     Good     Great    Bad      Not Good Bad      Not Good
## Levels: Bad Not Good Good Great
```

```
mtcars$cyl <- factor(mtcars$cyl,labels=paste("Cyl",seq(4,8,2),sep="_"))

table(MPG=my.cut,mtcars$cyl)
```

```
##
## MPG         Cyl_4 Cyl_6 Cyl_8
##   Bad           0     0    10
##   Not Good      2     7     4
##   Good          5     0     0
##   Great         4     0     0
```

xtabs

This wouldn't be R unless there were some other function that does pretty much the same thing as table. There is another function called "xtabs". Some people feel that it is superior to the table function because it supports a formula interface when creating tables. The formula interface requires the "+" symbol to be on the right of any grouping variable.

```
xtabs(~am+cyl,mtcars)     # Transmission type by number of cylinders
```

```
##    cyl
## am  Cyl_4 Cyl_6 Cyl_8
##   A     3     4    12
##   M     8     3     2
```

```
xtabs(~cyl+am,mtcars)     # Number of cylinders by transmission type
```

```
##        am
## cyl      A  M
##   Cyl_4  3  8
##   Cyl_6  4  3
##   Cyl_8 12  2
```

tapply summarize continuous quantities by categories

Up until now we've been working with tables created from categorical variables. Even in our example that used a continuous variable we wound up using the cut command to group the continuous variables into categories.

```
tapply(mtcars$mpg, mtcars$am, mean)
```

```
##        A        M
## 17.14737 24.39231
```

```
#
```

```
tapply(mtcars$mpg, list(mtcars$am, mtcars$vs),mean)
```

```
##       0        1
## A 15.05 20.74286
## M 19.75 28.37143
```

Note that we can supply our own function when using tapply. The function in this example is used to generate the standard error. This is what we call an anonymous function since we don't even bother to give it a name. It just exists only for the duration of the call to tapply.

```
myFunc <- function(x) {
    se = (sqrt(var(x)/length(x)))
    return(se)
}

#tapply(mtcars$mpg, list(mtcars$am, mtcars$vs), myFunc)
tapply(mtcars$mpg, list(mtcars$am,mtcars$cyl),myFunc)
```

```
##       Cyl_4     Cyl_6     Cyl_8
## A 0.8386497 0.8158584 0.8008991
## M 1.5852839 0.4333333 0.4000000
```

The Split function

```
mysplit <- split(mtcars,mtcars$cyl)
```

While the split command isn't specifically associated with aggregation it can be used to rapidly separate a data frame based on a factor. It the creates a list wherein each element of the list contains that portion of the data frame corresponding to a value of the factor. In the previous example we create a three element list with each element being the part of the data frame corresponding to 4,6, or 8 cylinders.

```
my_summary <- sapply(mysplit,function(x) mean(x$mpg))
```

Aggregate Command

```
aggregate(mtcars['mpg'],list(Transmission=mtcars$am),mean)
```

```
##   Transmission      mpg
## 1            A 17.14737
## 2            M 24.39231
```

```
aggregate(mtcars[c('mpg','hp')],list(Transmission_Type=mtcars$am),mean)
```

```
##   Transmission_Type      mpg       hp
## 1                 A 17.14737 160.2632
## 2                 M 24.39231 126.8462
```

```
aggregate(mtcars[c('mpg','hp')],
          list(Transmission_Type = mtcars$am,
          Cylinders = mtcars$cyl),mean)
```

```
##   Transmission_Type Cylinders      mpg        hp
## 1                 A     Cyl_4 22.90000  84.66667
## 2                 M     Cyl_4 28.07500  81.87500
## 3                 A     Cyl_6 19.12500 115.25000
## 4                 M     Cyl_6 20.56667 131.66667
## 5                 A     Cyl_8 15.05000 194.16667
## 6                 M     Cyl_8 15.40000 299.50000
```

The aggregate command also has a formula interface if you find that to be more convenient. Many do since it gives you an argument to specify the data frame you are trying to summarize. This saves typing.

```
aggregate(mpg ~ am, mtcars, mean)
```

```
##   am      mpg
## 1  A 17.14737
## 2  M 24.39231
```

```
aggregate(mpg ~ am + cyl, mtcars, mean)
```

```
##   am   cyl      mpg
## 1  A Cyl_4 22.90000
## 2  M Cyl_4 28.07500
## 3  A Cyl_6 19.12500
## 4  M Cyl_6 20.56667
## 5  A Cyl_8 15.05000
## 6  M Cyl_8 15.40000
```

We can actually reduce things down further if we want. In effect we are trying to further simplify the table.

```
(myagg <- aggregate(mpg ~ am + cyl, mtcars, mean))
```

```
##   am   cyl      mpg
## 1  A Cyl_4 22.90000
## 2  M Cyl_4 28.07500
## 3  A Cyl_6 19.12500
## 4  M Cyl_6 20.56667
## 5  A Cyl_8 15.05000
## 6  M Cyl_8 15.40000
```

```r
# Let's reduce this further.

xtabs(mpg~am+cyl,myagg)
```

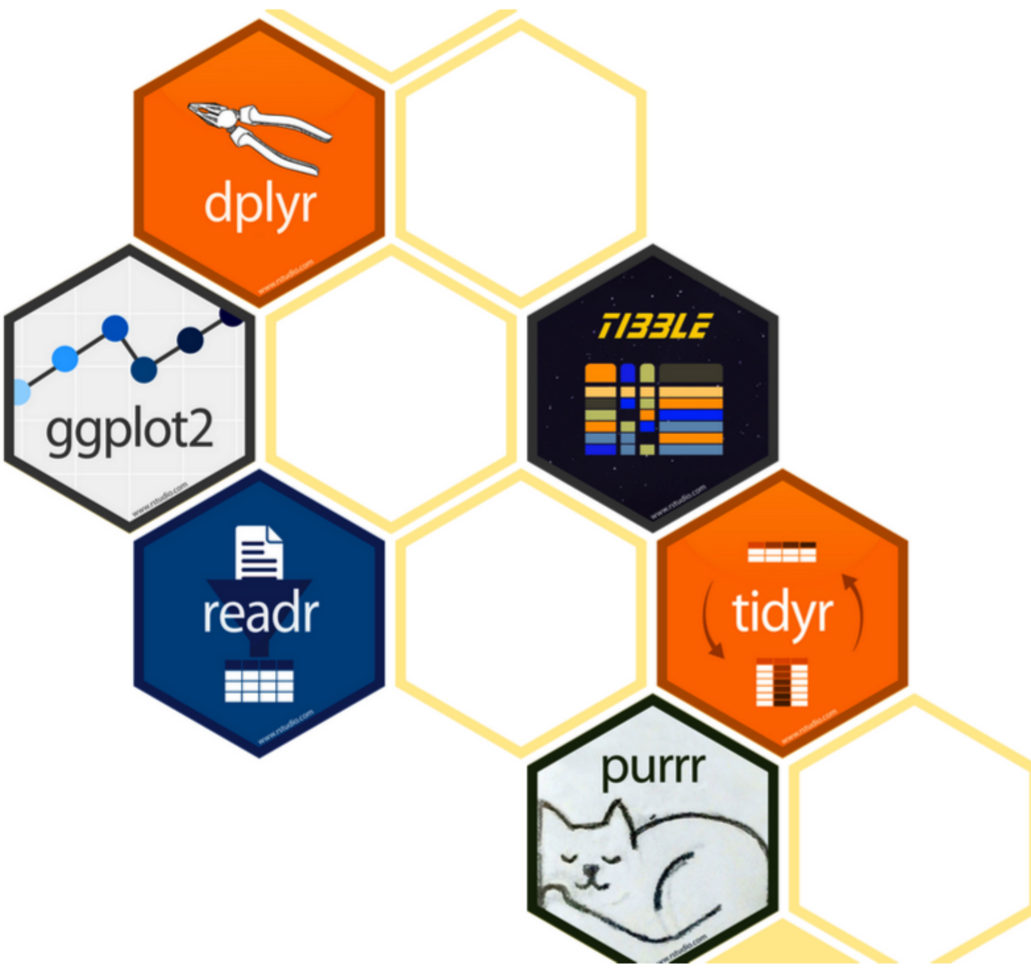
```
##    cyl
## am     Cyl_4    Cyl_6    Cyl_8
##   A 22.90000 19.12500 15.05000
##   M 28.07500 20.56667 15.40000
```

dplyr and the tidyverse

dplyr is an add on package designed to efficiently transform and summarize tabular data such as data frames. The package has a number of functions ("verbs") that perform a number of data manipulation tasks:

- Filtering rows
- Select specific columns
- Re-ordering or arranging rows
- Summarizing and aggregating data

One of the unique strengths of dplyr is that it implements what is known as a Split-Apply-Combine technique that we will explore in this session. dplyr is part of the tidyverse package:



The dplyr package is part of the larger tidyverse package set which has expanded considerably in recent years and continues to grow in size and utility such that many people never learn the "older way" of doing things in R. But we've already been through that in the previous section. The tidyverse has the following packages. The descriptions have been lifted from the tidyverse home page.

ggplot2 - ggplot2 is a system for declaratively creating graphics, based on The Grammar of Graphics. You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

dplyr - dplyr provides a grammar of data manipulation, providing a consistent set of verbs that solve the most common data manipulation challenges.

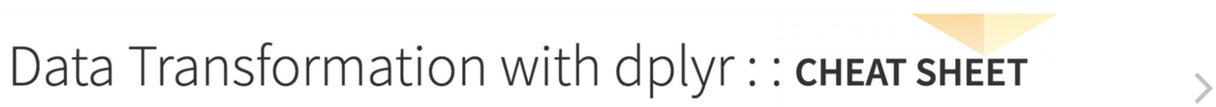
tidyr - tidyr provides a set of functions that help you get to tidy data. Tidy data is data with a consistent form: in brief, every variable goes in a column, and every column is a variable.

readr - readr provides a fast and friendly way to read rectangular data (like csv, tsv, and fwf). It is designed to flexibly parse many types of data found in the wild, while still cleanly failing when data unexpectedly changes.

tibble - tibble is a modern re-imagining of the data frame, keeping what time has proven to be effective, and throwing out what it has not. Tibbles are data.frames that are lazy and surly: they do less and complain more forcing you to confront problems earlier, typically leading to cleaner, more expressive code.

stringr - stringr provides a cohesive set of functions designed to make working with strings as easy as possible. It is built on top of stringi, which uses the ICU C library to provide fast, correct implementations of common string manipulations.

lubridate - Date-time data can be frustrating to work with in R. R commands for date-times are generally unintuitive and change depending on the type of date-time object being used. Moreover, the methods we use with date-times must be robust to time zones, leap days, daylight savings times, and other time related quirks. Lubridate makes it easier to do the things R does with date-times and possible to do the things R does not.



A handy reference for dplyr can be found here

```
# install the package

# install.packages("dplyr")
# install.packages("readr")    # Get's the equivalent to data.table's fread package

# You could load the entire tidyverse or just parts of it

# library(tidyverse)
suppressMessages(library(dplyr))

# Launches a browser to explore
# browseVignettes(package = "dplyr")
```

dplyr verbs

There are some common activities associated with a data frame:

filter - find observations satisfying some condition(s)

select - selecting specific columns by name

mutate - adding new columns or changing existing ones arrange - reorder or sort the rows

summarize - do some aggregation or summary by groups

```
df <- data.frame(id = 1:5,
                 gender = c("MALE","MALE","FEMALE","MALE","FEMALE"),
                 age = c(70,76,60,64,68))
df
```

```
##   id gender age
## 1  1   MALE  70
## 2  2   MALE  76
## 3  3 FEMALE  60
## 4  4   MALE  64
## 5  5 FEMALE  68
```

ID	GENDER	AGE
1	MALE	70
2	MALE	76
3	FEMALE	60
4	MALE	64
5	FEMALE	68

Filter is for finding rows that satisfy certain criteria

```
filter(df,gender == "FEMALE")
```

```
##   id gender age
## 1  3 FEMALE  60
## 2  5 FEMALE  68
```

```
filter(df, id %in% c(1,3,5))
```

```
##   id gender age
## 1  1   MALE  70
## 2  3 FEMALE  60
## 3  5 FEMALE  68
```

Equivalent to:

```
filter(df, id == "1")
```

```
##   id gender age
## 1  1   MALE  70
```

ID	GENDER	AGE
1	MALE	70
2	MALE	76
3	FEMALE	60
4	MALE	64
5	FEMALE	68

ID	GENDER	AGE
3	FEMALE	60
5	FEMALE	68

So, now find only the ids that relate to rows 1,3, or 5. This is a highly specialized search but it is helpful to show that you can use a wide variety of logical constructs.

```
filter(df, id %in% c(1,3,5))
```

```
##   id gender age
## 1  1   MALE  70
## 2  3 FEMALE  60
## 3  5 FEMALE  68
```

ID	GENDER	AGE
1	MALE	70
2	MALE	76
3	FEMALE	60
4	MALE	64
5	FEMALE	68

ID	GENDER	AGE
1	MALE	70
3	FEMALE	60
5	FEMALE	68

mutate is for changing columns or adding new ones

```
mutate(df,meanage = mean(age))
```

```
##   id gender age meanage
## 1  1   MALE  70    67.6
## 2  2   MALE  76    67.6
## 3  3 FEMALE  60    67.6
## 4  4   MALE  64    67.6
## 5  5 FEMALE  68    67.6
```

```r
mutate(df,old_young=ifelse(df$age>=mean(df$age),"Y","N"))
```

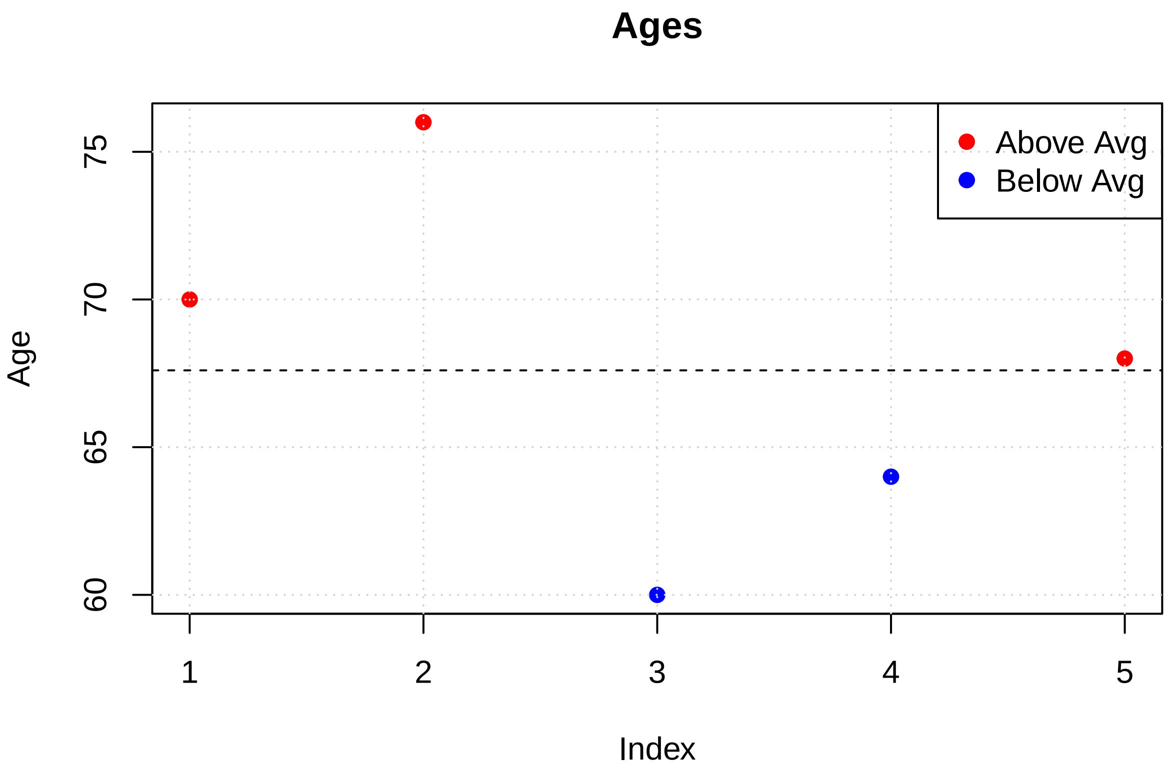
```
##   id gender age old_young
## 1  1   MALE  70         Y
## 2  2   MALE  76         Y
## 3  3 FEMALE  60         N
## 4  4   MALE  64         N
## 5  5 FEMALE  68         Y
```

ID	GENDER	AGE
1	MALE	70
2	MALE	76
3	FEMALE	60
4	MALE	64
5	FEMALE	68

ID	GENDER	AGE	MEANWT
1	MALE	70	67.6
2	MALE	76	67.6
3	FEMALE	60	67.6
4	MALE	64	67.6
5	FEMALE	68	67.6

Next we will create a new column designed to tell us if a given observation has an age that is greater than or equal to the average age. Specifically, create a variable called old_young and assign a value of "Y" if the observed age for that row is above the mean age and a value of "N" if it is not.

```r
tmp <- mutate(df, color = ifelse(age > mean(age),"red","blue"))

plot(tmp$age,col=tmp$color,type="p",pch=19,main="Ages",ylab="Age")

grid()

abline(h=mean(tmp$age),lty=2)

legend("topright",c("Above Avg","Below Avg"),col=c("red","blue"),pch=19)
```



Use arrange for sorting the data frame by a column(s)

```
# Sort df by age from highest to lowest

arrange(df, desc(age))
```

```
##   id gender age
## 1  2   MALE  76
## 2  1   MALE  70
## 3  5 FEMALE  68
## 4  4   MALE  64
## 5  3 FEMALE  60
```

```
# Sort df by gender (alphabetically) and then by age
# from highest to lowest

arrange(df, gender,desc(age))
```

```
##   id gender age
## 1  5 FEMALE  68
## 2  3 FEMALE  60
## 3  2   MALE  76
## 4  1   MALE  70
## 5  4   MALE  64
```

Select allows us to select groups of columns from a data frame

```
select(df,gender,id,age)  # Reorder the columns
```

```
##   gender id age
```

```
## 1   MALE  1  70
## 2   MALE  2  76
## 3 FEMALE  3  60
## 4   MALE  4  64
## 5 FEMALE  5  68
```

```
select(df,-age)   # Select all but the age column
```

```
##   id gender
## 1  1   MALE
## 2  2   MALE
## 3  3 FEMALE
## 4  4   MALE
## 5  5 FEMALE
```

```
select(df,id:age)  # Can use : to select a range
```

```
##   id gender age
## 1  1   MALE  70
## 2  2   MALE  76
## 3  3 FEMALE  60
## 4  4   MALE  64
## 5  5 FEMALE  68
```

```
df
```

```
##   id gender age
## 1  1   MALE  70
## 2  2   MALE  76
## 3  3 FEMALE  60
## 4  4   MALE  64
## 5  5 FEMALE  68
```

```
group_by(df,gender)   # Hmm. Did this really do anything ?
```

```
## # A tibble: 5 x 3
## # Groups:   gender [2]
##      id gender   age
##   <int> <chr>  <dbl>
## 1     1 MALE      70
## 2     2 MALE      76
## 3     3 FEMALE    60
## 4     4 MALE      64
## 5     5 FEMALE    68
```

Actually, it does but until it is paired with a summarize it does not become apparent

```
summarize(group_by(df,gender),total=n())
```

```
## # A tibble: 2 x 2
##   gender total
##   <chr>  <int>
## 1 FEMALE     2
## 2 MALE       3
```

ID	GENDER	AGE
1	MALE	70
2	MALE	76
3	FEMALE	60
4	MALE	64
5	FEMALE	68

GENDER	TOTAL
FEMALE	2
MALE	3

```r
summarize(group_by(df,gender),av_age=mean(age))
```

```
## # A tibble: 2 x 2
##   gender av_age
##   <chr>   <dbl>
## 1 FEMALE     64
## 2 MALE       70
```

ID	GENDER	AGE
1	MALE	70
2	MALE	76
3	FEMALE	60
4	MALE	64
5	FEMALE	68

GENDER	AV_AGE
FEMALE	64
MALE	70

```r
summarize(group_by(df,gender),
          av_age=mean(age),
          total=n())
```

```
## # A tibble: 2 x 3
##   gender av_age total
##   <chr>   <dbl> <int>
## 1 FEMALE     64     2
## 2 MALE       70     3
```

But do you really need dplyr to do this ? No but it makes it a lot easier

```r
df
```

```
##   id gender age
## 1  1   MALE  70
## 2  2   MALE  76
## 3  3 FEMALE  60
```

```
## 4  4   MALE  64
## 5  5 FEMALE  68
```

```r
tapply(df$age,df$gender,mean)    # tapply function
```

```
## FEMALE   MALE
##     64     70
```

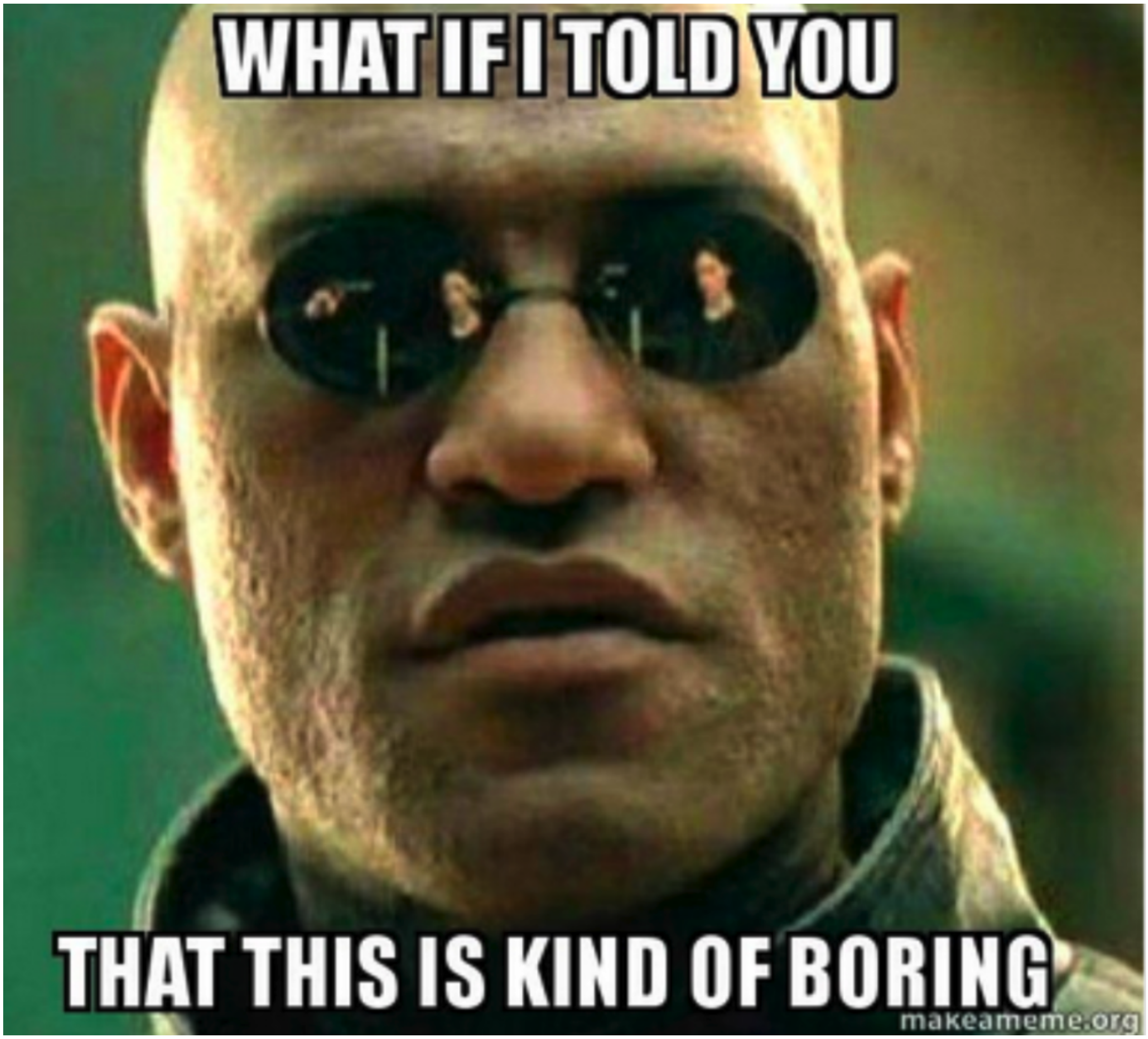
```r
aggregate(age~gender,data=df,mean) # aggregate works also
```

```
##   gender age
## 1 FEMALE  64
## 2   MALE  70
```

```r
lapply(split(df,df$gender),function(x) mean(x$age)) # complicated
```

```
## $FEMALE
## [1] 64
##
## $MALE
## [1] 70
```

Pipes

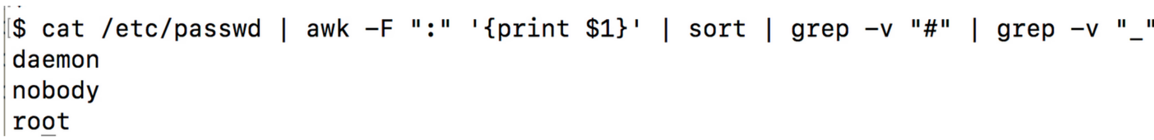


While what we have done with dplyr up until now is useful, it still might not be apparent how it could "replace" the native R commands. And in fact, you might not want to replace it. But the use of **pipes** makes a compelling case to do so.

Let us consider the `pipe` operator that is included with the tidyverse. This is used to make it possible to "pipe' ' the results of one command into another command and so on.

The inspiration for this comes from the UNIX/LINUX operating system where pipes are used all the time. So in effect using "pipes' ' is nothing new in the world of research computation.

Once you get used to pipes it is hard to go back to not using them. The inspiration for this comes from the UNIX/LINUX operating system where pipes are used all the time. So in effect using "pipes" is nothing new in the world of research computation. The concept goes back decades.



```
$ cat /etc/passwd | awk -F ":" '{print $1}' | sort | grep -v "#" | grep -v "_"
daemon
nobody
root
```

```
cat /etc/passwd | awk -F ":" '{print $1}' | sort | grep -v "#" | grep -v "_"
```

So this is the traditional way of using functions in R. We use the composite function approach which uses something like f(g(x)) where in this case, g(x) is represented by the select function and f(x) is the head function.

```
head(select(mtcars, mpg, am))
```

```
##                    mpg am
## Mazda RX4         21.0  M
## Mazda RX4 Wag     21.0  M
## Datsun 710        22.8  M
## Hornet 4 Drive    21.4  A
## Hornet Sportabout 18.7  A
## Valiant           18.1  A
```

Here we will select the mpg and am column from mtcars and view the top 5 rows but using dplyr and the piping operator. Instead of nesting functions (reading from the inside to the outside), the idea of of piping is to read the functions from left to right. In effect we have a "stream" of data flowing from left to right which could be filtered, mutated, arranged or grouped.

```
mtcars %>% select(mpg, am) %>% head
```

```
##                    mpg am
## Mazda RX4         21.0  M
## Mazda RX4 Wag     21.0  M
## Datsun 710        22.8  M
## Hornet 4 Drive    21.4  A
## Hornet Sportabout 18.7  A
## Valiant           18.1  A
```

The key to understanding how this works is to read this from left to right. It bears repeating that each command is "its own thing" independently of the pipe character. So the:

1. output of mtcars goes into the
2. input of the **select** function whose output goes into the
3. input of the **head** function

Split-Apply-Combine

Let's use our new found knowledge to re-imagine our use of the group_by and summarize functions that we have been using in composite form up until now. What about this ? We can chain together the output of one command to the input of another !

```r
df %>% group_by(gender) %>% summarize(avg=mean(age))
```

```
## # A tibble: 2 x 2
##   gender   avg
##   <chr>  <dbl>
## 1 FEMALE    64
## 2 MALE      70
```

```r
df %>% group_by(gender) %>% summarize(avg=mean(age),total=n())
```

```
## # A tibble: 2 x 3
##   gender   avg total
##   <chr>  <dbl> <int>
## 1 FEMALE    64     2
## 2 MALE      70     3
```

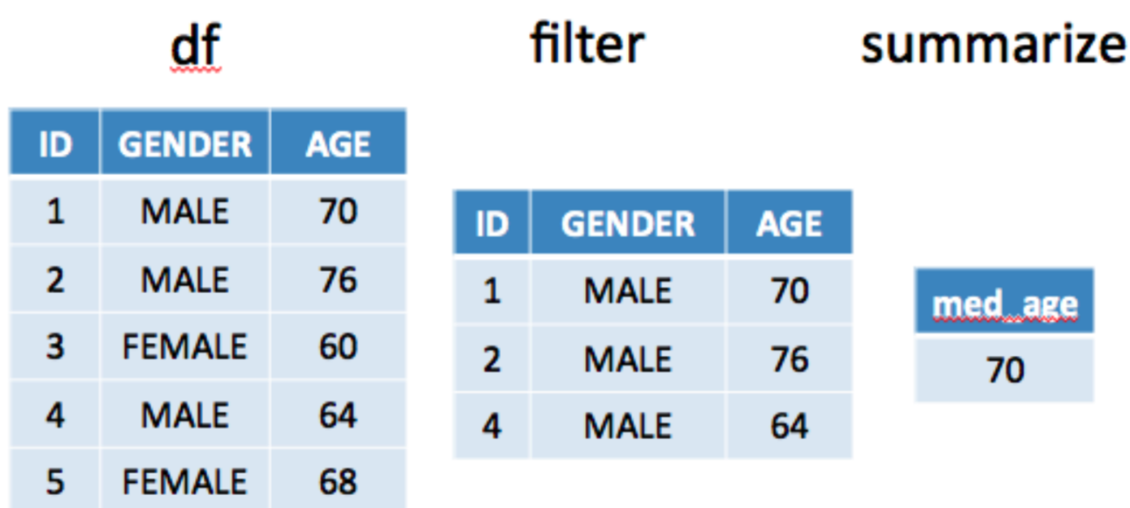
```r
# Same as the following but the pipes don't require you to "commit"
# With the following, you have to know in advance what you want to do

summarize(group_by(df,gender), avg=mean(age))
```

```
## # A tibble: 2 x 2
##   gender   avg
##   <chr>  <dbl>
## 1 FEMALE    64
## 2 MALE      70
```

What is the median age of all males ?

```r
df %>% filter(gender == "MALE") %>% summarize(med_age=median(age))
```

```
##   med_age
## 1      70
```



It should be observed that if you want to save the results of some sequence of commands that you will need to use the "<-" operator. Using the previous example we could the following to save our result.

```r
results <- df %>%
  filter(gender == "MALE") %>%
  summarize(med_age=median(age))
```

Using the built in mtcars data frame filter out records where the wt is greater than 3.3 tons.

Then create a column called ab_be (Y or N) that indicates whether that observation's mpg is greater (or not) than the average mpg for the filtered set.

Then present the average mpg for each group

```
mtcars %>%
  filter(wt > 3.3)  %>%
  mutate(ab_be=ifelse(mpg > mean(mpg),"Y","N")   ) %>%
  group_by(ab_be) %>%
  summarize(mean_mpg=mean(mpg))
```

```
## # A tibble: 2 x 2
##   ab_be mean_mpg
##   <chr>    <dbl>
## 1 N         13.8
## 2 Y         18.1
```

Using the built in mtcars dataframe filter out records where the wt is greater than 3.3 tons.

```
mtcars %>% filter(wt > 3.3)
```

```
##                      mpg   cyl  disp  hp drat    wt  qsec vs am gear carb
## Hornet Sportabout   18.7 Cyl_8 360.0 175 3.15 3.440 17.02  0  A    3    2
## Valiant             18.1 Cyl_6 225.0 105 2.76 3.460 20.22  1  A    3    1
## Duster 360          14.3 Cyl_8 360.0 245 3.21 3.570 15.84  0  A    3    4
## Merc 280            19.2 Cyl_6 167.6 123 3.92 3.440 18.30  1  A    4    4
## Merc 280C           17.8 Cyl_6 167.6 123 3.92 3.440 18.90  1  A    4    4
## Merc 450SE          16.4 Cyl_8 275.8 180 3.07 4.070 17.40  0  A    3    3
## Merc 450SL          17.3 Cyl_8 275.8 180 3.07 3.730 17.60  0  A    3    3
## Merc 450SLC         15.2 Cyl_8 275.8 180 3.07 3.780 18.00  0  A    3    3
## Cadillac Fleetwood  10.4 Cyl_8 472.0 205 2.93 5.250 17.98  0  A    3    4
## Lincoln Continental 10.4 Cyl_8 460.0 215 3.00 5.424 17.82  0  A    3    4
## Chrysler Imperial   14.7 Cyl_8 440.0 230 3.23 5.345 17.42  0  A    3    4
## Dodge Challenger    15.5 Cyl_8 318.0 150 2.76 3.520 16.87  0  A    3    2
## AMC Javelin         15.2 Cyl_8 304.0 150 3.15 3.435 17.30  0  A    3    2
## Camaro Z28          13.3 Cyl_8 350.0 245 3.73 3.840 15.41  0  A    3    4
## Pontiac Firebird    19.2 Cyl_8 400.0 175 3.08 3.845 17.05  0  A    3    2
## Maserati Bora       15.0 Cyl_8 301.0 335 3.54 3.570 14.60  0  M    5    8
```

Create a column called ab_be (Y or N) that indicates whether that observation's mpg is greater (or not) than the average mpg for the filtered set.

```
mtcars %>% filter(wt > 3.3) %>%
           mutate(ab_be=ifelse(mpg > mean(mpg),"Y","N"))
```

```
##                      mpg   cyl  disp  hp drat    wt  qsec vs am gear carb ab_be
## Hornet Sportabout   18.7 Cyl_8 360.0 175 3.15 3.440 17.02  0  A    3    2     Y
## Valiant             18.1 Cyl_6 225.0 105 2.76 3.460 20.22  1  A    3    1     Y
## Duster 360          14.3 Cyl_8 360.0 245 3.21 3.570 15.84  0  A    3    4     N
## Merc 280            19.2 Cyl_6 167.6 123 3.92 3.440 18.30  1  A    4    4     Y
## Merc 280C           17.8 Cyl_6 167.6 123 3.92 3.440 18.90  1  A    4    4     Y
## Merc 450SE          16.4 Cyl_8 275.8 180 3.07 4.070 17.40  0  A    3    3     Y
## Merc 450SL          17.3 Cyl_8 275.8 180 3.07 3.730 17.60  0  A    3    3     Y
## Merc 450SLC         15.2 Cyl_8 275.8 180 3.07 3.780 18.00  0  A    3    3     N
## Cadillac Fleetwood  10.4 Cyl_8 472.0 205 2.93 5.250 17.98  0  A    3    4     N
## Lincoln Continental 10.4 Cyl_8 460.0 215 3.00 5.424 17.82  0  A    3    4     N
```

```
## Chrysler Imperial   14.7 Cyl_8 440.0 230 3.23 5.345 17.42  0  A    3    4      N
## Dodge Challenger    15.5 Cyl_8 318.0 150 2.76 3.520 16.87  0  A    3    2      N
## AMC Javelin         15.2 Cyl_8 304.0 150 3.15 3.435 17.30  0  A    3    2      N
## Camaro Z28          13.3 Cyl_8 350.0 245 3.73 3.840 15.41  0  A    3    4      N
## Pontiac Firebird    19.2 Cyl_8 400.0 175 3.08 3.845 17.05  0  A    3    2      Y
## Maserati Bora       15.0 Cyl_8 301.0 335 3.54 3.570 14.60  0  M    5    8      N
```

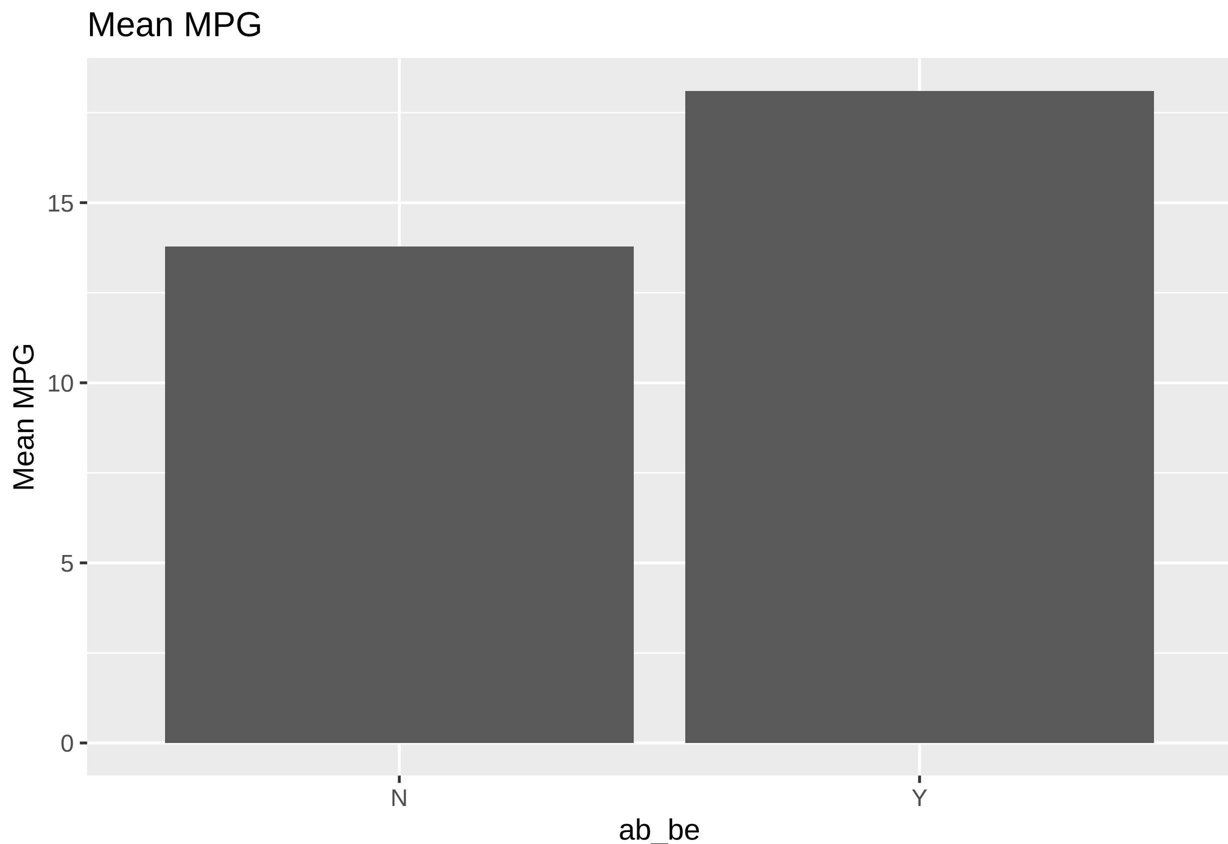
Then present the average mpg for each group as defined by ab_be

```
mtcars %>%
         filter(wt > 3.3)  %>%
         mutate(ab_be=ifelse(mpg > mean(mpg),"Y","N")) %>%
         group_by(ab_be) %>%
         summarize(mean_mpg=mean(mpg))
```

```
## # A tibble: 2 x 2
##   ab_be mean_mpg
##   <chr>    <dbl>
## 1 N         13.8
## 2 Y         18.1
```

This could be then "piped" into the input of the ggplot command to plot a corresponding bar chart. Both ggplot and dplyr are part of the tidyverse which means that the two packages "talk" to each other well.

```
library(ggplot2)
mtcars %>% filter(wt > 3.3)  %>%
         mutate(ab_be=ifelse(mpg > mean(mpg),"Y","N")  ) %>%
         group_by(ab_be) %>% summarize(mean_mpg=mean(mpg)) %>%
         ggplot(aes(x=ab_be,y=mean_mpg)) + geom_bar(stat="identity") +
         ggtitle("Mean MPG") + labs(x = "ab_be", y = "Mean MPG")
```



Other activities are possible

```
mtcars %>% sample_n(2) # Sample 2 records from a data frame
```

```
##                      mpg   cyl disp  hp drat    wt  qsec vs am gear carb
## Lincoln Continental 10.4 Cyl_8  460 215 3.00 5.424 17.82  0  A    3    4
## Hornet Sportabout   18.7 Cyl_8  360 175 3.15 3.440 17.02  0  A    3    2
```

Note that we could do something like:

```
mtcars %>% group_by(cyl) %>% sample_n(2)
```

```
## # A tibble: 6 x 11
## # Groups:   cyl [3]
##     mpg cyl    disp    hp  drat    wt  qsec    vs am     gear  carb
##   <dbl> <fct> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <fct> <dbl> <dbl>
## 1  21.5 Cyl_4  120.    97  3.7   2.46  20.0     1 A         3     1
## 2  27.3 Cyl_4   79     66  4.08  1.94  18.9     1 M         4     1
## 3  21   Cyl_6  160    110  3.9   2.62  16.5     0 M         4     4
## 4  19.2 Cyl_6  168.   123  3.92  3.44  18.3     1 A         4     4
## 5  13.3 Cyl_8  350    245  3.73  3.84  15.4     0 A         3     4
## 6  10.4 Cyl_8  472    205  2.93  5.25  18.0     0 A         3     4
```

Wages

Remember last week when we spent a lot of time with the wages data frame as part of our introduction to ggplot2. Let's see how we might work with this using dplyr.

```r
url <- "https://raw.githubusercontent.com/pittardsp/bios545r_spring_2018/master/SUPPORT/wage.csv"

library(readr)

wage <- read_csv(url)
```

```
## Rows: 3000 Columns: 11

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (8): sex, maritl, race, education, region, jobclass, health, health_ins
## dbl (3): year, age, wage

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
glimpse(wage)
```

```
## Rows: 3,000
## Columns: 11
## $ year       <dbl> 2006, 2004, 2003, 2003, 2005, 2008, 2009, 2008, 2006, 2004,~
## $ age        <dbl> 18, 24, 45, 43, 50, 54, 44, 30, 41, 52, 45, 34, 35, 39, 54,~
## $ sex        <chr> "Male", "Male", "Male", "Male", "Male", "Male", "Male", "Ma~
## $ maritl     <chr> "Never Married", "Never Married", "Married", "Married", "Di~
## $ race       <chr> "White", "White", "White", "Asian", "White", "White", "Othe~
## $ education  <chr> "< HS Grad", "College Grad", "Some College", "College Grad"~
## $ region     <chr> "Middle Atlantic", "Middle Atlantic", "Middle Atlantic", "M~
## $ jobclass   <chr> "Industrial", "Information", "Industrial", "Information", "~
## $ health     <chr> "<=Good", ">=Very Good", "<=Good", ">=Very Good", "<=Good",~
## $ health_ins <chr> "No", "No", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes"~
## $ wage       <dbl> 75.04315, 70.47602, 130.98218, 154.68529, 75.04315, 127.115~
```

```r
sapply(wage,n_distinct)
```

```
##       year        age        sex     maritl       race  education     region
##          7         61          1          5          4          5          1
##   jobclass     health health_ins       wage
##          2          2          2        508
```

```r
summary(wage)
```

```
##       year           age             sex               maritl
##  Min.   :2003   Min.   :18.00   Length:3000        Length:3000
##  1st Qu.:2004   1st Qu.:33.75   Class :character   Class :character
##  Median :2006   Median :42.00   Mode  :character   Mode  :character
##  Mean   :2006   Mean   :42.41
##  3rd Qu.:2008   3rd Qu.:51.00
##  Max.   :2009   Max.   :80.00
##      race            education            region            jobclass
##  Length:3000        Length:3000        Length:3000        Length:3000
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##     health           health_ins             wage
```

```
##  Length:3000        Length:3000        Min.   : 20.09
##  Class :character   Class :character   1st Qu.: 85.38
##  Mode  :character   Mode  :character   Median :104.92
##                                        Mean   :111.70
##                                        3rd Qu.:128.68
##                                        Max.   :318.34
```

Do some basic aggregations:

```
# What is the average wage across JobClass ?

wage %>% group_by(jobclass) %>% summarize(avg_salary=mean(wage))
```

```
## # A tibble: 2 x 2
##   jobclass    avg_salary
##   <chr>            <dbl>
## 1 Industrial        103.
## 2 Information       121.
```

```
# What is the average wage across healt ins ? Present the result in descending
# order of salary
#race
wage %>% group_by(health_ins) %>%
  summarize(mean_salary=mean(wage)) %>%
  arrange(desc(mean_salary))
```

```
## # A tibble: 2 x 2
##   health_ins mean_salary
##   <chr>            <dbl>
## 1 Yes              120.
## 2 No                92.3
```

What is the average wage across health insurance and education ? Present the result in descending order of salary and show only the top 3 mean salaries

```
wage %>% group_by(health_ins,education) %>%
  summarize(mean_salary=mean(wage)) %>%
  arrange(mean_salary)
```

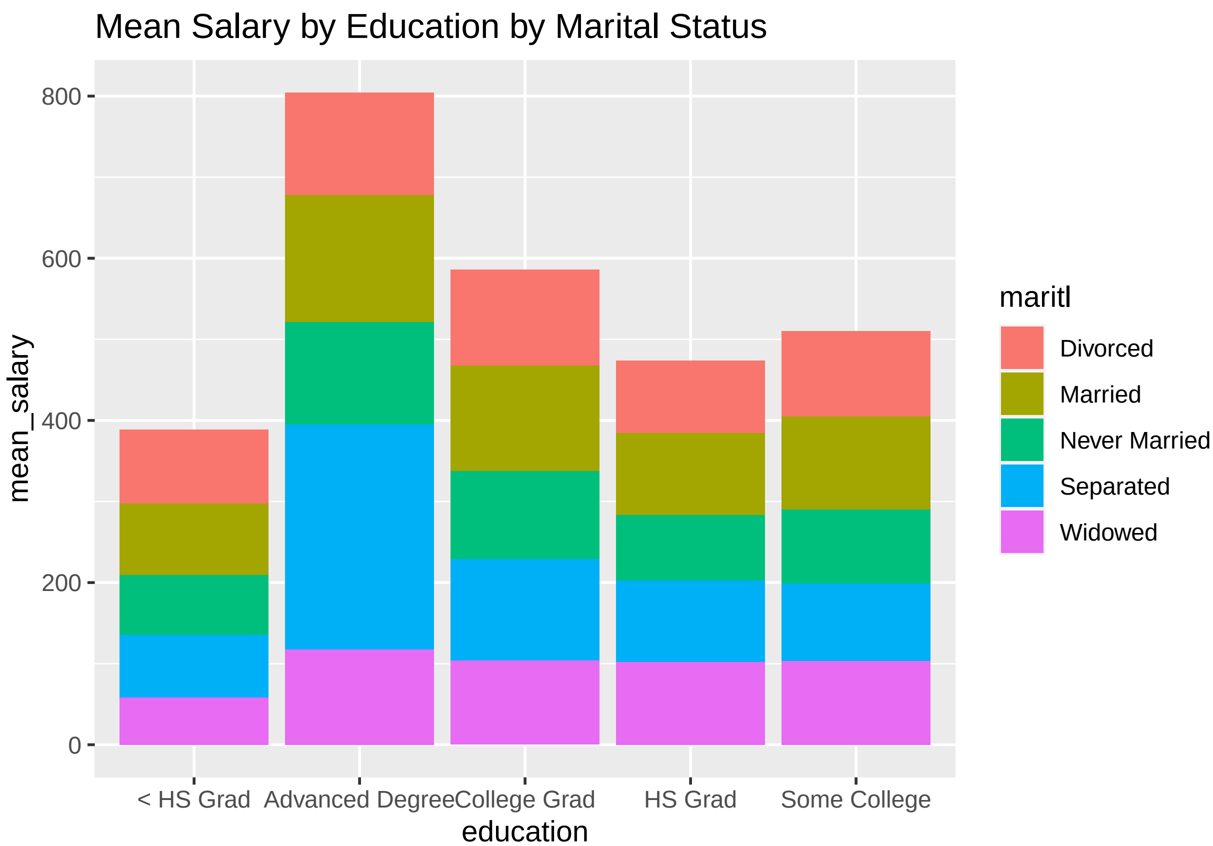
```
## `summarise()` has grouped output by 'health_ins'. You can override using the `.groups` argument.
```

```
## # A tibble: 10 x 3
## # Groups:   health_ins [2]
##    health_ins education       mean_salary
##    <chr>      <chr>                 <dbl>
##  1 No         < HS Grad              75.9
##  2 No         HS Grad                84.4
##  3 Yes        < HS Grad              93.6
##  4 No         Some College           95.1
##  5 Yes        HS Grad               102.
##  6 No         College Grad          103.
##  7 Yes        Some College          113.
##  8 Yes        College Grad          131.
##  9 No         Advanced Degree       132.
## 10 Yes        Advanced Degree       155.
```

Create a graph of the top three mean salaries by Education level as grouped by Marital Status ? Notice anything interesting ?

```
wage %>% group_by(education,maritl) %>%
  summarize(mean_salary=mean(wage)) %>%
  arrange(desc(mean_salary)) %>%
  ggplot(aes(x=education,y=mean_salary,fill=maritl)) + geom_bar(stat="identity") +
  ggtitle("Mean Salary by Education by Marital Status")
```

```
## `summarise()` has grouped output by 'education'. You can override using the `.groups` argument.
```



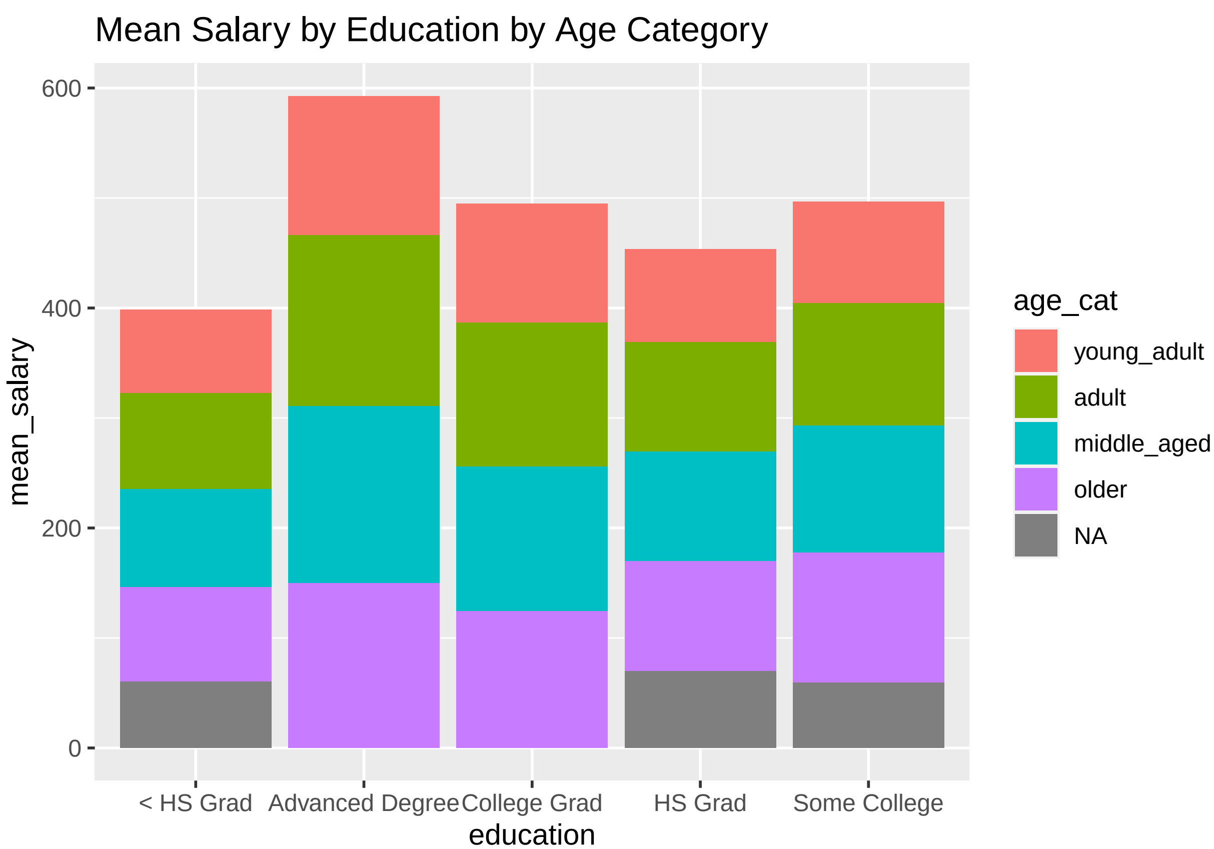
Let's group the ages into 4 categories based on where the age is "binned" into the percentiles.

```
labs <- c("young_adult","adult","middle_aged","older")

wage <- wage %>% mutate(age_cat=cut(age,labels=labs,breaks=quantile(age)))
```

Now let's look at the barplot of mean salary by education and age_cat

```
wage %>% group_by(education,age_cat) %>%
  summarize(mean_salary=mean(wage)) %>%
  arrange(desc(mean_salary)) %>%
  ggplot(aes(x=education,y=mean_salary,fill=age_cat)) + geom_bar(stat="identity") +
  ggtitle("Mean Salary by Education by Age Category")
```

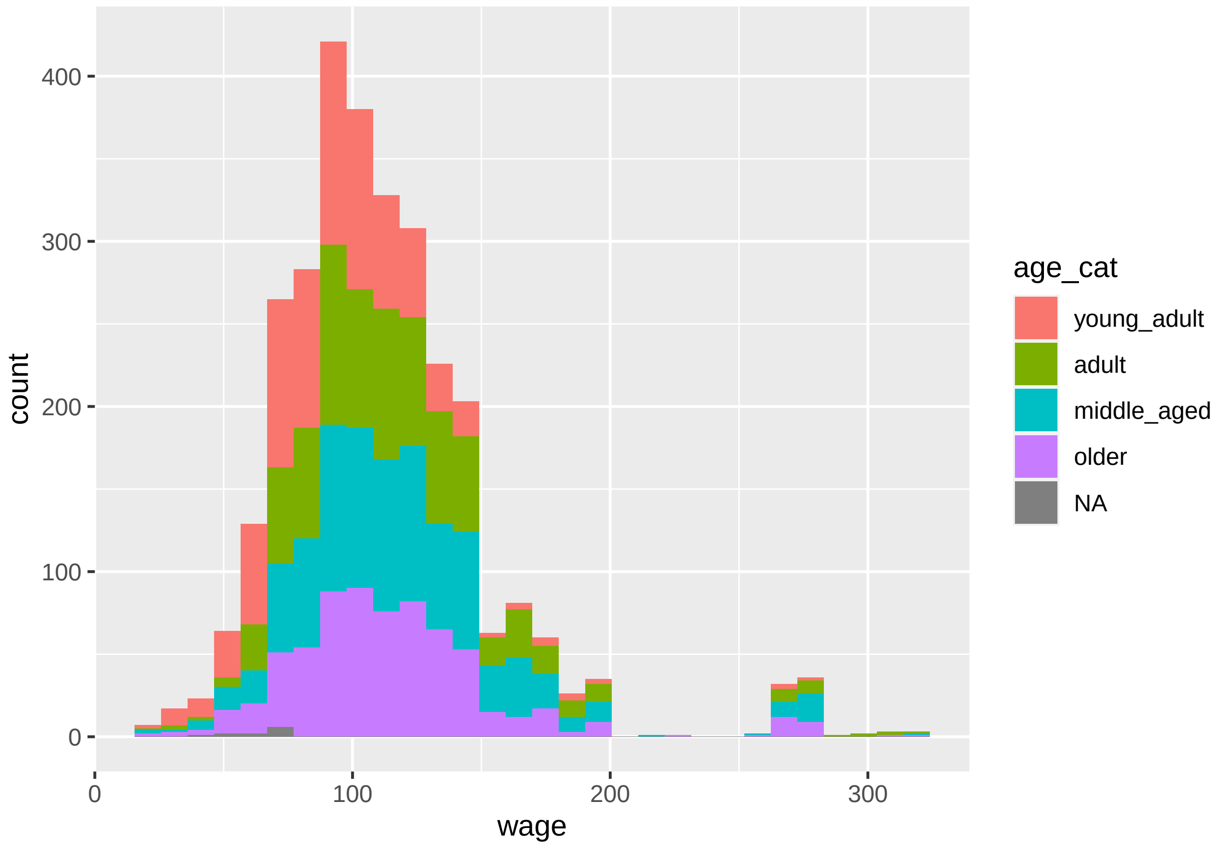
```
## `summarise()` has grouped output by 'education'. You can override using the `.groups` argument.
```



Let's look at the distribution of salary as grouped by age category

```
wage %>% ggplot(aes(x=wage)) + geom_histogram(aes(fill=age_cat))
```

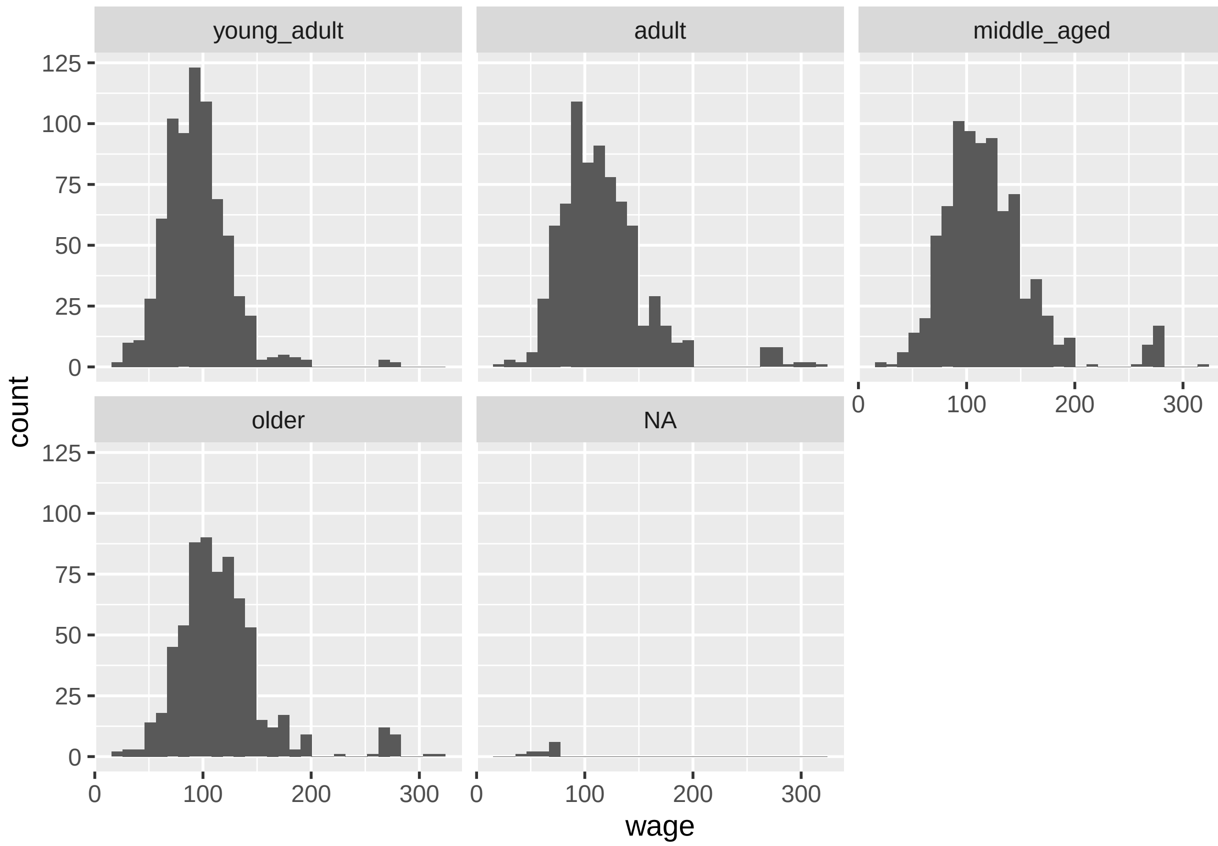
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Let's look at the distribution of salary as grouped by age category but let's use facets this time

```
wage %>% ggplot(aes(x=wage)) +
  geom_histogram() +
  facet_wrap(~age_cat)
```

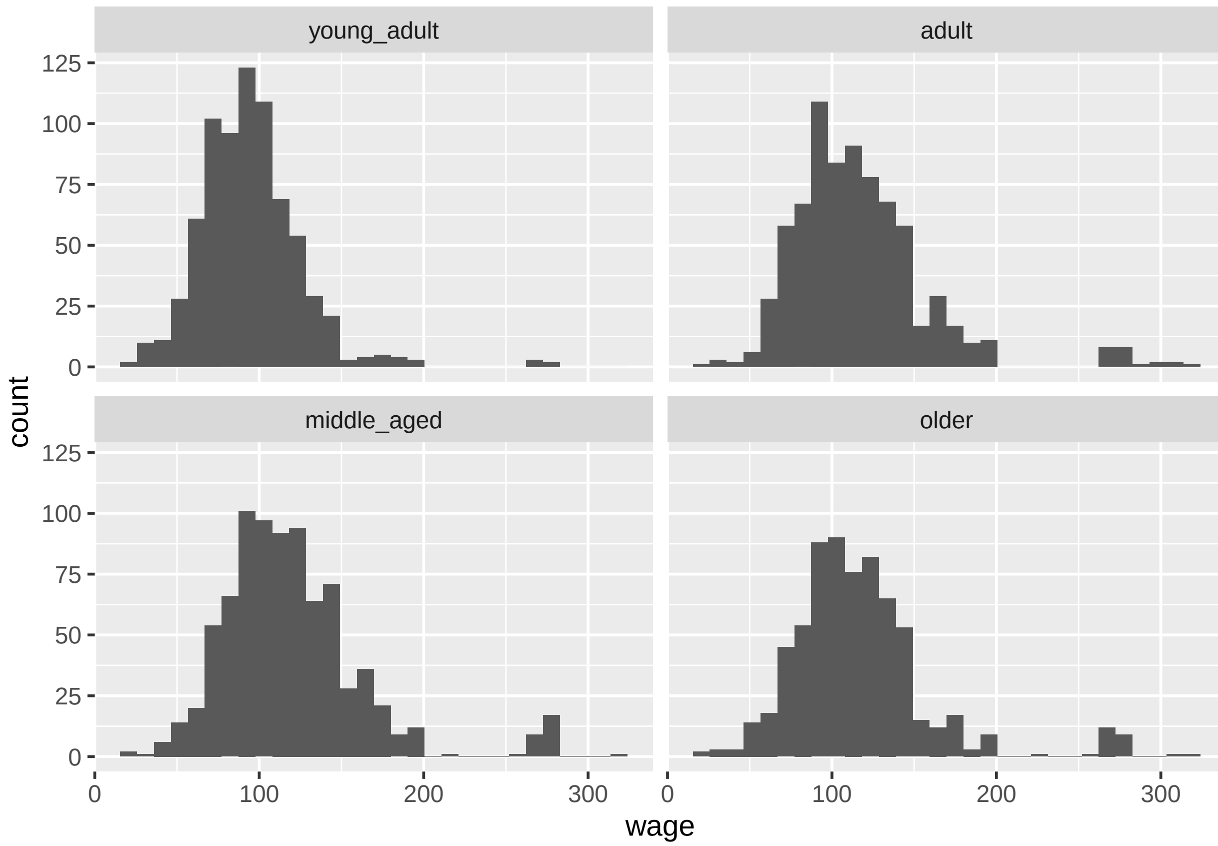
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



It might be better to first filter out any rows with missing data

```
wage %>% na.omit() %>%
  ggplot(aes(x=wage)) +
  geom_histogram() +
  facet_wrap(~age_cat)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Window Functions

There are functions that let us look over a range of values for a column of a data frame. For example, consider the following data:

```
months1 <- factor(month.abb,ordered=TRUE,levels=month.abb)
set.seed(123)
amount <- 1:12 + sample(c(-.5,.5),1)
(figures <- data.frame(months=months1,amount=amount))
```

```
##    months amount
## 1     Jan    0.5
## 2     Feb    1.5
## 3     Mar    2.5
## 4     Apr    3.5
## 5     May    4.5
## 6     Jun    5.5
## 7     Jul    6.5
## 8     Aug    7.5
## 9     Sep    8.5
## 10    Oct    9.5
## 11    Nov   10.5
## 12    Dec   11.5
```

What if we wanted to find the percent change in the **amount** variable from one month to the next ? A possible formula for computing the change might look like:

change = (current amount - previous amount) / previous amount

Let's write a function to implement this:

```
amount_change <- function(df=figures,col=2) {
# function to compute percent change in amount
# INPUT: df - a data frame
#        col - the column number corresponding to the amount
#
# OUTPUT: a dataframe (the same as df) with the change column
#         added to it
#

  # Check to see if specified column is numeric
  if (!is.numeric(df[,col])) {
    stop("Specified column is not numeric. Bye!")
  }
  # Setup an empty vector
  change <- vector()

  for (ii in 1:nrow(df)) {
    if (ii == 1) {
       change[ii] <- NA
    } else {
       change[ii] <- (df[ii,col] - df[(ii-1),col])/df[(ii-1),col]
    }
  } # end for loop`

  return(cbind(df,change))
} # end function
```

```
amount_change()
```

```
##    months amount    change
## 1     Jan    0.5        NA
## 2     Feb    1.5 2.0000000
## 3     Mar    2.5 0.6666667
## 4     Apr    3.5 0.4000000
## 5     May    4.5 0.2857143
## 6     Jun    5.5 0.2222222
## 7     Jul    6.5 0.1818182
## 8     Aug    7.5 0.1538462
## 9     Sep    8.5 0.1333333
## 10    Oct    9.5 0.1176471
## 11    Nov   10.5 0.1052632
## 12    Dec   11.5 0.0952381
```

But that was a lot of work wasn't it ? What about using the **lag** function from dplyr which allows us to refer to a previous value in a vector of information.

```
figures$amount
```

```
##  [1]  0.5  1.5  2.5  3.5  4.5  5.5  6.5  7.5  8.5  9.5 10.5 11.5
```

```
lag(figures$amount)
```

```
##  [1]   NA  0.5  1.5  2.5  3.5  4.5  5.5  6.5  7.5  8.5  9.5 10.5
```

So using this information along with a formula for computing the change using the **lag** function.

```
figures %>% mutate(change = (amount-lag(amount))/lag(amount))
```

```
##    months amount    change
## 1     Jan    0.5        NA
## 2     Feb    1.5 2.0000000
## 3     Mar    2.5 0.6666667
## 4     Apr    3.5 0.4000000
## 5     May    4.5 0.2857143
## 6     Jun    5.5 0.2222222
## 7     Jul    6.5 0.1818182
## 8     Aug    7.5 0.1538462
## 9     Sep    8.5 0.1333333
## 10    Oct    9.5 0.1176471
## 11    Nov   10.5 0.1052632
## 12    Dec   11.5 0.0952381
```