Lesson 5

EDA Multivariate Data

```
# Loading libraries
library(ggplot2)
library(tidyr)
suppressMessages(library(dplyr))
suppressMessages(library(gridExtra))
suppressMessages(library(GGally))
```

```
pf <- read.delim("pseudo_facebook.tsv")
head(pf)
```

```
##    userid age dob_day dob_year dob_month gender tenure friend_count
## 1 2094382  14      19     1999        11   male    266            0
## 2 1192601  14       2     1999        11 female      6            0
## 3 2083884  14      16     1999        11   male     13            0
## 4 1203168  14      25     1999        12 female     93            0
## 5 1733186  14       4     1999        12   male     82            0
## 6 1524765  14       1     1999        12   male     15            0
##   friendships_initiated likes likes_received mobile_likes
## 1                     0     0              0            0
## 2                     0     0              0            0
## 3                     0     0              0            0
## 4                     0     0              0            0
## 5                     0     0              0            0
## 6                     0     0              0            0
##   mobile_likes_received www_likes www_likes_received
## 1                     0         0                  0
## 2                     0         0                  0
## 3                     0         0                  0
## 4                     0         0                  0
## 5                     0         0                  0
## 6                     0         0                  0
```

Third Qualitative Variable

You can include multiple variables to split the data frame when using group_by() function in the dplyr package.

new_groupings <- group_by(data, variable1, variable2)

OR using chained commands...

```
new_data_frame <- data_frame %>%
  group_by(variable1, variable2) %>%
```

dplyr issues when using group_by(multiple variables)

Repeated use of summarise() and group_by(): The summarize function will automatically remove one level

of grouping (the last group it collapsed).

```
data("mtcars")
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```
df1 <- mtcars %>%
  group_by(cyl, gear) %>%
  summarise(newvar = sum(wt))

head(df1)
```

```
## # A tibble: 6 x 3
## # Groups:   cyl [2]
##     cyl  gear newvar
##   <dbl> <dbl>  <dbl>
## 1     4     3  2.465
## 2     4     4 19.025
## 3     4     5  3.653
## 4     6     3  6.675
## 5     6     4 12.375
## 6     6     5  2.770
```

Then say I want to further summarise this dataframe. With ddply, it'd be straightforward, but when I try to do with with dplyr, it's not actually "grouping by": Still yields an ungrouped output:

```
df2 <- df1 %>%
  group_by(cyl) %>%
  mutate(newvar2 = newvar + 5)

head(df2)
```

```
## # A tibble: 6 x 4
## # Groups:   cyl [2]
##     cyl  gear newvar newvar2
##   <dbl> <dbl>  <dbl>   <dbl>
## 1     4     3  2.465   7.465
## 2     4     4 19.025  24.025
## 3     4     5  3.653   8.653
## 4     6     3  6.675  11.675
## 5     6     4 12.375  17.375
## 6     6     5  2.770   7.770
```

Taking Dickoa's answer one step further – as Hadley says **summarise peels off a single layer of grouping**. **It peels off grouping from the reverse order in which you applied** it so you can just use :

```r
df3 <- mtcars %>%
  group_by(cyl, gear) %>%
  summarise(newvar = sum(wt))

head(df3)
```

```
## # A tibble: 6 x 3
## # Groups:   cyl [2]
##     cyl  gear newvar
##   <dbl> <dbl>  <dbl>
## 1     4     3  2.465
## 2     4     4 19.025
## 3     4     5  3.653
## 4     6     3  6.675
## 5     6     4 12.375
## 6     6     5  2.770
```

```r
df4 <- df3 %>%
 group_by(cyl) %>%
 summarise(newvar2 = sum(newvar)+5)

head(df4)
```

```
## # A tibble: 3 x 2
##     cyl newvar2
##   <dbl>   <dbl>
## 1     4  30.143
## 2     6  26.820
## 3     8  60.989
```

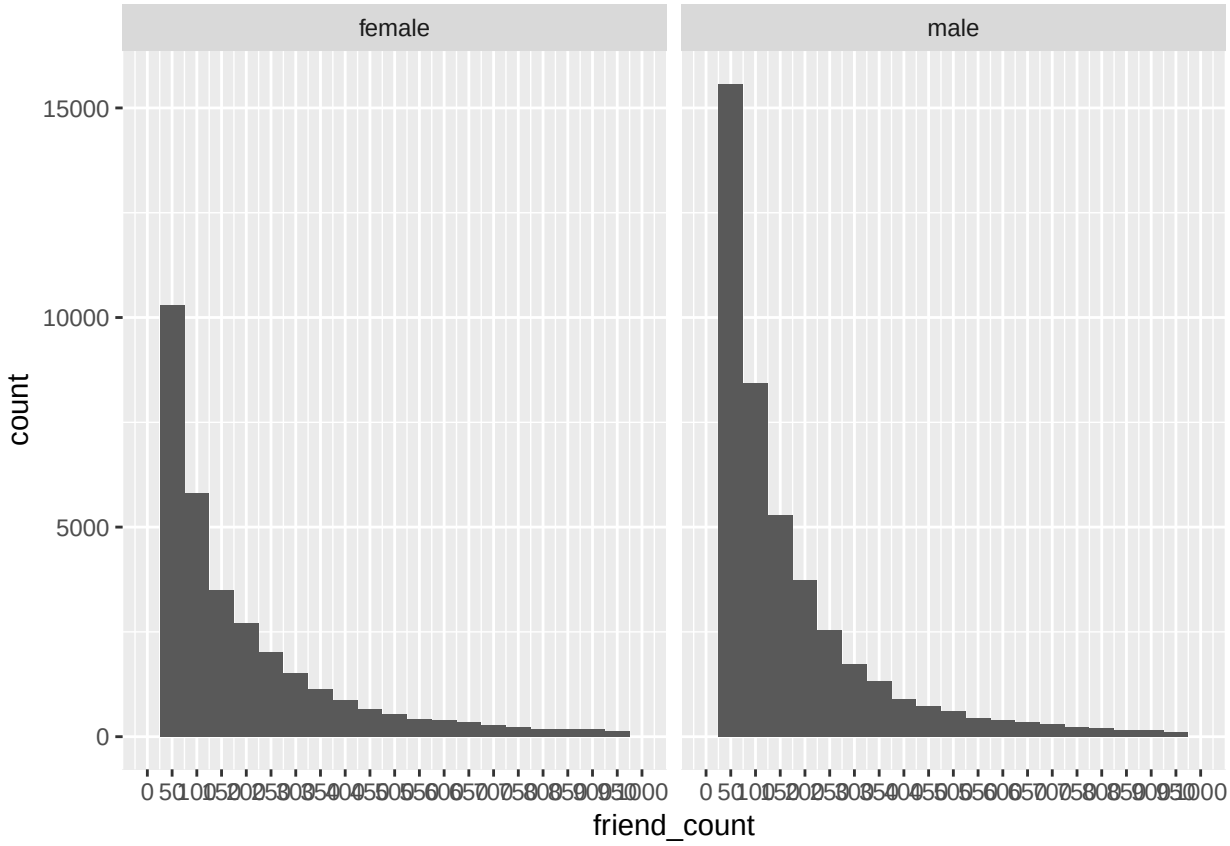
EDA : Third Qualitative Variable

Sometimes when we conduct exploratory data analysis we do reach dead ends.

Let's see if we can get any further in examining our relationship between friends count and age by adding a third variable. Previously we noted that female users have more friends on average than male users. And, we might wonder, is this just because female users have a different age distribution? Or, maybe conditional on age, the differences are actually larger.

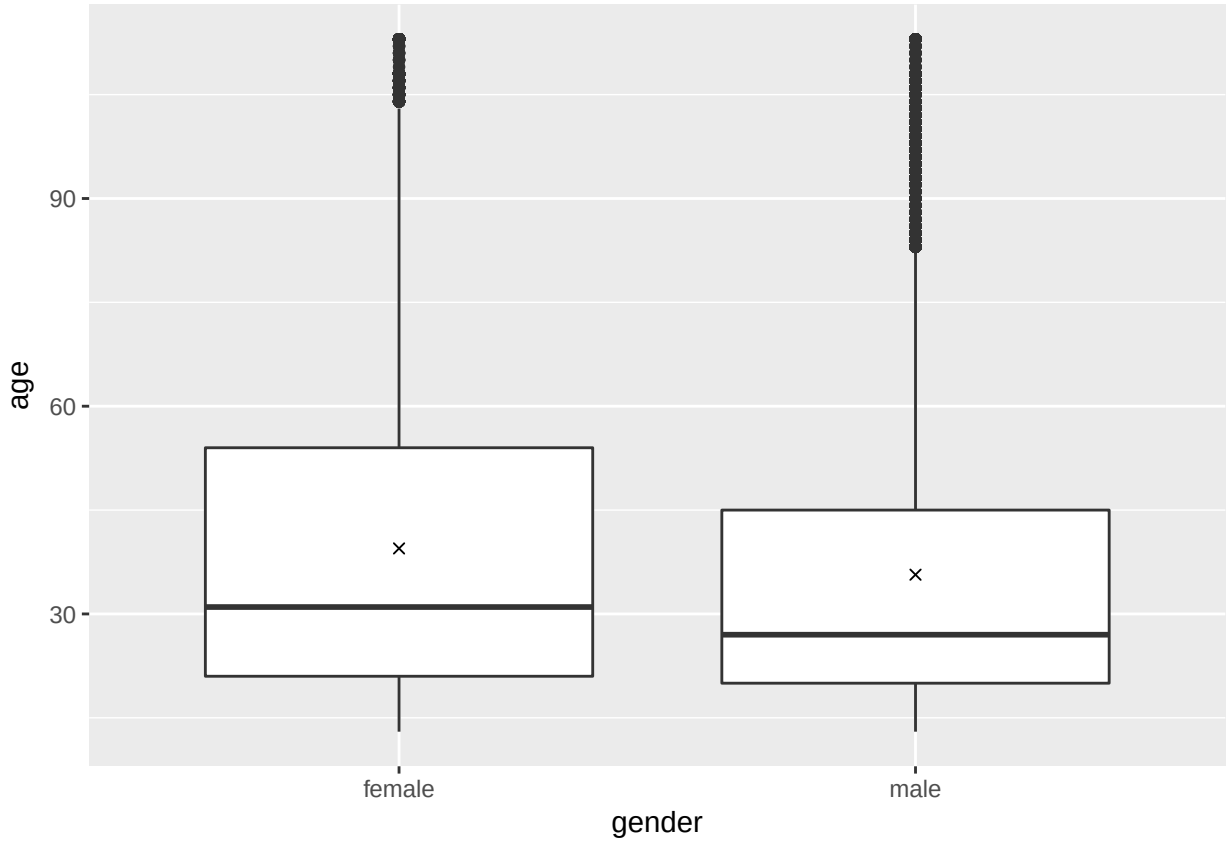
```r
ggplot(aes(x = friend_count), data = subset(pf, !is.na(gender))) +
  geom_histogram(binwidth=50) +
  scale_x_continuous(limits = c(0, 1000), breaks = seq(0, 1000, 50)) +
  facet_wrap(~gender)
```

```
## Warning: Removed 2949 rows containing non-finite values (stat_bin).
```



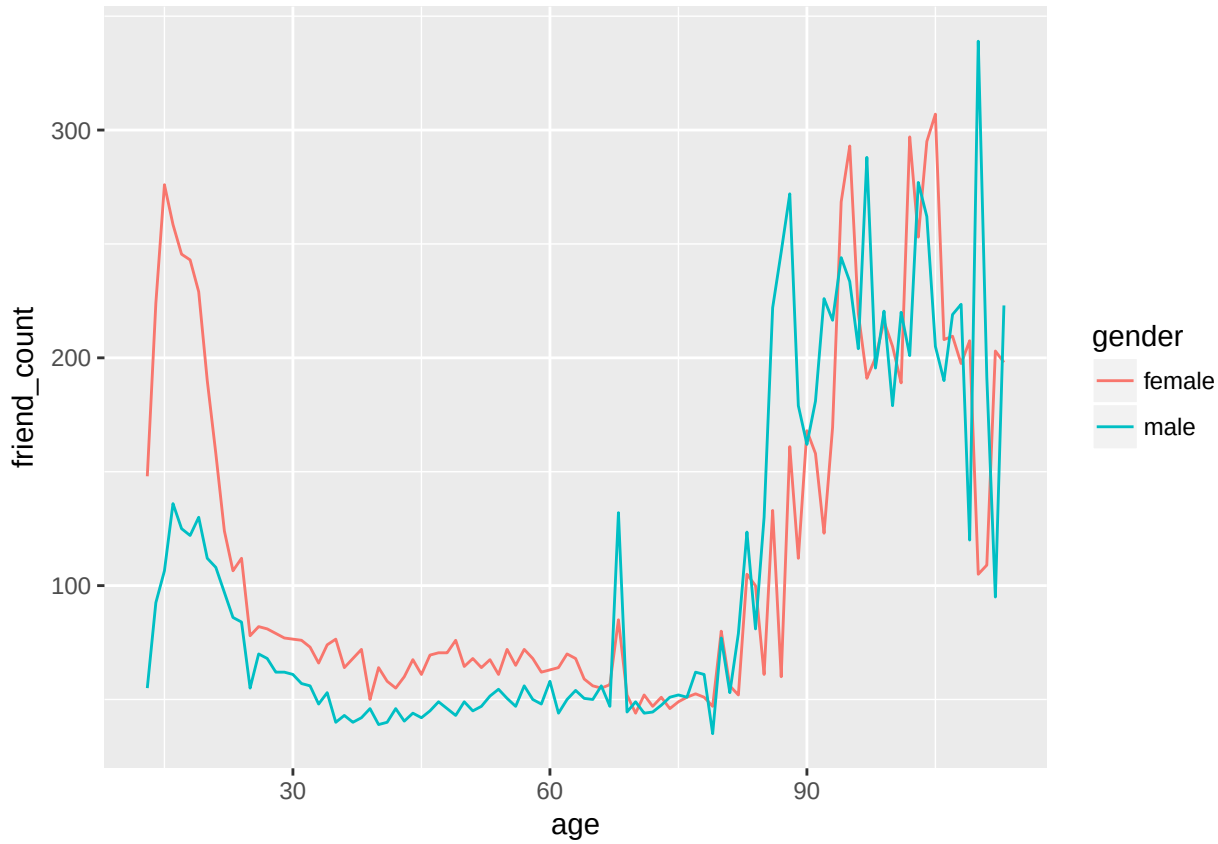
Here's a box plot of ages by gender. Now, I'm going to add the mean for each gender to the box plots, using stat summary. Here we can see the averages marked by an x since I used shape=4.

```
ggplot(data = subset(pf, !is.na(gender)), aes(x = gender, y = age))+
  geom_boxplot()+
  stat_summary(fun.y = mean, geom = 'point', shape = 4)
```



Since male users are a bit younger, we might actually think a simple male to female comparison doesn't capture their substantial differences in friend count. Let's look at median friend count by age and gender instead.

```
ggplot(data = subset(pf,!is.na(gender)), aes(x=age, y = friend_count))+
  geom_line(aes(color = gender), stat = 'summary', fun.y = median)
```



When I run this code, we can see that nearly everywhere the median friend count is larger for women than it is for men. Now there are some exceptions, and this includes these noisy estimates for our very old users. Now, I'm using old with quotation marks here, since we're not really confident about these reported ages. And notice that users reporting to be of reported gender.

You're going to reproduce the same plot, but first let's see if you can create the summary data to create it. Recall that we can produce the same summary data underlying this plot by using the dplyr package. We can divide the data by age and gender and then compute the median and mean friend count for each sub-group. In this next program assignment you're going to do just that by using the group buy, summarize, and arrange functions from the dplyr package.

```
# Write code to create a new data frame, called 'pf.fc_by_age_gender', that contains information on eac
The data frame should contain the following variables:
   mean_friend_count,
   median_friend_count,
   n (the number of users in each age and gender grouping)

Here is an example of the structure of your data frame. Your data values will be different. Note that i

#   age gender mean_friend_count median_friend_count    n
# 1  13 female          247.2953                150  207
# 2  13   male          184.2342                 61  265
# 3  14 female          329.1938                245  834
# 4  14   male          157.1204                 88 1201
```

```r
pf.fc_by_age_gender <- pf %>%
  filter(!is.na(gender)) %>%
  group_by(age, gender) %>%
```

```
  summarise(mean_friend_count = mean(friend_count),
            median_friend_count = median(friend_count),
            n = n()
  ) %>%
  ungroup()

head(pf.fc_by_age_gender)
```

```
## # A tibble: 6 x 5
##     age gender mean_friend_count median_friend_count     n
##   <int> <fctr>             <dbl>               <dbl> <int>
## 1    13 female          259.1606               148.0   193
## 2    13   male          102.1340                55.0   291
## 3    14 female          362.4286               224.0   847
## 4    14   male          164.1456                92.5  1078
## 5    15 female          538.6813               276.0  1139
## 6    15   male          200.6658               106.5  1478
```

```
dd <- pf.fc_by_age_gender %>%
  group_by(age) %>%
  summarise(mean_friend_count_age = mean(mean_friend_count),
            median_friend_count_age = median(median_friend_count),
            n=n())

head(dd)
```

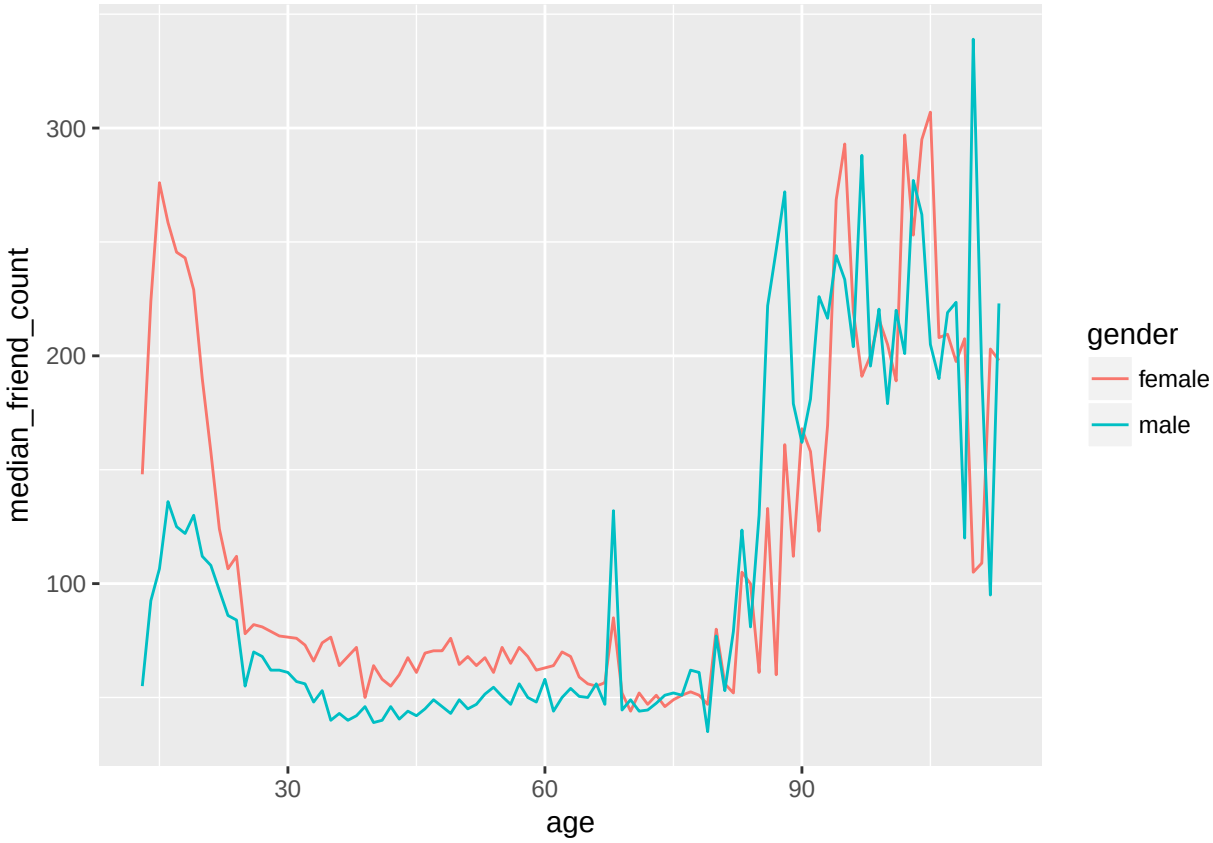
```
## # A tibble: 6 x 4
##     age mean_friend_count_age median_friend_count_age     n
##   <int>                 <dbl>                   <dbl> <int>
## 1    13              180.6473                  101.50     2
## 2    14              263.2871                  158.25     2
## 3    15              369.6735                  191.25     2
## 4    16              379.5947                  197.25     2
## 5    17              387.7434                  185.25     2
## 6    18              357.9489                  182.50     2
```

Now, summarize will remove one layer of grouping when it runs, so we'll remove the gender layer. So, we need to run ungroup one more time to remove the age layer and finally I'll arrange my data frame by age.

Plotting Conditional Summaries

Notes: construct this plot that shows the median friend count for each gender as age increases.

```
ggplot(data = pf.fc_by_age_gender, aes(x = age, y = median_friend_count, group = gender ,color = gender)
  geom_line()
```



Thinking in Ratios

Notes: This can be useful if we want to inspect these values, or carry out further operations to help us understand how the difference between male and female users varies with age. For example, looking at this plot, it seems like the gender difference is largest for our young users. It would be to put this in relative terms though.

So, let's answer a different question. Let's answer the question, how many times more friends does the average female user have than the male user? Maybe, females have twice as many friends as male users, or maybe it's ten times as many friends.

Wide and Long Format

Notes: To answer that question, we need to rearrange our data a little bit. Right now, **our data is in long format**. We have many rows. And, **notice how that the variables that we grouped over, male and female, have been repeated**. They're **repeated for each year**. So let's do something else besides this long data format.

```
head(pf.fc_by_age_gender,10)
```

```
## # A tibble: 10 x 5
##      age gender mean_friend_count median_friend_count     n
```

```
##    <int> <fctr>            <dbl>              <dbl> <int>
##  1    13 female          259.1606              148.0   193
##  2    13   male          102.1340               55.0   291
##  3    14 female          362.4286              224.0   847
##  4    14   male          164.1456               92.5  1078
##  5    15 female          538.6813              276.0  1139
##  6    15   male          200.6658              106.5  1478
##  7    16 female          519.5145              258.5  1238
##  8    16   male          239.6748              136.0  1848
##  9    17 female          538.9943              245.5  1236
## 10    17   male          236.4924              125.0  2045
```

What we're going to do is convert it to a wide format. This new data frame will have one row for each age, and then we'll put the median friend count inside of males and females. Many times when computing with and exploring data, it's helpful to move back and forth between these different arrangements. To carry this out in R, we're going to be using the reshape2 package.

Reshaping Data

Notes:

```
#(install.packages('reshape2'))
suppressMessages(library(reshape2))
```

let's create a variable for a new data frame that will be in wide format.

Now, we're going to make use of the dcast function, which comes with the R shape two package. The letter d is used since we want the result to be a data frame. If we wanted an array or a matrix we could use a cast.

So, here's the data frame I want to modify and then this is where I'll enter my formula. Now, the first part of the formula, or the part to the left of the tilde sign, will list the variables I want to keep with an addition sign in between them.Here I just want to keep age.

On the right side of the tilde, we use the gender variable since we want male and female users to have their own columns for median friend count in the data frame. And finally, we set value dot var equal to median friend count because value dot var holds the key measurements or their values in our new data frame. And it looks like that I forgot quotes around this variable.

```
pf.fc_by_age_gender.wide <- dcast(pf.fc_by_age_gender,
                                  age~gender,
                                  value.var = 'median_friend_count')

head(pf.fc_by_age_gender.wide)
```

```
##   age female  male
## 1  13  148.0  55.0
## 2  14  224.0  92.5
## 3  15  276.0 106.5
## 4  16  258.5 136.0
## 5  17  245.5 125.0
## 6  18  243.0 122.0
```

```
pf.fc_by_age_gender.wide$ratio <- with(pf.fc_by_age_gender.wide, female/male)
head(pf.fc_by_age_gender.wide)
```

```
##   age female  male    ratio
```

```
## 1  13  148.0  55.0 2.690909
## 2  14  224.0  92.5 2.421622
## 3  15  276.0 106.5 2.591549
## 4  16  258.5 136.0 1.900735
## 5  17  245.5 125.0 1.964000
## 6  18  243.0 122.0 1.991803
```

```
# Using tidyr

pf.fc_by_age_gender.wide_dplyr <- spread(pf.fc_by_age_gender,
                                         key = 'gender',
                                         value = 'median_friend_count')

head(pf.fc_by_age_gender.wide_dplyr)
```

```
## # A tibble: 6 x 5
##     age mean_friend_count     n female  male
##   <int>             <dbl> <int>  <dbl> <dbl>
## 1    13          102.1340   291     NA  55.0
## 2    13          259.1606   193    148    NA
## 3    14          164.1456  1078     NA  92.5
## 4    14          362.4286   847    224    NA
## 5    15          200.6658  1478     NA 106.5
## 6    15          538.6813  1139    276    NA
```

The error above is due to not using subset of the df. (and selecting the desired column)

```
pf.fc_by_age_gender.wide_dplyr <- subset(pf.fc_by_age_gender[c('age','gender','median_friend_count')],
  spread(key = 'gender', value = 'median_friend_count') %>%
  mutate(ratio = female/male)

head(pf.fc_by_age_gender.wide_dplyr)
```

```
## # A tibble: 6 x 4
##     age female  male    ratio
##   <int>  <dbl> <dbl>    <dbl>
## 1    13  148.0  55.0 2.690909
## 2    14  224.0  92.5 2.421622
## 3    15  276.0 106.5 2.591549
## 4    16  258.5 136.0 1.900735
## 5    17  245.5 125.0 1.964000
## 6    18  243.0 122.0 1.991803
```

Ratio Plot

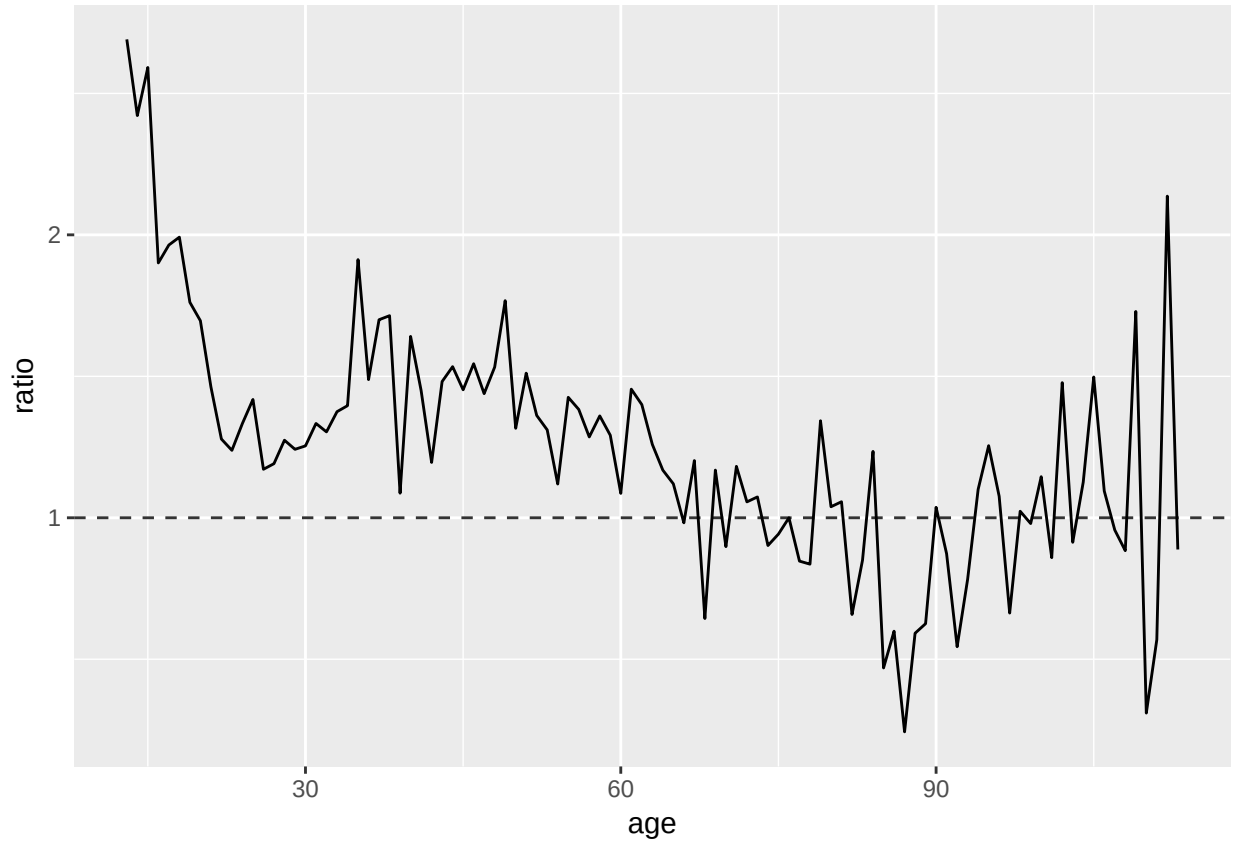
Notes: to plot the ratio of females to males to determine how many times more friends the average female user has, compared to the number of friends the average male user has. Think about what geom you should use.

Add a horizontal line to the plot with a y intercept of 1, which will be the base line. Look up the documentation for geom_hline to do that. Use the parameter linetype in geom_hline to make the line dashed.

```
The linetype parameter can take the values 0-6:
0 = blank, 1 = solid, 2 = dashed
3 = dotted, 4 = dotdash, 5 = longdash
```

6 = twodash

```
ggplot(data = pf.fc_by_age_gender.wide, aes(x = age, y = ratio))+
  geom_line()+
  geom_hline(yintercept = 1, linetype = 2, alpha=0.8 )
```



I'll assign age to the X parameter, and then I'll assign females divided by males, to the Y parameter. This will give me my ratio. And then I just need to make sure that I pass my newest data frame to data. We'll add a geome line to connect the points, and then we'll also add a horizontal line.

This geom_hline will take a couple parameters. We'll set the y-intercept to one, the alpha equal to 0.3, and the line type equal to two. Running this code, we get out ratio plot.

We can easily see that for very young users, the median female user has over two and a half times as many friends as the median male user. Clearly, it was helpful to condition on age in understanding the relationship of gender with friend count. This helped assure us this pattern is robust for users of many different ages. And **it also highlighted where this difference is most striking**.

Now, there are many processes that can produce this difference, including the biased distribution from which this pseudo Facebook data was generated. One idea which shows the complexity of interpretation here, is that people from particular countries who more recently joined Facebook are more likely to be male with lower friend counts.

Third Quantitative Variable

Notes:

In the previous examples, we were looking at our data Age and Friend Count across the categorical variable Gender. Usually, **color or shape tend to be aesthetics for representing such changes over a categorical variable**.

But **what if we looked at Age and Friend Count over, say, another numerical variable?**

For example we might notice that since users are likely to accumulate friends over time using Facebook that Facebook tenure is important for predicting friend count. Tenure or how many days since registering with Facebook is associated with age.

The first people to start using Facebook were college students as of 2004 and 2005.

One way to explore all four variables friend count, age, gender and tenure is using a two-dimensional display like a scatter plot. And we can bend one of the quantitative variables and compare those bends.

In this case, we can group users by the year that they joined. So let's create a new variable called year_joined in our data frame. This variable is going to hold the year that our users first joined Facebook. In our next program and exercise, you're going to create this variable, year_joined, and put it inside the data frame. You need to make use of the variable tenure and use 2014 as the reference year.

```
pf$year_joined <- 2014 - ceiling(pf$tenure/365)
```

Cut a Variable

Notes: We've got our new variable your joined so let's look at its summary it looks like most of our users joined in 2012 or 2013.

```
summary(pf$year_joined)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    2005    2012    2012    2012    2013    2014       2
```

Since the values for this variable are discreet and the range is pretty narrow I'm going to go ahead and table this variable as well.

```
table(pf$year_joined)
```

```
##
##  2005  2006  2007  2008  2009  2010  2011  2012  2013  2014
##     9    15   581  1507  4557  5448  9860 33366 43588    70
```

Here we can see the distributions of users and each year joined. Notice that there isn't much data here about early joiners

To increase the data we have in each tenure category, we can group some of these years together. The **cut function is often quite useful for making discrete variables from continuous or numerical ones, sometimes in combination with the function quantile**.

Your task is to cut the variable year joined to create four bins, or buckets of users. The bins will be from 2004 to 2009, 2009 to 2011, 2011 to 2012, and then 2012 to 2014.

The cut function

Create a new variable in the data frame called year_joined.bucket by using the cut function on the variable year_joined.

You need to create the following buckets for the new variable, year_joined.bucket

```
(2004, 2009]
(2009, 2011]
```

```
    (2011, 2012]
    (2012, 2014]
```

Note that a parenthesis means exclude the year and a bracket means include the year.

```
pf$year_joined.bucket <- cut(pf$year_joined, c(2004,2009,2011,2012,2014))
table(pf$year_joined.bucket, useNA = 'ifany')
```

```
##
## (2004,2009] (2009,2011] (2011,2012] (2012,2014]        <NA>
##        6669       15308       33366       43658           2
```

For this programming exercise, you need to create a variable called, year joined bucket. That bin together users depending on which year they joined Facebook. So, for example, this would be one bucket, this would be one bucket, this would be a bucket, and then these users would be another bucket. To do this, we just need to use the cut function on the variable year joined. I just need to tell cut on what years I should split my data. So I'm going to split at 2004, 2009, 2011, 2012, and 2014. Running this code, I can see that I got a new variable inside of my data frame.

Plotting it All Together

Notes:

We've done two things up to this point. - We created a variable called year_joined, based on the tenure variable, - and we converted year_joined, to the variable year_joined_bucket. A categorical variable. That bin their users into different groups.

On table(ing) this new variable to see the distribution in each group. we see that we have our four bins of users, depending on when they joined Facebook, and it looks like two people have a value of NA.

Let's use this new year joined bucket variable to create a line graph Like we did for gender at the start of the lesson. As a reminder, here's the code that generated this plot earlier.

```
ggplot(aes(x = age, y = friend_count),
       data = subset(pf, !is.na(gender))) +
  geom_line(aes(color = gender), stat = 'summary', fun.y = median)
```

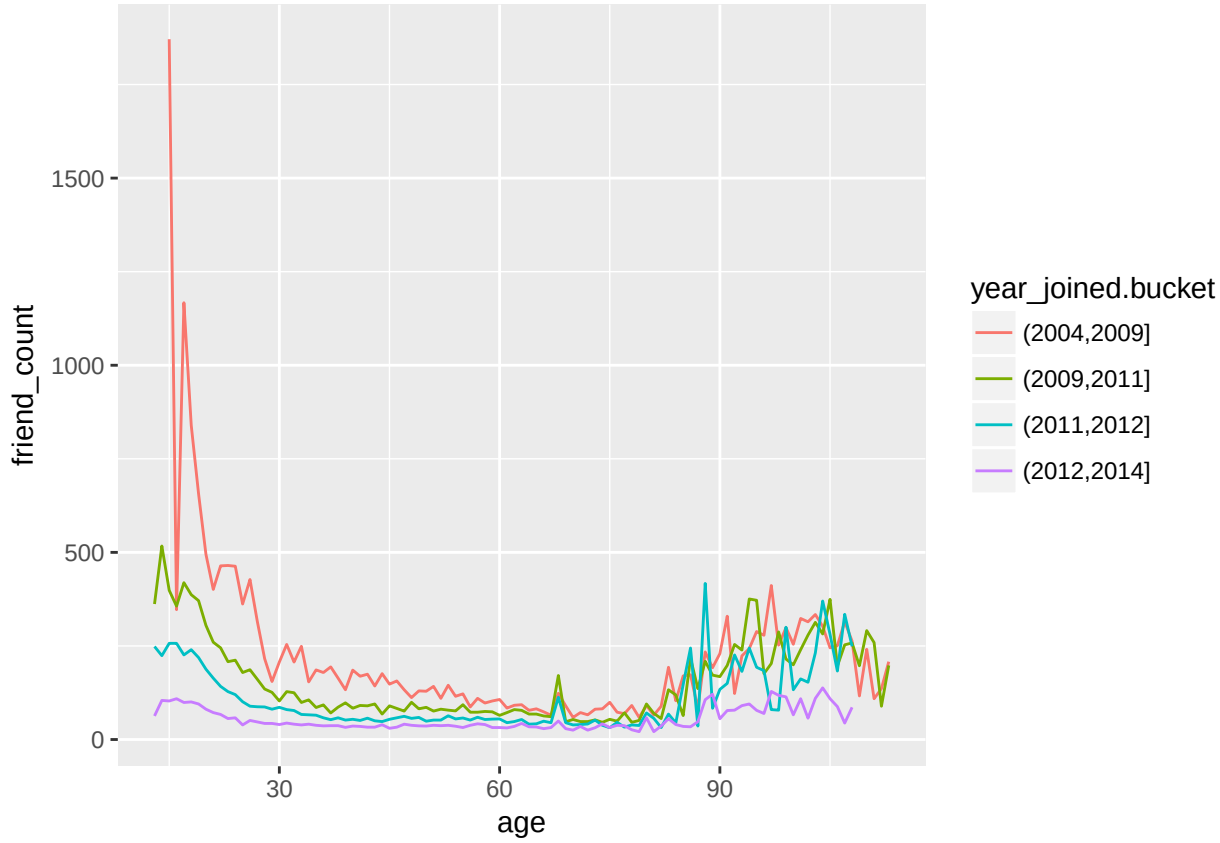
Also, notice how we compute the median friend count for each age using the fun.wide parameter, and the summary for the stat parameter. Using a similar code structure that we see here.

create a plot for friend count versus age, so that each year join bucket has its own line on the graph. In other words, each bucket will be a color line that tracks the median friend count across the age of users, just as it did in this plot for genders.

Create a line graph of friend_count vs. age so that each year_joined.bucket is a line tracking the median user friend_count across age. This means you should have four different lines on your plot.

You should subset the data to exclude the users whose year_joined.bucket is NA

```
ggplot(data = subset(pf, !is.na(year_joined.bucket)), aes(x=age, y = friend_count))+
  geom_line(aes(color = year_joined.bucket), stat = 'summary', fun.y = median)
```



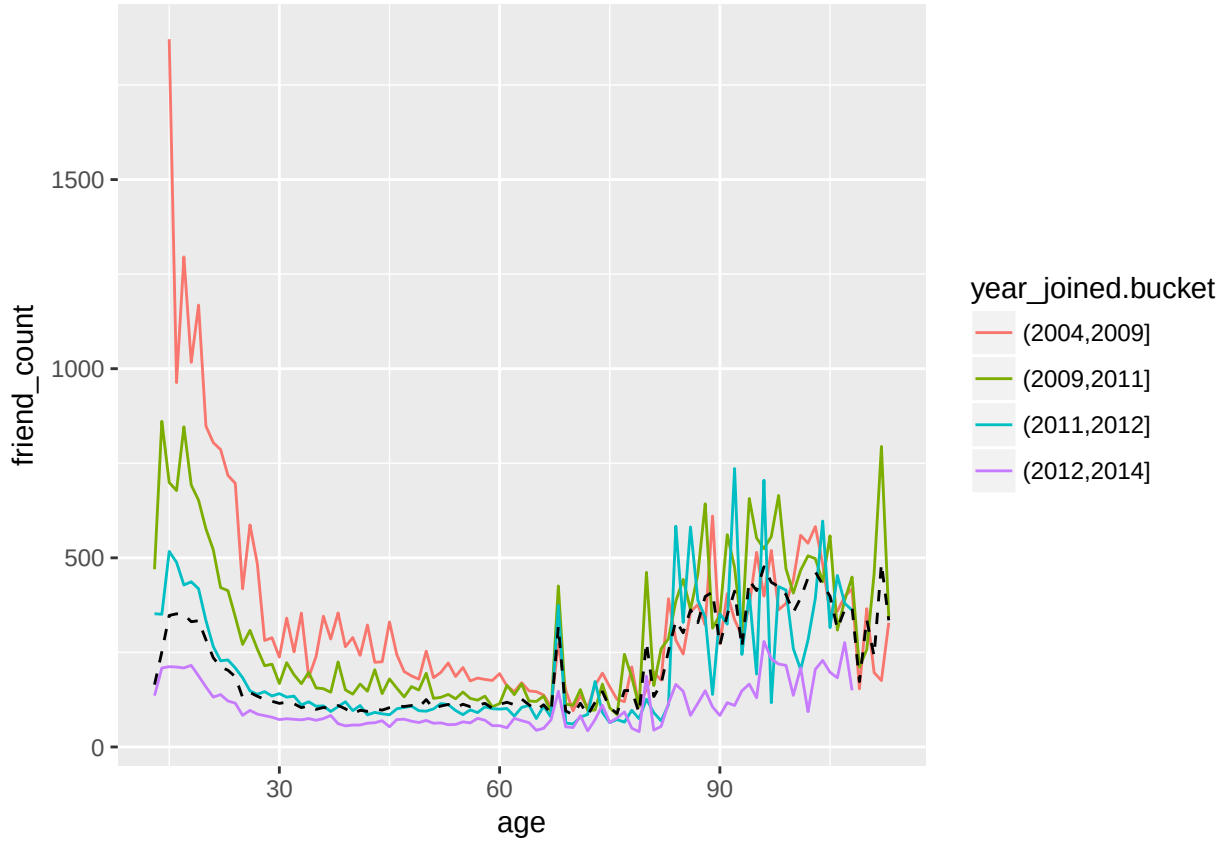
Looking at this plot, we can see that our suspicion is confirmed. Users with a longer tenure tend to have higher friend counts, with the exception of our older users, say, about 80 and up.

Plot the Grand Mean

Notes:

To put these cohort specific medians in perspective, we can change them to cohort specific means. And then plot the grand mean down here as well. The grand mean is the line we saw in lesson four when we plotted average friend count across the ages.

```
ggplot(data = subset(pf, !is.na(year_joined.bucket)), aes(x=age, y = friend_count))+
  geom_line(aes(color = year_joined.bucket), stat = 'summary', fun.y = mean) +
  geom_line(stat = 'summary', fun.y = mean, linetype = 2)
```



To plot the means and study the medians for each of the cohorts, we just need to change our fun.y parameter. This needs to be mean instead. Now, to get the grand mean, we just add a geom line and then set the parameters. We'll have a stat of summary. We'll set fun.y to equal the mean, and then we'll set the line type equal to two. Here, I'm using two so that way, I get a dash line so it stands out in comparison to these others. Plotting the grand mean is a good reminder that much of the data in the sample is about members of recent cohorts. This is the type of more high level observation that you want to make as you explore data.

Since the general pattern continues to hold after conditioning on each of the buckets of year joined, we might increase our confidence that this observation isn't just an artifact of the time users have had to accumulate friends

---

Friending Rate

Notes: Let's look at this relationship in another way. We could also look at tenure and friend count as a rate instead. For example, we could see how many friends does a user have for each day since they've started using the service.

Let's see if you can create a summary of this rate, that shows how many friends a user has for each day since the user started using Facebook. Subset the data so you only consider users with at least one day of tenure. Once you have that summary answer these two questions.

What's the median rate? And, what's the maximum rate?

```
with(subset(pf, tenure>=1), summary(friend_count/tenure))
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
```

```
##    0.0000    0.0775    0.2205    0.6096    0.5658 417.0000
```

Here we want a summary of the friend rate. So, we can use the with command and subset the data so we only consider users with tenure of at least one day. Now we just want a summary of friend count divided by tenure, which gives us the friends per day since the user's been active. When we run the code, we see that the median rate is about 0.22, and the maximum rate is 417. Now, this is definitely an outlier, considering our data, since the third quartile is only about 0.5.

Friendships Initiated

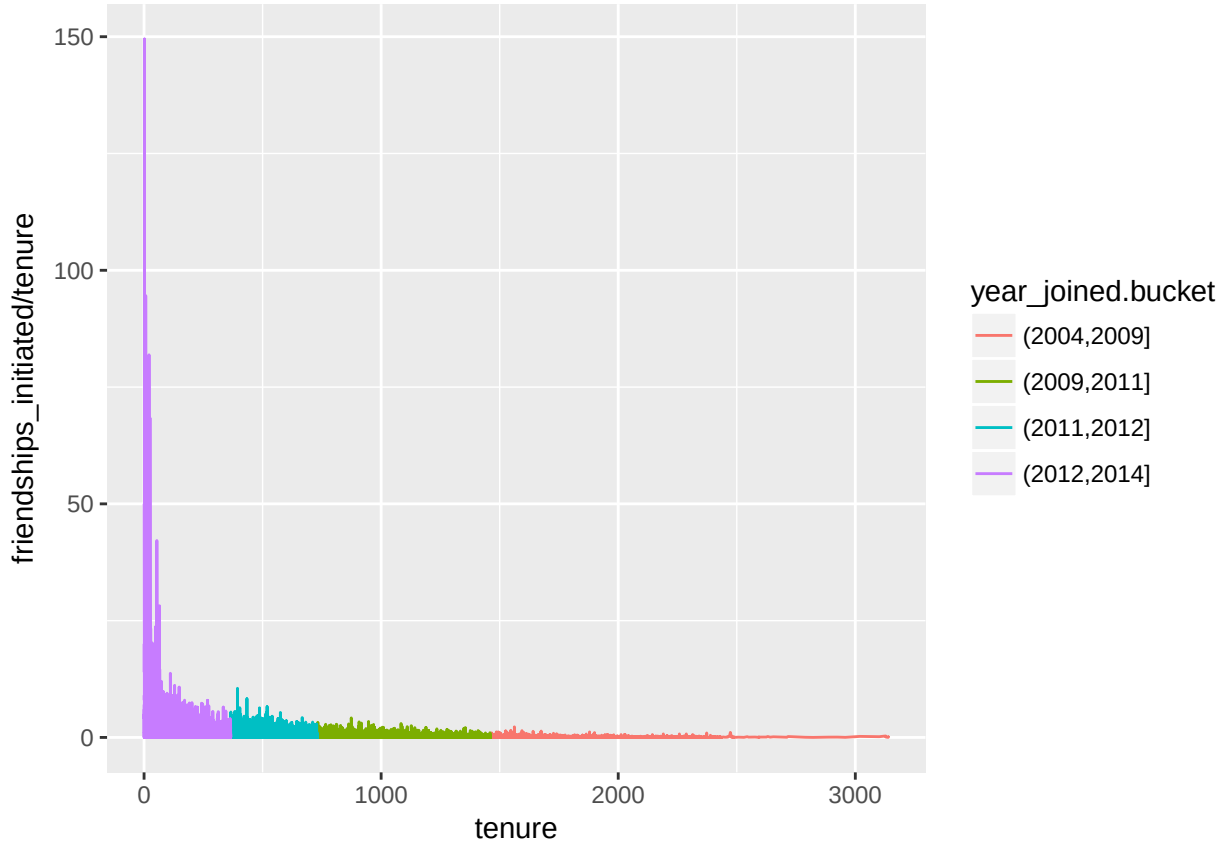
Notes:

What is the median friend rate? 0.2205

What is the maximum friend rate? 417

Let's explore this with a plot. I want you to create a line graph of friendships initiated per day, versus tenure. You need to make use of the variables age, tenure, friendships initiated, and year_joined.bucket. The color of each part of your one line should correspond to each bucket of year_joined. You'll also need to subset the data to only consider users with at least one day of tenure.

```
ggplot(data = subset(pf, tenure>=1), aes(x= tenure, y = friendships_initiated/tenure))+
  geom_line(aes(color = year_joined.bucket))
```



To create this plot, we're going to have much of the same code as before. We'll pass tenure to our x variable, and then we'll pass friendships initiated divided by tenure to our y variable, and we'll subset our data frame so that we only consider users who have a tenure of at least one day. We'll color our line by year_joined.bucket

and then we'll plot the mean of the y variable across tenure. Taking a closer look, **it appears that users with more tenure typically initiate less friendships**.

Bias-Variance Tradeoff Revisited

Notes:

From blog

When we discuss prediction models, prediction errors can be decomposed into two main subcomponents we care about: - error due to "bias" and - error due to "variance".

There is a tradeoff between a model's ability to minimize bias and variance. Understanding these two types of error can help us diagnose model results and avoid the mistake of over- or under-fitting.

Define bias and variance in three ways: conceptually and graphically.

1. Conceptual Definition

- **Error due to Bias:** The error due to bias is taken as the difference between the expected (or average) prediction of our model and the correct value which we are trying to predict. Of course you only have one model so talking about expected or average prediction values might seem a little strange. However, imagine you could repeat the whole model building process more than once: each time you gather new data and run a new analysis creating a new model. Due to randomness in the underlying data sets, the resulting models will have a range of predictions. **Bias measures how far off in general these models' predictions are from the correct value**.
- **_Error due to Variance:** The error due to variance is taken as the variability of a model prediction for a given data point. Again, imagine you can repeat the entire model building process multiple times. The **variance is how much the predictions for a given point vary between different realizations of the model**.

2. Graphical Definition We can create a graphical visualization of bias and variance using a bulls-eye diagram. Imagine that the center of the target is a model that perfectly predicts the correct values. **As we move away from the bulls-eye, our predictions get worse and worse**.

Imagine we can repeat our entire model building process to get a number of separate hits on the target.

Each hit represents an individual realization of our model, given the chance variability in the training data we gather. Sometimes we will get a good distribution of training data so we predict very well and we are close to the bulls-eye, while sometimes our training data might be full of outliers or non-standard values resulting in poorer predictions. These different realizations result in a scatter of hits on the target.

We can plot four different cases representing combinations of both high and low bias and variance.

observe : **Bias** : Shows Distance of points from the correct value. **variance** : Shows the spread of different points

The Bias Variance Tradeoff is an important concept in machine learning. This concept helps you evaluate which model will work the best. The bais varance problem arises when you start to use non linear models that don't have to follow straight lines.

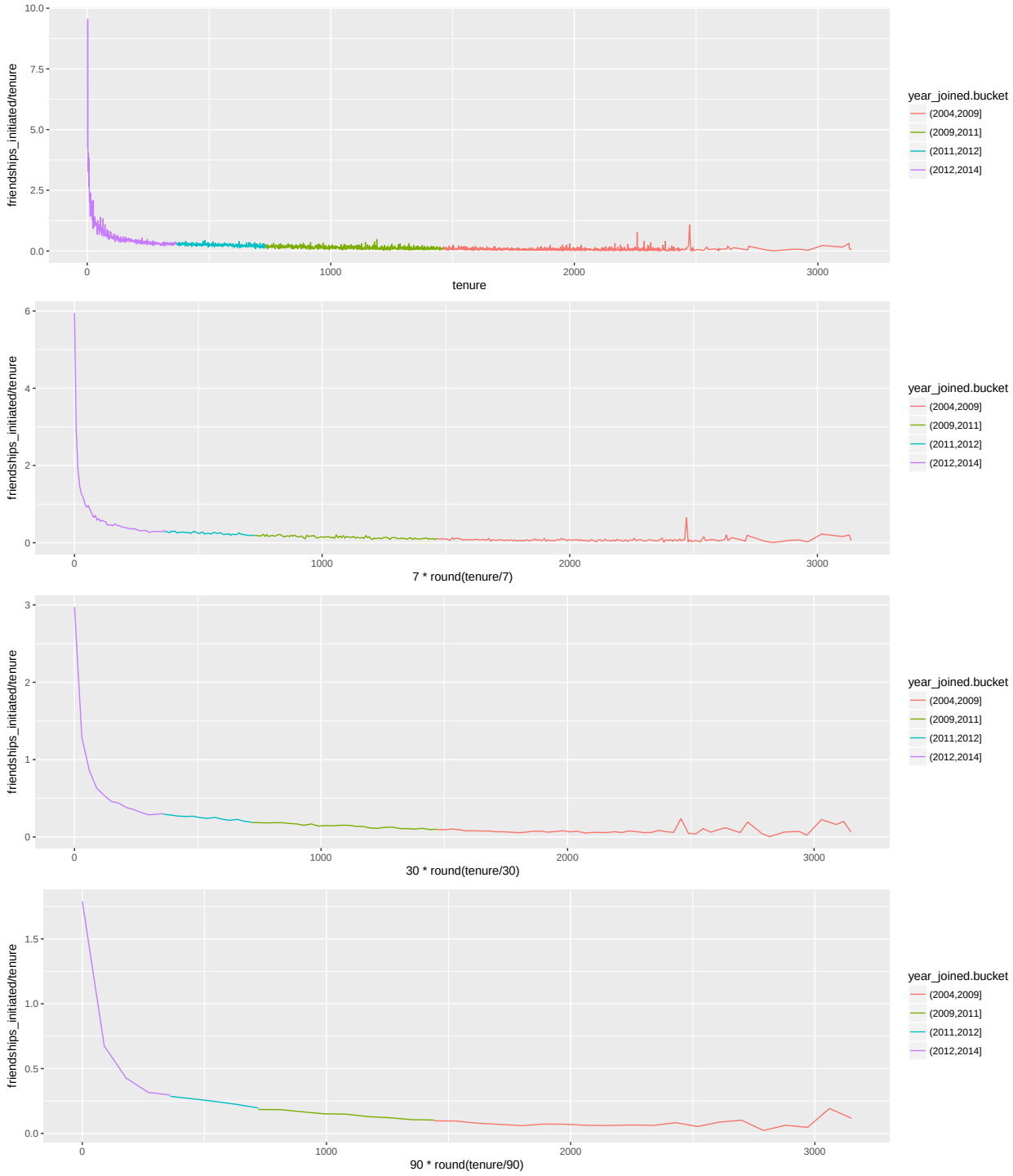
There's a lot of noise in our graph since we are plotting the mean of y for every possible tenure x value. Recall from lesson four that we can adjust this noise by bending our x-axis differently. Let me show you one

of those changes in the bend width. Here's our code from before (#A), and instead of using tenure here, I'm going to replace this with a different version or formula so that way I can bend some of the tenures together.

Now let's see the difference when I plot this graph(# B). Notice how I have slightly less noise in my plot. We still see some of the same peaks from before, especially here, but it's much smoother in general.

Here's another one using the number 30 (#C) instead of the number seven and here's a graph with very high bias, but much less variance (# D) using the number 90.

```
# A
p1 <- ggplot(aes(x = tenure, y = friendships_initiated / tenure),
       data = subset(pf, tenure >= 1)) +
  geom_line(aes(color = year_joined.bucket),
            stat = 'summary',
            fun.y = mean)
# B

p2 <- ggplot(aes(x = 7 * round(tenure / 7), y = friendships_initiated / tenure),
       data = subset(pf, tenure > 0)) +
  geom_line(aes(color = year_joined.bucket),
            stat = "summary",
            fun.y = mean)
# C
p3 <- ggplot(aes(x = 30 * round(tenure / 30), y = friendships_initiated / tenure),
       data = subset(pf, tenure > 0)) +
  geom_line(aes(color = year_joined.bucket),
            stat = "summary",
            fun.y = mean)
# D
p4 <- ggplot(aes(x = 90 * round(tenure / 90), y = friendships_initiated / tenure),
       data = subset(pf, tenure > 0)) +
  geom_line(aes(color = year_joined.bucket),
            stat = "summary",
            fun.y = mean)

grid.arrange(p1,p2,p3,p4,ncol=1)
```



Here I plotted all the graphs so we could compare it to the original one. Notice that **as the bin size increases we see less noise on the plot**. Our estimates are adjusted since we have more data points for our new values of tenure.

In lesson four we introduce smoothers as one tool for an analyst to use in these types of situations. So instead of using gm line here, I'd like you to use gm smooth, to add a smoother, to this plot.

Instead of geom_line(), use geom_smooth() to add a smoother to the plot. You can use the defaults for geom_smooth() but do color the line by year_joined.bucket.

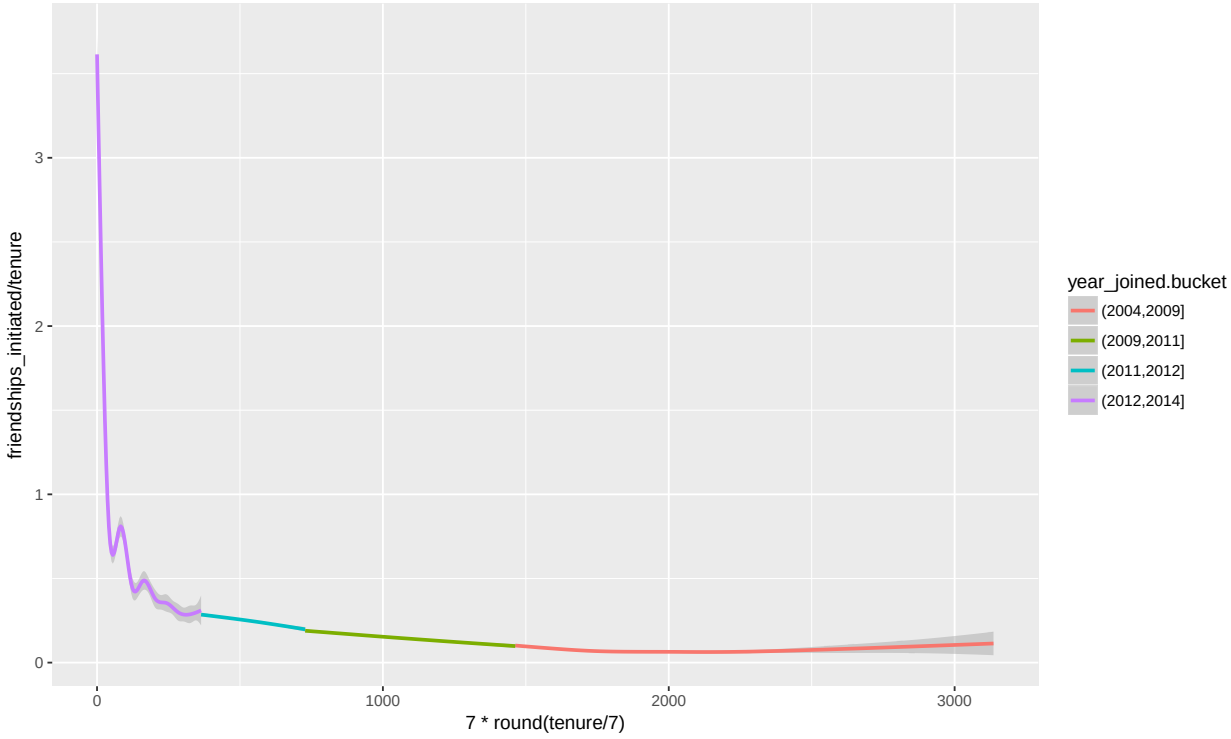
Ans : To add a smoother to this plot, we need ___to change geom line to geom smooth. We also need to **get rid of this fun.y parameter and the stat parameter**.

here using the **defaults for geom smooth. So R will automatically choose the appropriate statistical methods based on our data**.

All of that additional information can be found in the output down here. So here again, in this smooth version of the graph, we still see that the friendships initiated declines as tenure increases.

```
ggplot(aes(x = 7 * round(tenure / 7), y = friendships_initiated / tenure),
       data = subset(pf, tenure > 0)) +
  geom_smooth(aes(color = year_joined.bucket))
```

```
## `geom_smooth()` using method = 'gam'
```



Introducing the Yogurt Data Set

Notes: Throughout all of our analyses of the pseudo-Facebook user data set, we've come to learn a lot about our users. From their birthdays, to their friend counts, to their friendships initiated, we've really come to understand their behaviors and how they use the Facebook platform. But, now I think it's time for something completely different.

In the next couple of segments, we'll look at another data set, and then we'll return to this Facebook data set to draw some comparisons.

Because of online purchases, credit cards, and loyalty cards, lots of retail purchase data is associated with individuals or households, such that there is a history of purchase data over time. Analysts in industry often mine this panel scanner data, and economists and other behavioral scientists use it to test and develop theories about consumer behavior.

We are going to work with a **data set describing household purchases, of five flavors of Dannon yogurt in the eight-ounce size**. Their price is recorded with each purchase occasion. This yogurt data set has a quite different structure than our pseudo-Facebook data set. The **synthetic Facebook data has one row per individual with that row giving their characteristics and counts of behaviors over a single period of time. On the other hand**, the **yogurt data has many rows per household, one for each purchase occasion**.

This kind of microdata is often useful for answering different types of questions than we've looked at so far.

Histograms Revisited

Notes:

```
yo <- read.csv('yogurt.csv')
str(yo)
```

```
## 'data.frame':    2380 obs. of  9 variables:
##  $ obs        : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ id         : int  2100081 2100081 2100081 2100081 2100081 2100081 2100081 2100081 2100081 2100081
##  $ time       : int  9678 9697 9825 9999 10015 10029 10036 10042 10083 10091 ...
##  $ strawberry : int  0 0 0 0 1 1 0 0 0 0 ...
##  $ blueberry  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pina.colada: int  0 0 0 0 1 2 0 0 0 0 ...
##  $ plain      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ mixed.berry: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ price      : num  59 59 65 65 49 ...
```

convert one of the variables to a factor, and that's the ID variable.

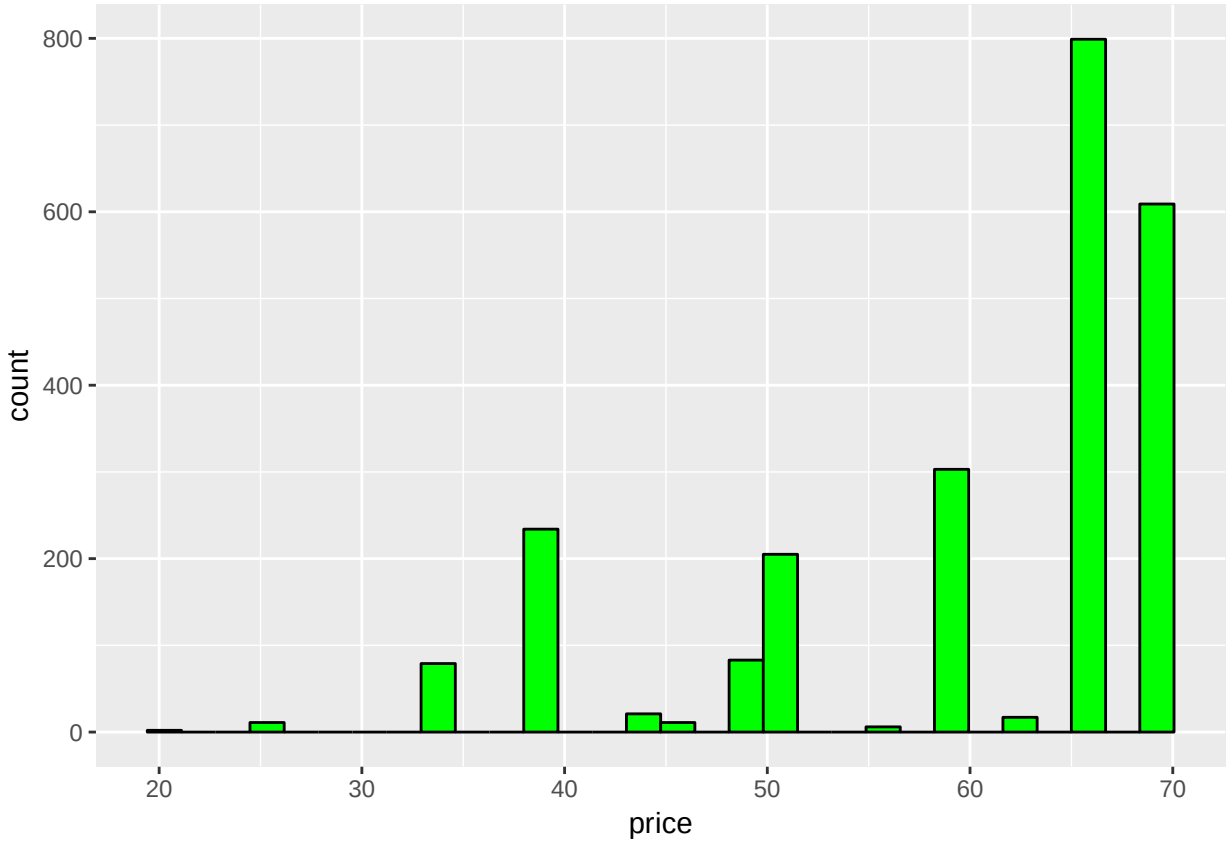
```
yo$id <- factor(yo$id)
str(yo)
```

```
## 'data.frame':    2380 obs. of  9 variables:
##  $ obs        : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ id         : Factor w/ 332 levels "2100081","2100370",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ time       : int  9678 9697 9825 9999 10015 10029 10036 10042 10083 10091 ...
##  $ strawberry : int  0 0 0 0 1 1 0 0 0 0 ...
##  $ blueberry  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pina.colada: int  0 0 0 0 1 2 0 0 0 0 ...
##  $ plain      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ mixed.berry: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ price      : num  59 59 65 65 49 ...
```

create a histogram of the Yogurt prices.

```
ggplot(data = yo, aes(x=price)) +
  geom_histogram(color = 'black', fill = 'green')
```

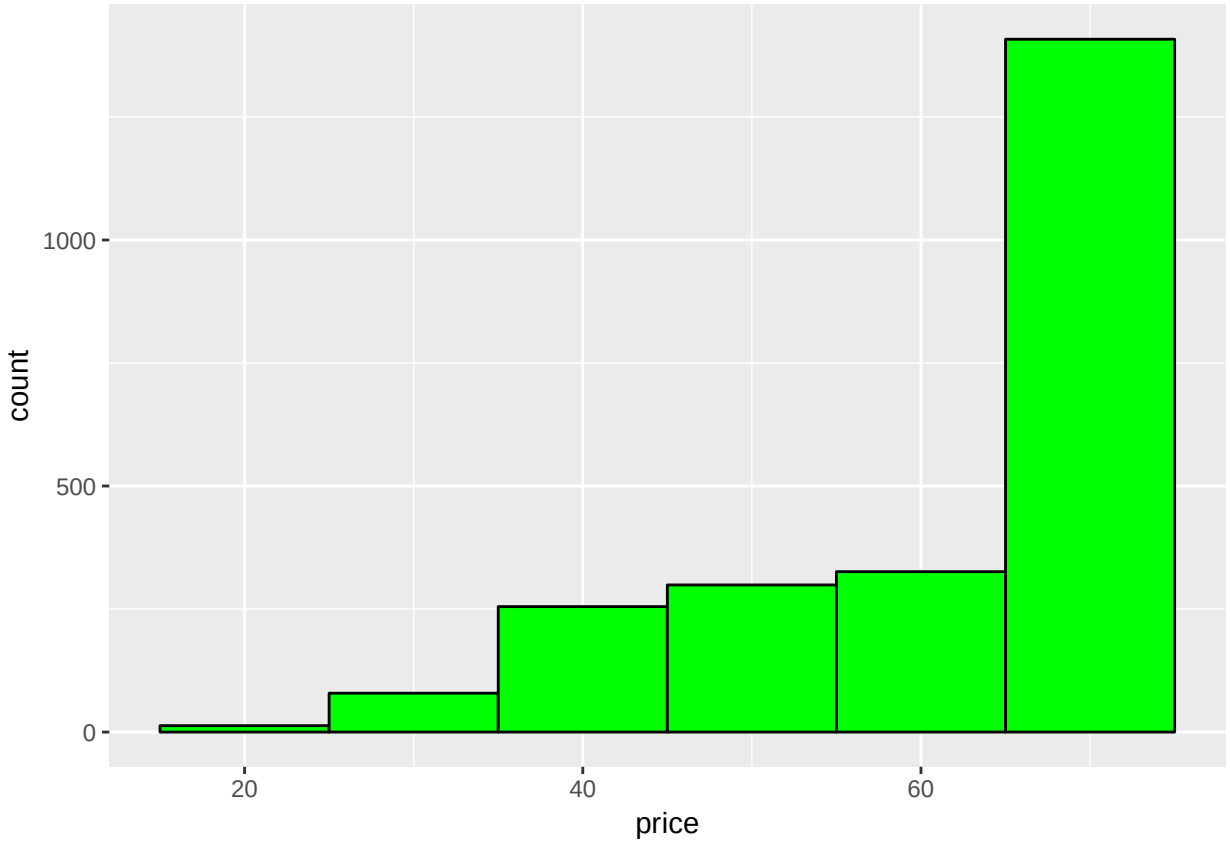
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



There appear to be prices at which there are many observations, but then no observations in adjacent prices. This makes sense if prices are set in a way that applies to many of the consumers. There are some purchases that involve much lower prices, and if we are interested in price sensitivity, we definitely want to consider what sort of variations is in these prices.

Now, I also want you to note that if we chose a different bin width we might obscure this discreetness. Say if I chose a bandwidth equal to ten.

```
ggplot(data = yo, aes(x=price)) +
  geom_histogram(binwidth = 10, color = 'black', fill = 'green')
```



In this histogram, we would miss the observation for some of the empty spaces for the adjacent prices. So, it's no surprise that for this very discreet data this histogram is a very biased model.

Number of Purchases

Notes:

There are other ways we could have noticed this important feature of the data set. But, there are also other ways, that by jumping into a different analysis, we might have missed this.

For example, if we just look at a five number summary of the data, we might not notice this so easily. One clue to the discreteness is that the 75th percentile for price is the same as the maximum.

```
summary(yo)
```

```
##       obs              id            time         strawberry
##  Min.   :   1.0   2132290:  74   Min.   : 9662   Min.   : 0.0000
##  1st Qu.: 696.5   2130583:  59   1st Qu.: 9843   1st Qu.: 0.0000
##  Median :1369.5   2124073:  50   Median :10045   Median : 0.0000
##  Mean   :1367.8   2149500:  50   Mean   :10050   Mean   : 0.6492
##  3rd Qu.:2044.2   2101790:  47   3rd Qu.:10255   3rd Qu.: 1.0000
##  Max.   :2743.0   2129528:  39   Max.   :10459   Max.   :11.0000
##                   (Other):2061
##    blueberry        pina.colada          plain         mixed.berry
##  Min.   : 0.0000   Min.   : 0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.:0.0000
```

```
##  Median : 0.0000   Median : 0.0000   Median :0.0000   Median :0.0000
##  Mean   : 0.3571   Mean   : 0.3584   Mean   :0.2176   Mean   :0.3887
##  3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000
##  Max.   :12.0000   Max.   :10.0000   Max.   :6.0000   Max.   :8.0000
##
##      price
##  Min.   :20.00
##  1st Qu.:50.00
##  Median :65.04
##  Mean   :59.25
##  3rd Qu.:68.96
##  Max.   :68.96
##
```

We could also see this discreteness by looking at how many distinct prices there are in the data set.

```
unique(yo$price)
```

```
##  [1] 58.96 65.04 48.96 68.96 39.04 24.96 50.00 45.04 33.04 44.00 33.36
## [12] 55.04 62.00 20.00 49.60 49.52 33.28 63.04 33.20 33.52
```

So here it looks like there's about 20 different prices.

Tabling the variable we get an idea of the distribution like we saw in the histogram.

```
table(yo$price)
```

```
##
##    20 24.96 33.04  33.2 33.28 33.36 33.52 39.04    44 45.04 48.96 49.52
##     2    11    54     1     1    22     1   234    21    11    81     1
##  49.6    50 55.04 58.96    62 63.04 65.04 68.96
##     1   205     6   303    15     2   799   609
```

let's figure out on a given purchase occasion how many eight ounce yogurts does a household purchase. To answer this we need to combine accounts of the different yogurt flavors into one variable.

To figure this out for all the households, we need to make use of a new function. The function is called the **transform function**.

create a new variable called all.purchases, which gives the total counts of yogurt for each observation or household purchase. Keep in mind that you need to save this variable to the data frame

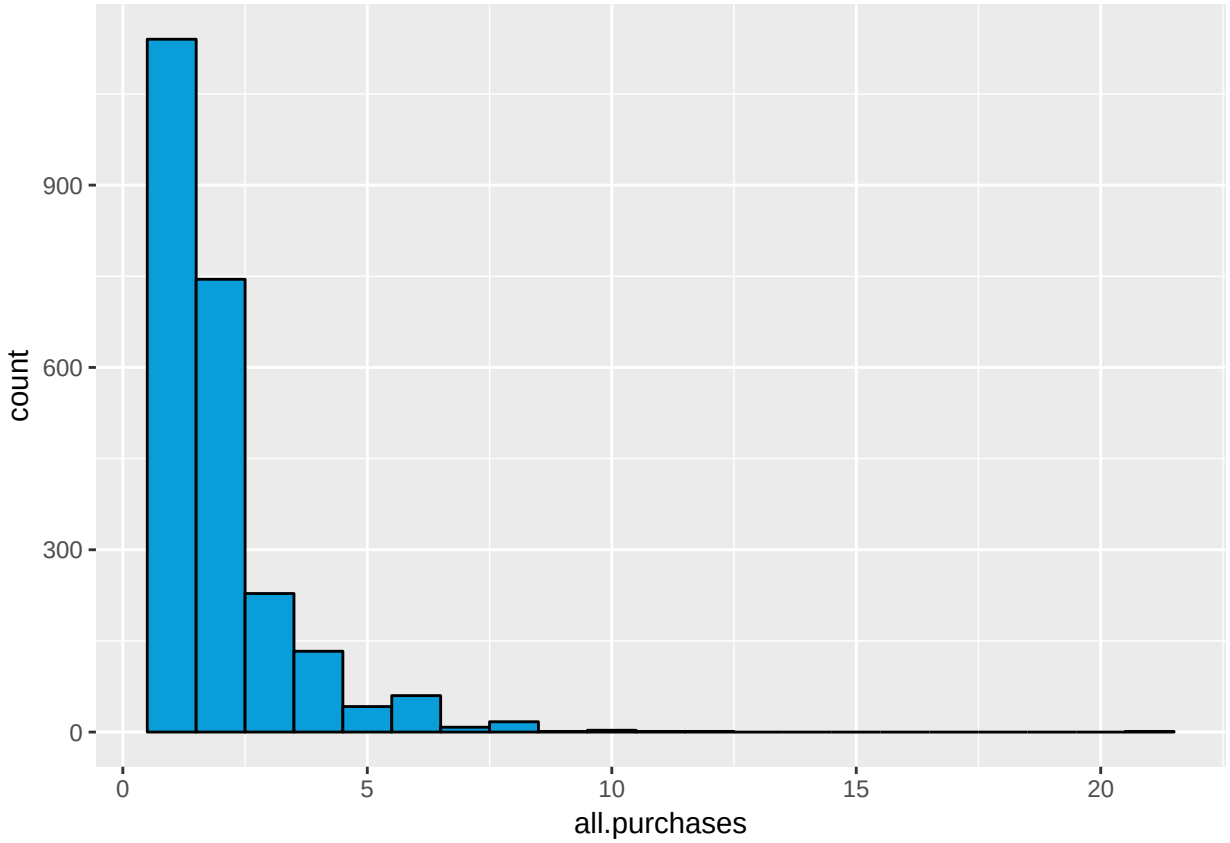
```
yo <- transform(yo , all.purchases = strawberry + blueberry + pina.colada + plain + mixed.berry )
summary(yo$all.purchases)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   2.000   1.971   2.000  21.000
```

alternately, we can alos do this :

```
yo$all.purchases <-  yo$strawberry + yo$blueberry + yo$pina.colada + yo$plain + yo$mixed.berry
```

```
ggplot(data = yo, aes(x=all.purchases))+
  geom_histogram(color = 'black', fill = '#099DD9', binwidth = 1)
```

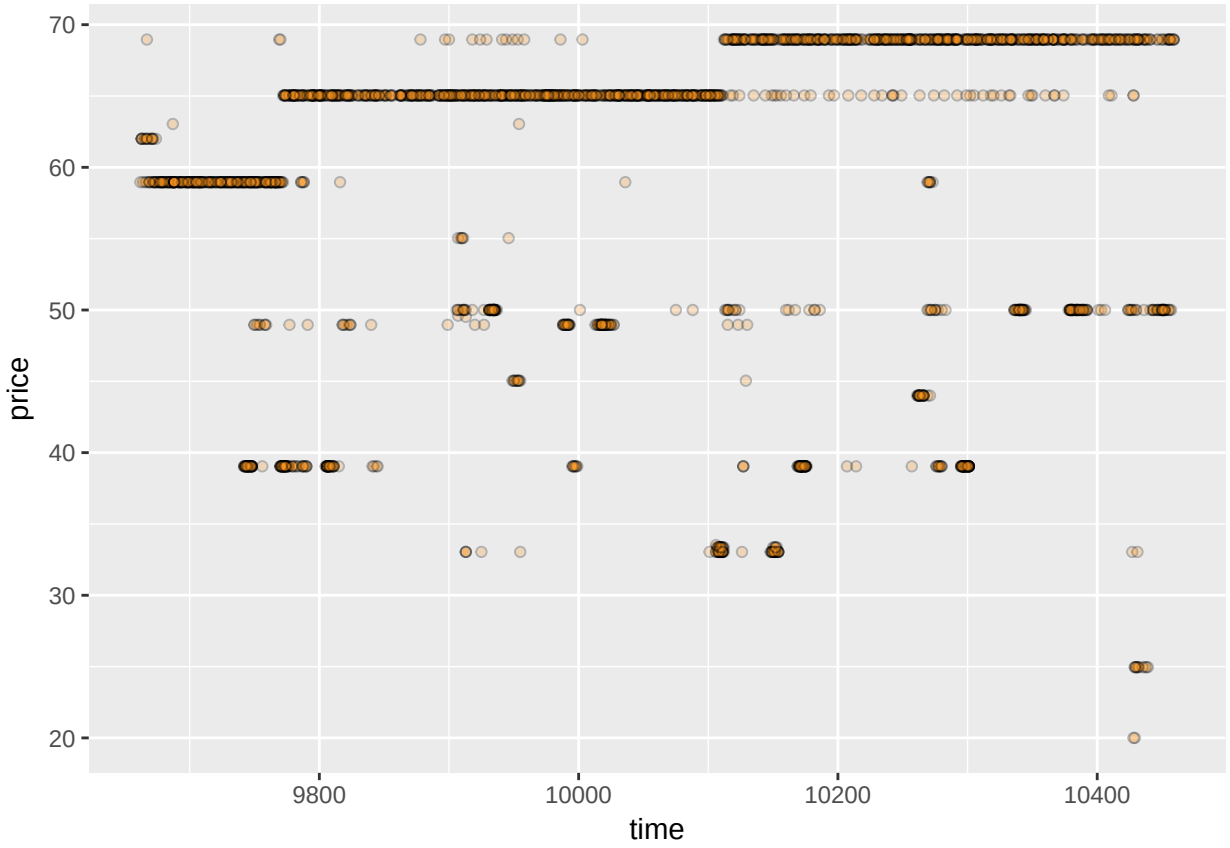


This histogram reveals that most households buy one or two yogurts at a time.

Prices over Time

Notes: Let's investigate the price over time in more detail. This is where our Facebook data was deficient. In this data set, we can examine changes in prices because we have data on the same households over time. So, now I want you to use your knowledge of scatterplots and over-plotting to create an appropriate visualization. Your visualization should be a scatter plot of price versus time.

```
ggplot(data = yo, aes(x=time, y = price))+
  geom_point(alpha = 1/4, shape = 21, fill = '#F79420')
```



Looking at the plot, we can **see that the mode or the most common prices, seem to be increasing over time**. We also see some lower price points scattered about the graph. These may be due to sales or, perhaps, buyers using coupons that bring down the price of yogurt.

Sampling Observations

Notes: our work up to this point with this yogurt data set has been review and it should be, but lets hear from Dean about how we might proceed differently with this type of a data set.

When familiarizing yourself with a new data set that contains multiple observations of the same units, it's often useful to work with a sample of those units so that it's easy to display the raw data for that sample.

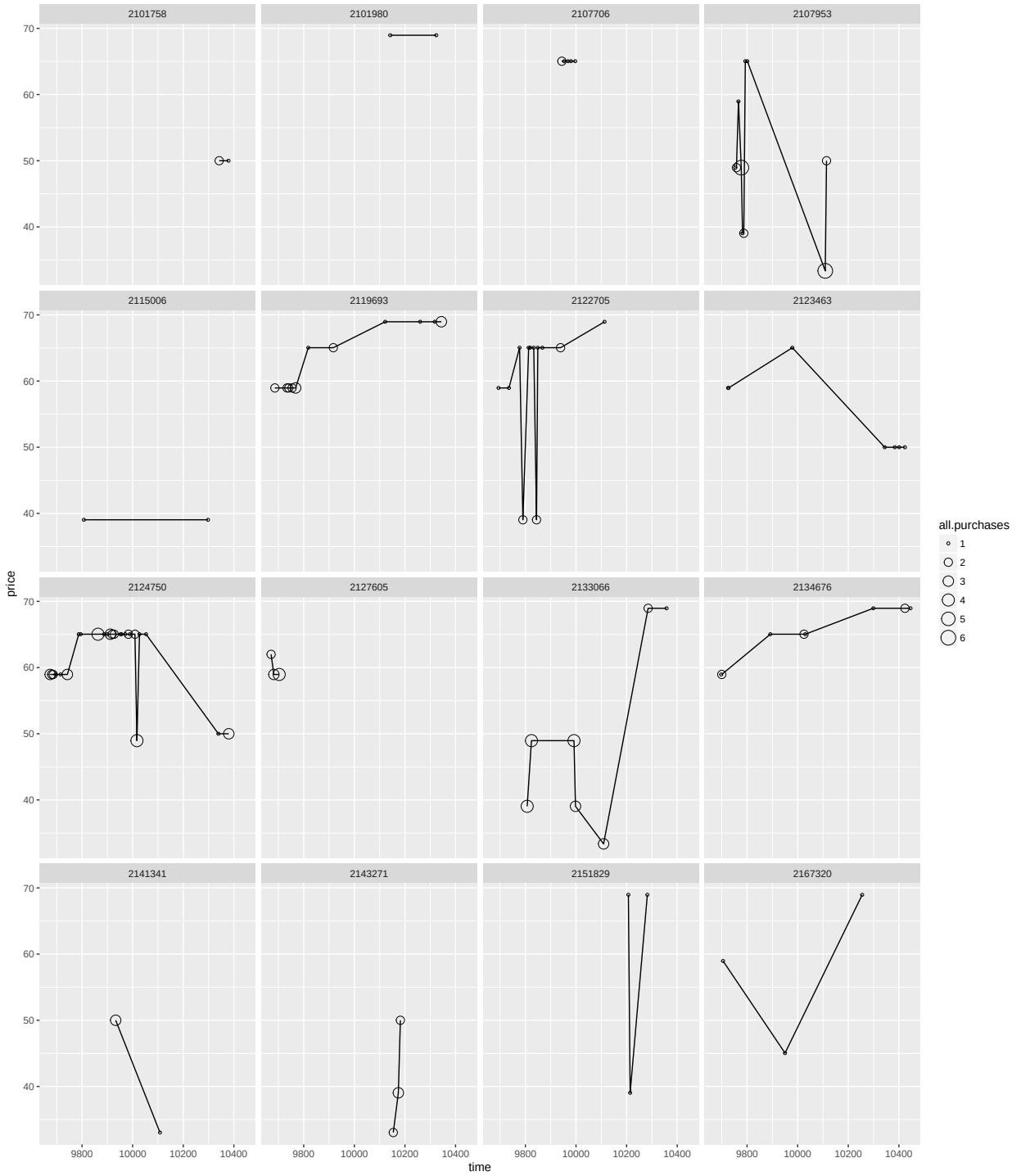
In the case of the yogurt data set, we might want to look at a small sample of households in more detail so that we know what kind of within and between household variation we are working with.

This analysis of a sub-sample might come before trying to use within household variation as part of a model. For example, this data set was originally used to model consumer preferences for variety. But, before doing that, we'd want to look at how often we observe households buying yogurt, how often they buy multiple items, and what prices they're buying yogurt at. One way to do this is to look at some sub-sample in more detail.

Let's pick 16 households at random and take a closer look.

Looking at Samples of Households

```
set.seed(4230)
sample.ids <- sample(levels(yo$id),16)

ggplot(data = subset(yo, id %in% sample.ids), aes(x=time, y=price))+
  facet_wrap( ~ id) +
  geom_line()+
  geom_point(aes(size=all.purchases), pch =1)
```



First, we should use the set seed function to make this reproducible. Running this one line of code will also allow you to see the same households as in udacity tutorials' explorations, if you use the same seed number.

Now, let's sample 16 of the households from our yogurt data set, from yogurt ID. Notice that I'm sampling from the levels because those are all of the different households that I have. I'll run the code for the sample ID's. And then, I'll print out the sample ID's. There's the 16 of them.

Now we can plot each purchase occasion for each of the households that we sampled. We have the time of the purchase, the price per item of the yogurt, and the number of items. Here, I'm using the size parameter to

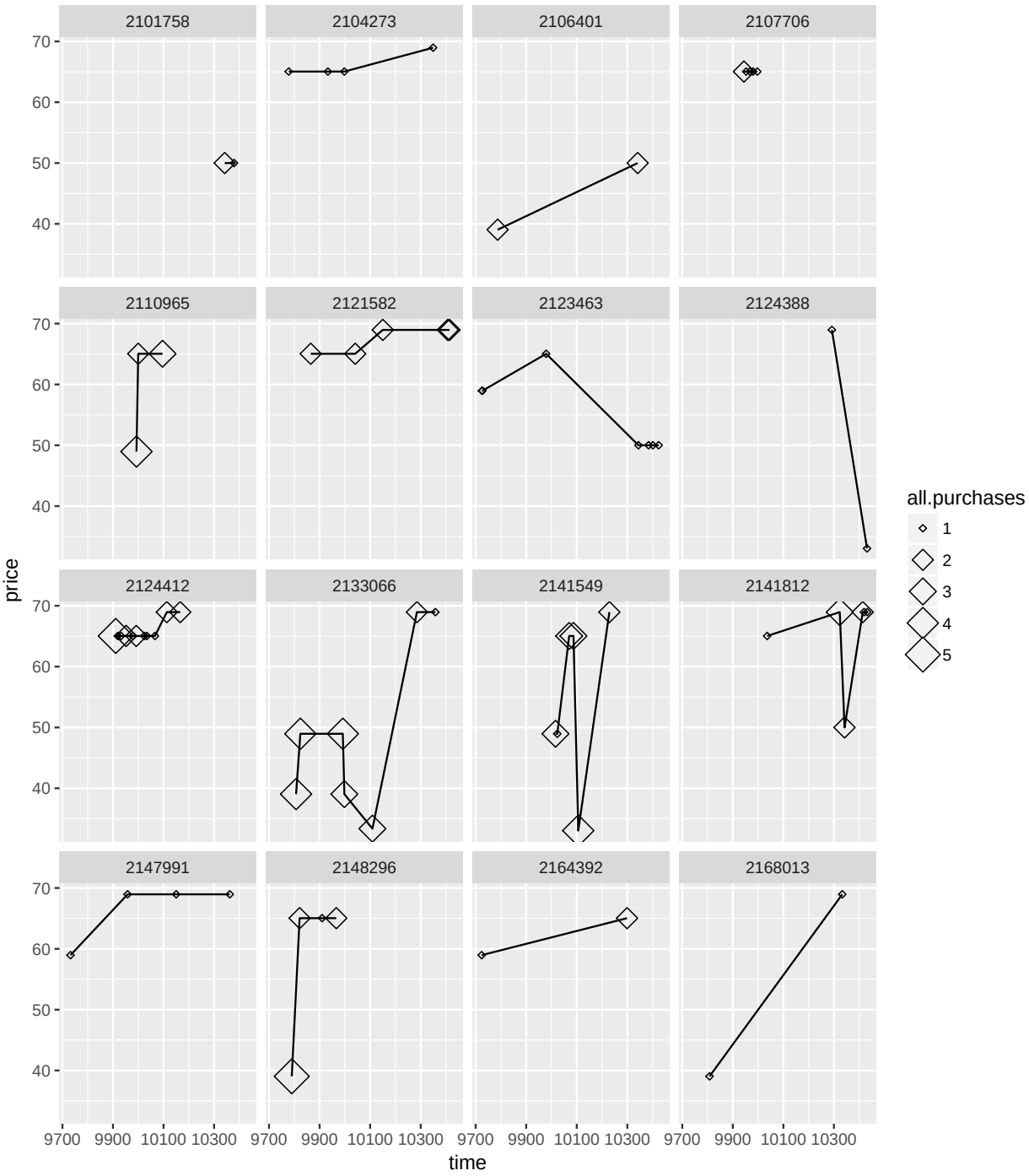
add more detail to my plot. I'm passing at the all purchases variable, so that way I can consider the number of items in terms of size of the point on the plot.

From these plots, we can see the variation and how often each household buys yogurt. Here, a lot and here, not to much. And it seems that some household purchases more quantities than others with these larger circles indicating not here.

For most of the households, the price of yogurt holds steady, or tends to increase over time. Now, there are, of course, some exceptions, like in this household and in this household, and even here, we might think that the household is using coupons to drive the price down. Now, we don't have the coupon data to associate with this buying data, but you could see how that information could be paired to this data to better understand the consumer behavior.

- **x %in% y** returns a logical (boolean) vector the same length as x that says whether each entry in x appears in y. That is, for each entry in x, it checks to see whether it is in y. This allows us to subset the data so we get all the purchases occasions for the households in the sample.
- Use the **pch or shape parameter to specify the symbol when plotting points**. Scroll down to 'Plotting Points' on QuickR's Graphical Parameters.

```
set.seed(2000)
sample_id2 <- sample(levels(yo$id),16)

ggplot(data = subset(yo, id %in% sample_id2), aes(x=time,y=price))+
  geom_line()+
  facet_wrap( ~ id)+
  geom_point(aes(size=all.purchases), pch = 5)
```



The Limits of Cross Sectional Data

Notes:

The general idea is that **if we have observations over time, we can facet by the primary unit, case, or individual in the data set. For our yogurt data it was the households we were faceting over**.

This faceted time series plot is something we can't generate with our pseudo Facebook data set. Since we don't have data on our sample of users over time. Let's get back to that plot to see that limitation.

The Facebook data isn't great for examining the process of friending over time. The data set is just a cross section, it's just one snapshot at a fixed point that tells us the characteristics of individuals.

Not the individuals over, say, a year. But if we had a dataset like the yogurt one, we would be able to track friendships initiated over time and compare that with tenure. This would give us better evidence to explain the difference or the drop in friendships initiated over time as tenure increases.

Many Variables

Notes:

Most recently, **when analyzing the relationship between two variables we look to incorporate more variables in the analysis to improve it**. For example, by seeing whether a particular relationship is consistent across values of those other variables.

In choosing a third or fourth variable to plot we relied on our domain knowledge.

But often, we might want visualizations or summaries to help us identify such auxiliary variables.

In some analyses, we may plan to make use of a large number of variables.

Perhaps, we are planning on predicting one variable with ten, 20, or hundreds of others. Or maybe we want to summarize a large set of variables into a smaller set of dimensions. Or perhaps, we're looking for interesting relationships among a large set of variables. In such cases, we can help speed up our exploratory data analysis by producing many plots or comparisons at once. This could be one way to let the data set as a whole speak in part by drawing our attention to variables we didn't have a preexisting interest in.

Scatterplot Matrix

Notes: we should let the data speak to determine variables of interest.

There's a tool that we can use to create a number of scatter plots automatically. It's called a **scatter plot matrix**. In a **scatter plot matrix. There's a grid of scatter plots between every pair of variables**.

As we've seen, scatter plots are great, but not necessarily suited for all types of variables. For example, categorical ones. So there are other types of visualizations that can be created instead of scatter plots. Like box plots or histograms when the variables are categorical.

Let's produce the scatter plot matrix for our pseudo Facebook data set. We're going to **use the GGally package** to do so.

First we want to set the seed so we get reproducible results. Now, you might be wondering why we set the seed in the first place. And it's because we're going to sample from our data set. Our data set contains all these variables and I actually don't want all the variables. I don't want user ID, year joined, or year joined.bucket. So what I can do is subset my data frame and then sample from that subset. If I check out the variables in my subset data frame these are the ones of interest.

```
You'll need to run the code install.packages('GGally') to install the package for creating this particul

If the plot takes a long time to render or if you want to see some of the scatterplot matrix, then only
```

```
pf_subset = pf[, c('age', 'dob_year', 'dob_month', 'gender', 'tenure')]
```

You can also select a subset using the subset() function and the "select" argument:

```
pf_subset <- subset(pf, select = -c(userid, year_joined, year_joined_bucket))
```

The - sign in the "select" value indicates all but the listed columns.

You may find in your matrix that variable labels are on the outer edges of the scatterplot matrix, rath

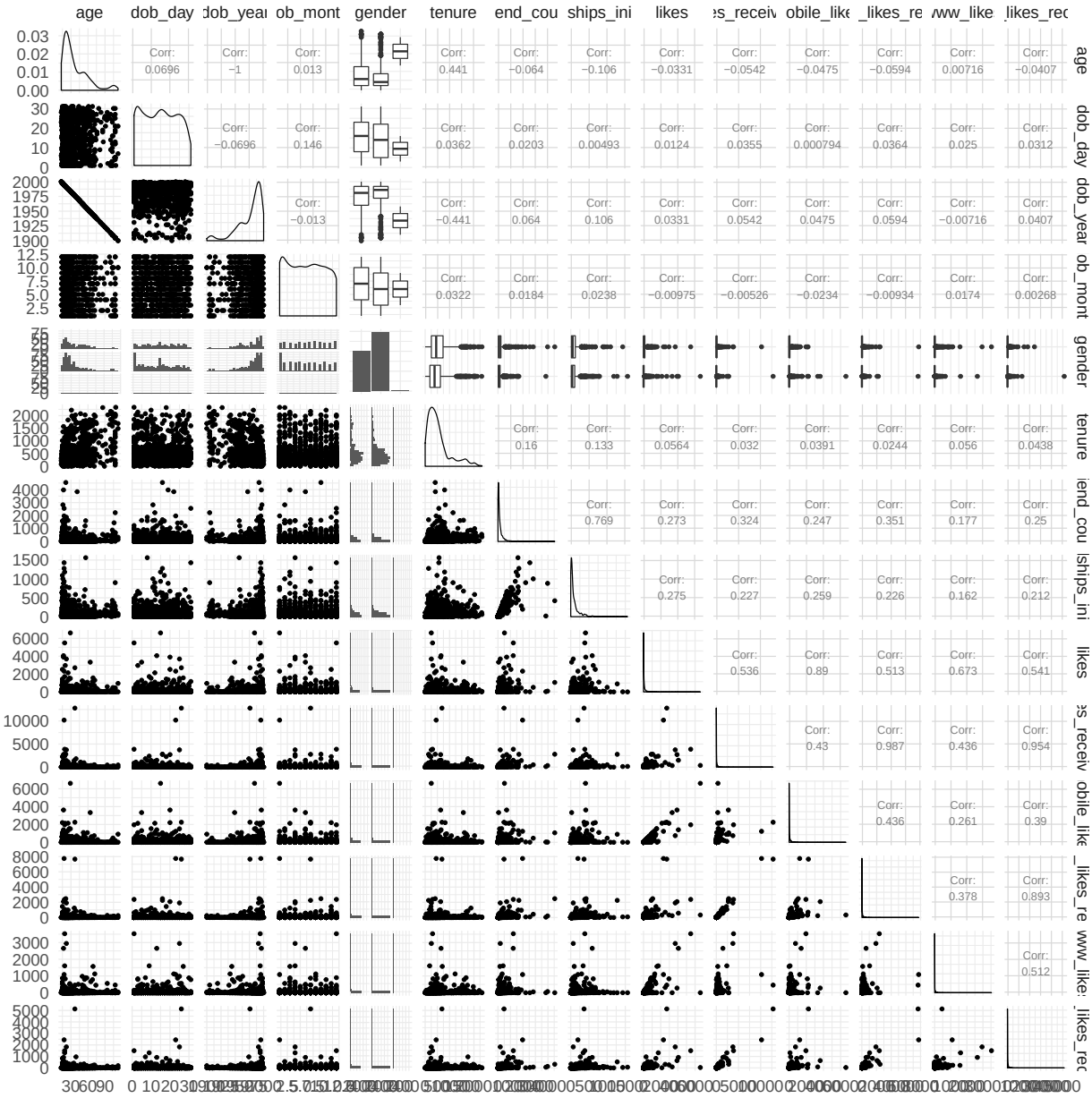
```
theme_set(theme_minimal(20))

# set the seed for reproducible results
set.seed(1836)
pf_subset <- pf[ , c(2:15)]
names(pf_subset)
```

```
##  [1] "age"                   "dob_day"
##  [3] "dob_year"              "dob_month"
##  [5] "gender"                "tenure"
##  [7] "friend_count"          "friendships_initiated"
##  [9] "likes"                 "likes_received"
## [11] "mobile_likes"          "mobile_likes_received"
## [13] "www_likes"             "www_likes_received"
```

```
ggpairs(pf_subset[sample.int(nrow(pf_subset), 1000 ), ])
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning: Removed 2 rows containing non-finite values (stat_boxplot).

## Warning: Removed 2 rows containing non-finite values (stat_boxplot).

## Warning: Removed 2 rows containing non-finite values (stat_boxplot).

## Warning: Removed 2 rows containing non-finite values (stat_boxplot).

## Warning: Removed 2 rows containing non-finite values (stat_boxplot).

## Warning: Removed 2 rows containing non-finite values (stat_boxplot).

## Warning: Removed 2 rows containing non-finite values (stat_boxplot).

## Warning: Removed 2 rows containing non-finite values (stat_boxplot).

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

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




```
#ggsave('scatter_matrx.png')
```

Scatterplots are below the diagonal, and categorical variables, like gender, create faceted histograms.

The GG pairs function uses a different plot type for different types of combinations of variables.

Hence, we have histograms here and we have scatter plots here. Many of these plots aren't quite as nice as they would be if we fine-tuned them for the particular variables. For example, for all the counts of likes, we might want to work on a logarithmic scale. But, ggpairs doesn't do this for us. At the very least, a scatter plot matrix can be a useful starting point in many analyses.

Even More Variables

Notes:

A matrix such as this one will be extremely helpful when we have even more variables than those in the pseudo-Facebook data set. Examples arise in many areas, but one that has attracted the attention of statisticians is **genomic data. In these data sets, they're often thousands of genetic measurements for each of a small number of samples**. In some cases, some of these samples have a disease, and so we'd like to identify genes that are associated with the disease.

Heat Maps

```
nci <- read.table("nci.tsv")

# Changing the column names to produce a nicer plot
colnames(nci) <- c(1:64)
```

The **data contains the expression of 6,830 genes, compared with a larger baseline reference sample. Now, this is a ton of data**. So let's go ahead and read in the data set, and then I'll change the color names of the data set to be the numbers from one to 64. Now, I'm just doing this so that way the plot that I create is going to be a little bit nicer with the labeling on the x axis.

The last plot that we'll make for this course is called a Heat Map. **For our data set we want to display each combination of gene and sample case, the difference in gene expression and the sample from the base line**.

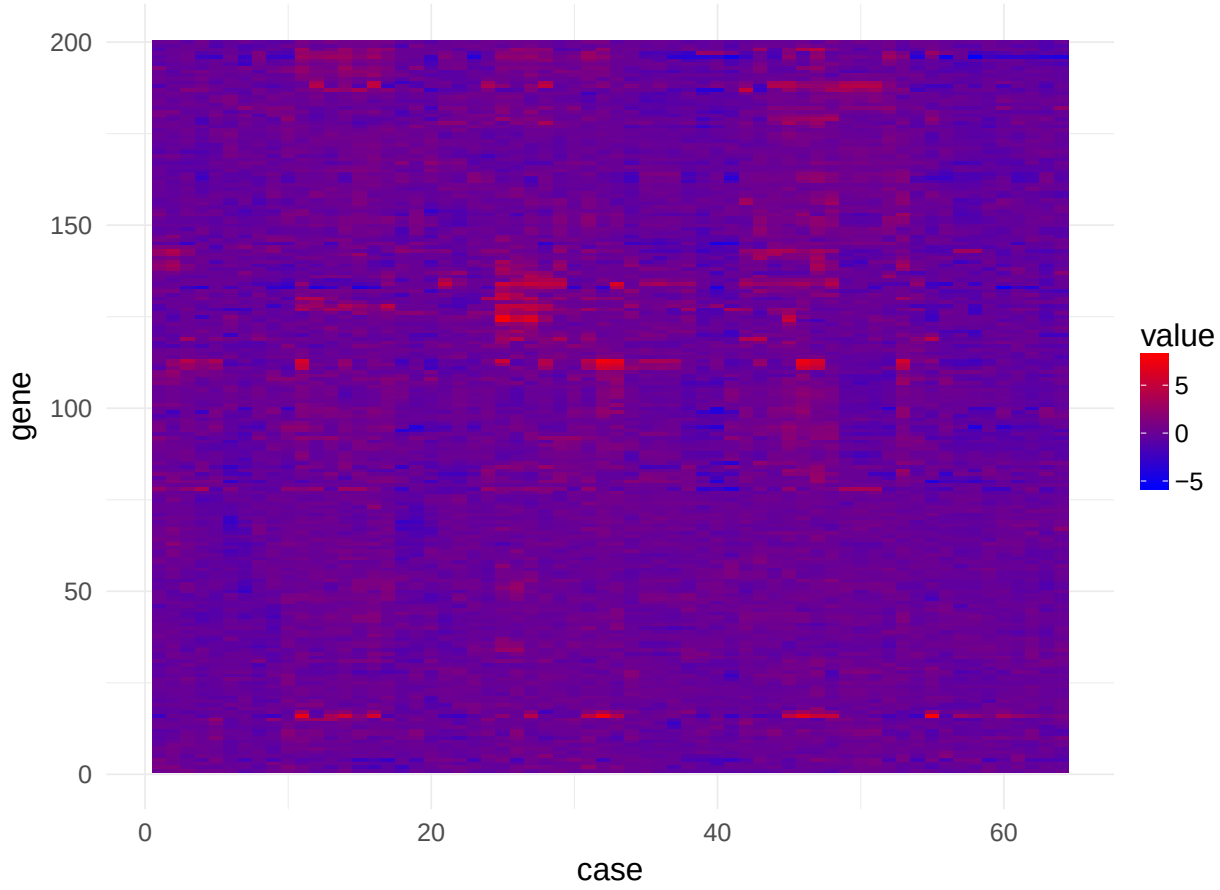
We want to display combinations where a gene is overexpressed in red. in combinations where it is under expressed in blue.

First, we'll run all of this in order to melt our data to a long format. And then we just run our ggplot code using the geom, geom tile. Now, this last line is going to give us a scale gradient. And we're going to use the colors from blue to red.

```
nci.long.samp <- melt(as.matrix(nci[1:200,]))
names(nci.long.samp) <- c("gene", "case", "value")
head(nci.long.samp)
```

```
##   gene case  value
## 1    1    1  0.300
## 2    2    1  1.180
## 3    3    1  0.550
## 4    4    1  1.140
## 5    5    1 -0.265
## 6    6    1 -0.070
```

```
ggplot(aes(y = gene, x = case, fill = value),
  data = nci.long.samp) +
  geom_tile() +
  scale_fill_gradientn(colours = colorRampPalette(c("blue", "red"))(100))
```



And, there's our Heat Map.

Even with such a dense display, we aren't looking at all the data. In particular, we're just showing the first 200 genes. That's 200 genes of over **Genomic data sets** of these kind, sometimes called **micro data** are only getting larger, and more complex.

What's most interesting, is that other data sets also look like this. For example, internet companies run lots of randomized experiments. Where in the simplest versions, users are randomly assigned to a treatment like a new version of a website or some sort of new feature or product or a control condition. Then the difference in outcome between the treatment and control can be computed for a number of metrics of interest.

In many situations, there might have been hundreds or thousands of experiments and hundreds of metrics. This data looks very similar to the genomic data in some ways. And this is why the useful maxim plot all the data might not always apply to a data set as it did to most of this course.

Analyzing Three of More Variables

Reflection:

We started with simple extensions to the scatter plot, and plots of conditional summaries that you worked with in lesson four, such as adding summaries for multiple groups. Then, we tried some techniques for examining a large number of variables at once, such as scatter-plot matrices and heat maps. >> We also learned how to reshape data, moving from broad data with one row per case, to aggregate data with one row per combination of variables, and we moved back and forth between long and wide formats for our data. ***