Recitation 3: R Notes and PS2 Empirical Practice Problems

Matthew Alampay Davis

October 5, 2021

As mentioned in the notes from my first recitation, if you have not used R Notebooks before, you will need to run the following command into your console or command panel (not the editor panel):

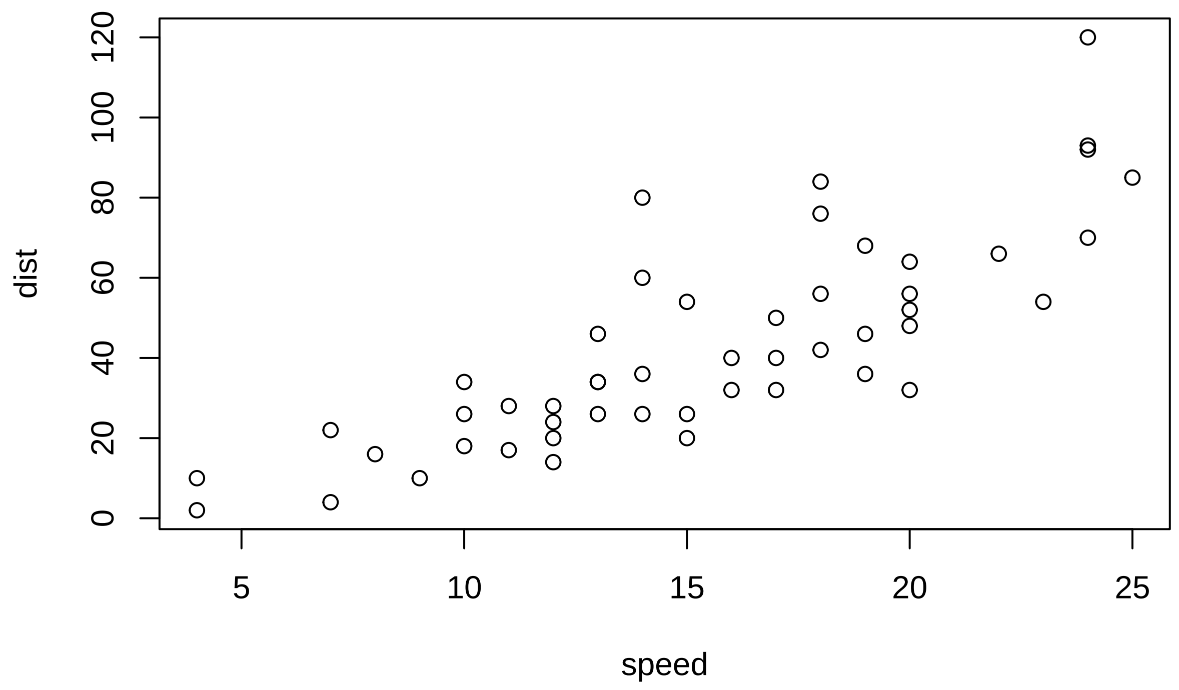
tinytex::install_tinytex()

You only ever need to do this once so if you've done it before, don't do it again. This just allows R to produce pdfs so you need to do this so you can actually produce your problem sets!

Part 1: Plotting data

Going back to our cars dataset, let's try plotting the data. Again, do this by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

```
plot(cars)
```



Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Cmd+Option+I*.

When you save the notebook, an HTML or pdf file containing the code and output will be saved alongside it in the same folder (click the *Preview* button or press *Cmd+Shift+K* to preview the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed. We'll tend to prefer to Knit.

Part 2: Downloading packages

We can access new or different functions by downloading "packages", which are basically third-party collections of functions that someone coded up to complement the functions that come in-built with R. You'll find that we'll rely on these third-party functions basically every week. To download a package, for example the the "ggplot2" package, run the following commmand:

install.packages('ggplot2')

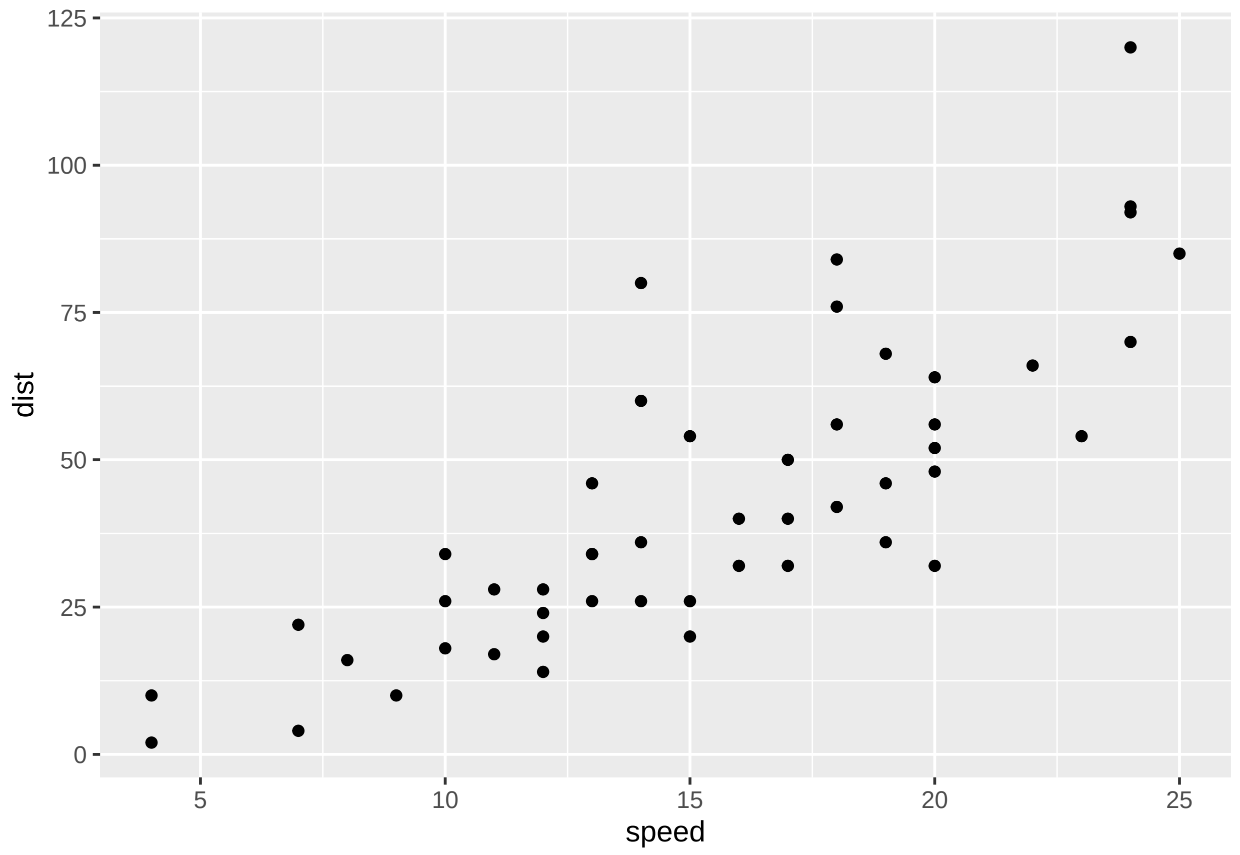
When installing packages, like we did before, run this command in the console panel rather than in the notebook panel. You'll only ever need to do this once per package (per computer).

This package specializes in creating customized plots and graphs. To load all the functions that come with ggplot2, use the library command:

```
library(ggplot2)
```

The library command loads third-party packages like ggplot2 that we've previously downloaded so that we can use the functions contained in them. Now consider this alternative way of plotting the same graph:

```
ggplot(data = cars, aes(x = speed, y = dist)) +
  geom_point()
```



It's the same graph, but using the ggplot package. ggplot() is a function, cars is its "data" argument, and aes (short for aesthetic) is an argument that itself has arguments: x = speed tells it to treat the speed column in cars as the x-axis variable and y = dist tells it to treat the dist column as the y-axis variable.

We end the line with a "+" to denote we want to add an additional plot element. Here, we wanted a scatterplot so we use the geom_point() function with no argument (nothing in its parentheses). This may seem much more complicated than the standard plot function, but once you get a hang of this sort of coding grammar, it allows us to make a lot of easy customizations:

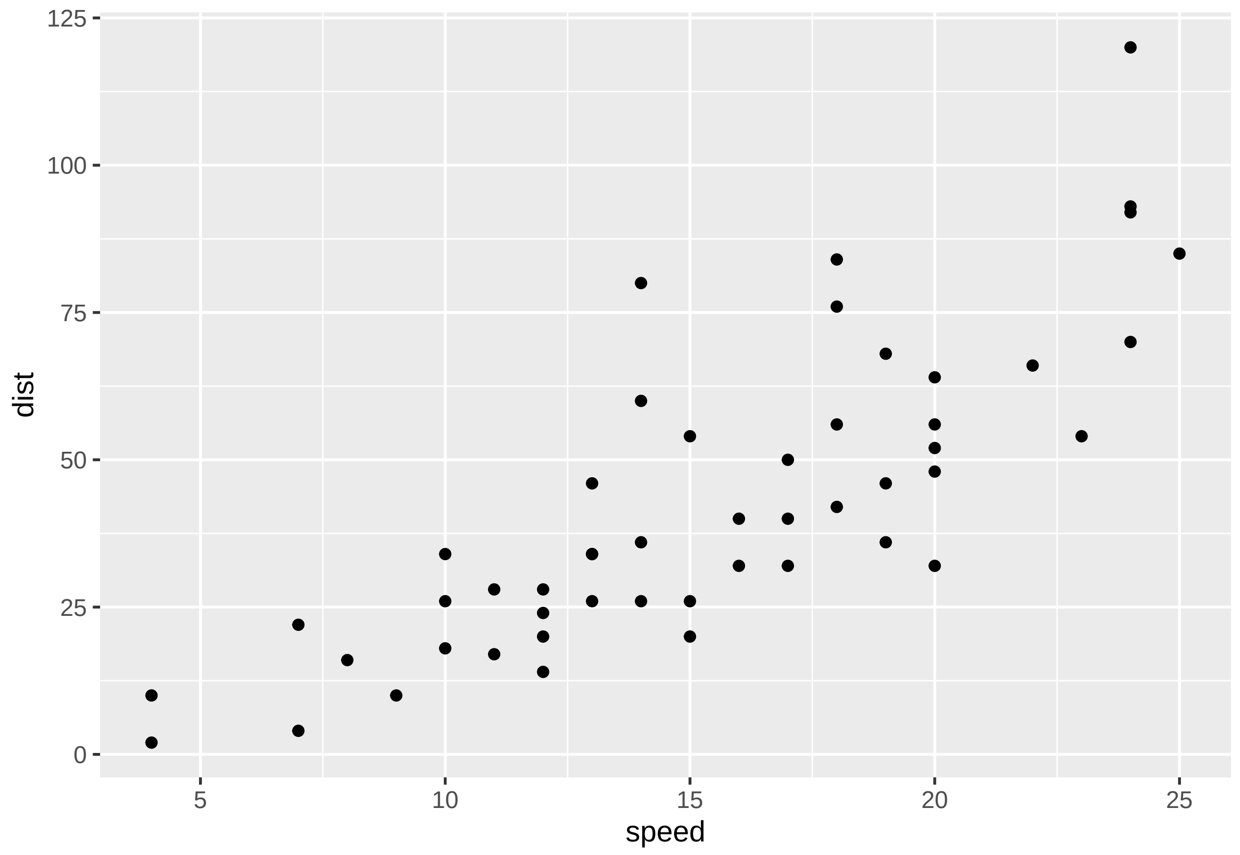
Let us save the graph above as an object. To do this, we come up with some name (let's say "test.plot") and assign the graph to it using the "<-" assignment:

```
test.plot <- ggplot(data = cars, aes(x = speed, y = dist)) +
  geom_point()
```

The first line calls the ggplot function and tells it what dataset we want to use and which variables we want to use as our x and y variables. The second line choose the kind of graph we want, a scatterplot. The "+" links the two lines as one command.

Now we have an object called test.plot which is the above graph. You can see a list of all the objects in our 'environment' in the 'Environment' panel in RStudio. We can plot this object:

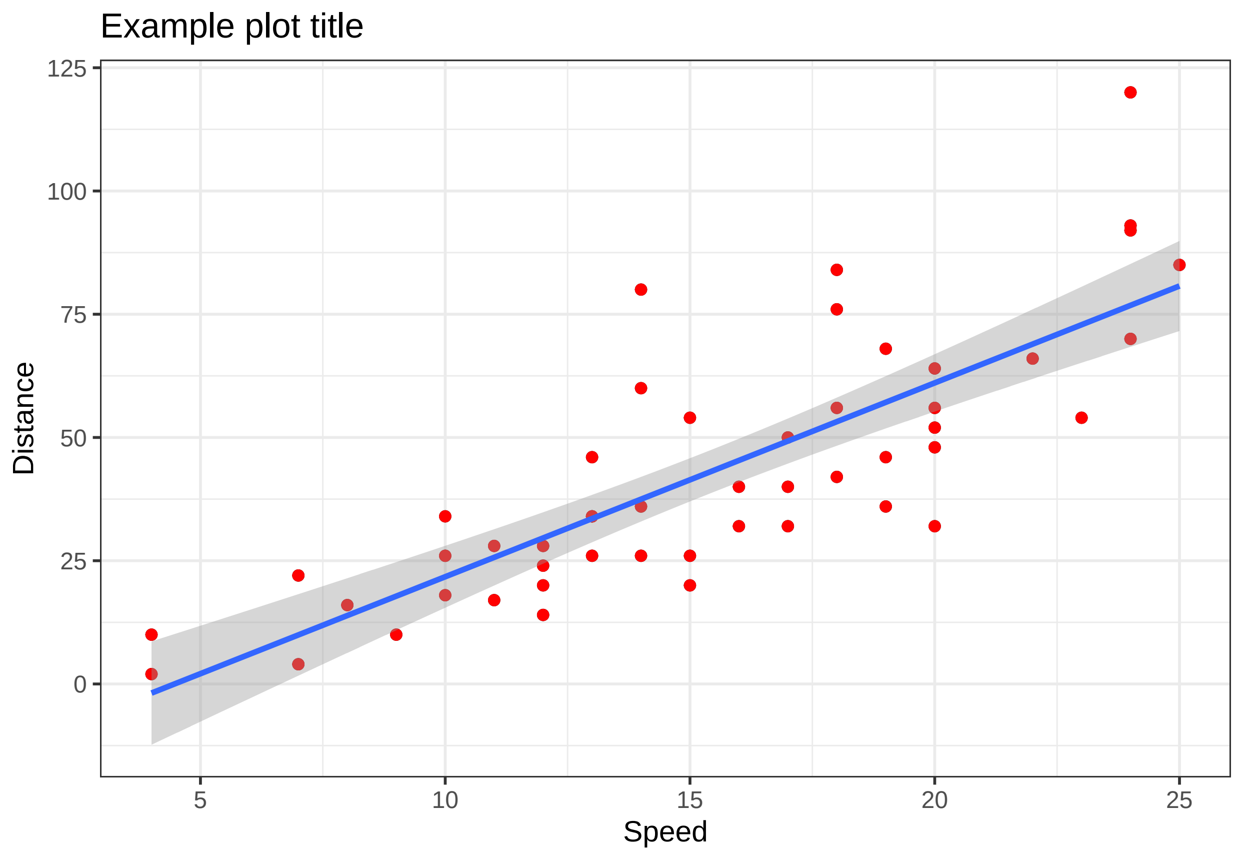
```
print(test.plot)
```



And we can make some new modifications, thanks to the grammar of ggplot2. Let's define the modified plot as a new object called test.plot2:

```
test.plot2 <- test.plot +
  # Change the point colors to red
  geom_point(col = 'red') +
  # Add a line of best fit with a confidence interval
  geom_smooth(method = 'lm') +
  # Modify the axis titles
  ylab('Distance') +
  xlab('Speed') +
  ggtitle('Example plot title') +
  # Simplify the plot theme to black-and-white
  theme_bw()
print(test.plot2)
```

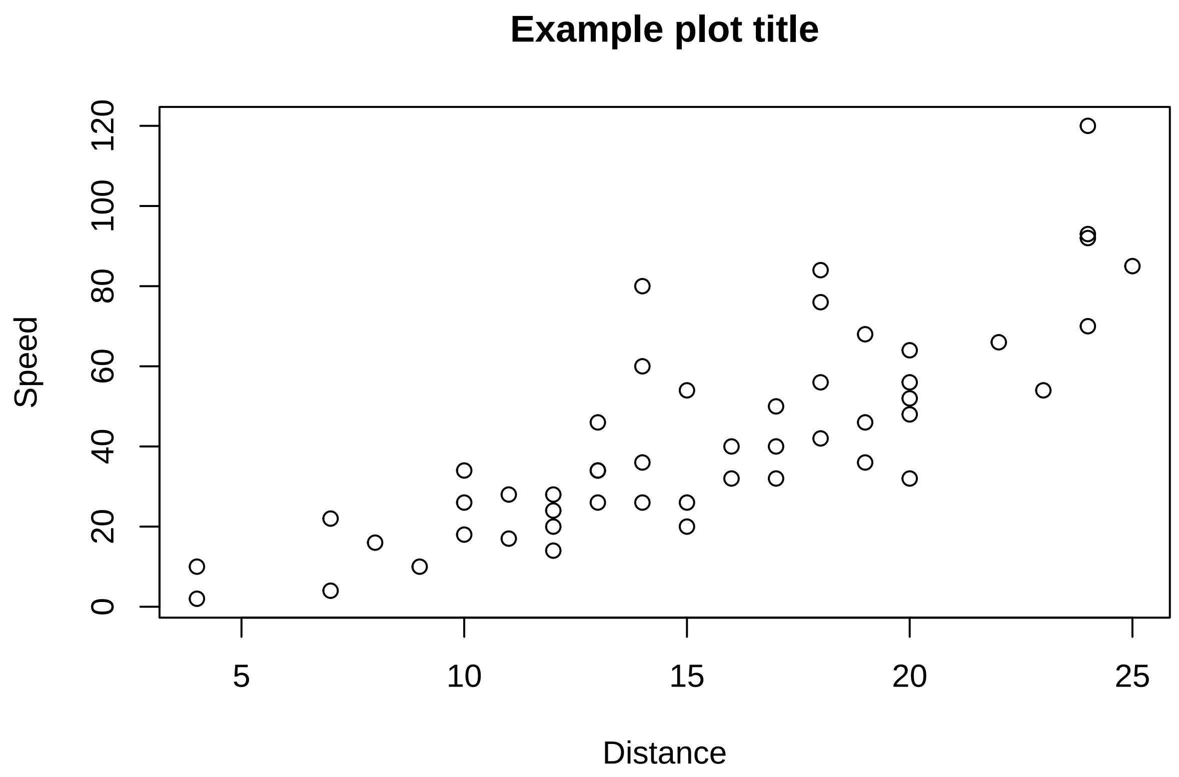
```
## `geom_smooth()` using formula 'y ~ x'
```



Notice that we can add comments to our chunks of code by using "#" at the end of a line. Anything after the "#" will be ignored by R until the next line of code. This is useful for communicating to yourself or your reader what each line.

Of course, we can also modify the standard "plot" function:

```r
plot(cars, main = 'Example plot title', xlab = 'Distance', ylab = 'Speed')
```



Again, ggplot may seem much more complicated than the simple plot command, but a little more effort early on will pay off later on when we do other types of plots because its modifiability is intuitive.

Part 3: Loading data

The way R Notebooks work, the “working directory” is the folder in which the notebook is saved. This means that if you want to open a file, it is most convenient to have that file in the same folder as the notebook. For example, I have a Stata dataset called “animals.dta” (all Stata datasets end with .dta) in my “Recitation 1” folder. Let’s open that file.

First note that since R by default cannot open a Stata dataset, we must download/install a package that can. So just like we did with the ggplot package, run this command in the console panel:

install.packages(‘readstata13’)

Then load the package:

```
library(readstata13)
```

Then we’ll use the “read.dta13” command from this package to read the file in our working directory. Let’s call the dataset “animals” by using the assignment notation from before:

```
animals <- read.dta13('animals.dta')
```

Note you put the filename/filepath ‘animals.dta’ in quotes since it is not an object in our environment

Let’s get a quick summary of this data as we did with the cars dataset:

```r
dim(animals) # What are the dimensions (number of rows, number of columns) of this dataset
```

```
## [1] 790   7
```

```r
nrow(animals) # Same as above, but just the number of rows
```

```
## [1] 790
```

```r
ncol(animals) # The number of columns
```

```
## [1] 7
```

```r
head(animals) # The first few observations
```

```
##   village hhn id   animal          type number price
## 1       1   1  1    Goats   Calves-Male      3 15000
## 2       1   1  1 Chickens        Layers      3  3000
## 3       1   1  1    Goats Calves-Female      2 15000
## 4       1   1  1    Goats        Female      4 25000
## 5       1   2  0    Goats Calves-Female      3  9000
## 6       1   2  0    Sheep   Calves-Male      3    dk
```

```r
summary(animals) # A brief summary of each variable in the dataset
```

```
##     village           hhn              id             animal         
##  Min.   :1.000   Min.   : 1.00   Min.   :0.0000   Length:790        
##  1st Qu.:2.000   1st Qu.:16.00   1st Qu.:0.0000   Class :character  
##  Median :3.000   Median :33.00   Median :0.0000   Mode  :character  
##  Mean   :2.647   Mean   :34.33   Mean   :0.4924                     
##  3rd Qu.:4.000   3rd Qu.:50.75   3rd Qu.:1.0000                     
##  Max.   :4.000   Max.   :99.00   Max.   :2.0000                     
##      type              number             price          
##  Length:790         Length:790         Length:790        
##  Class :character   Class :character   Class :character  
##  Mode  :character   Mode  :character   Mode  :character  
##                                                          
##                                                          
##                                                          
```

We can see that different variables are of different types: village, hhn, and id are numbers (that's why we can compute its mean and maximum) while others are "characters" or "strings", i.e. its values are text entries like "Goats" or "Chickens". We will be interested in both types throughout this course.

Another way we can summarize this data is by tabulation. This lets us look at the frequency of each value of a variable. For example:

```r
table(animals$animal)
```

```
##
##      Chickens          Ducks          Goats    Guinea Fowl Other Poultry
##           307             30            251              5             4
##           Pig        Rabbits          Sheep         Turkey
##             4              3            180              6
```

This tells us how often each animal appears in our dataset. Notice that we can refer to a specific column in the animals dataset by its name using the "$" character. "animals$animal" refers to the column named "animal" in the "animals" object (i.e. dataset in this case). We could have also referred to the animals$village column. We'll use this often.

Part 4: Regressions

The most important thing we'll do in this course is run regression models of various types. Going back to the cars dataset, let's suppose we want to run a very simple univariate regression of speed on distance. Let's create a model object called "cars.mod" and use the "lm" function (short for linear regression) to run a regression on the cars dataset:

```
cars.mod <- lm(speed ~ dist, data = cars)
```

Here, the first argument is a formula "speed ~ dist" meaning speed is our outcome variable and distance is our single regressor. The second argument tells us which object/dataset these variables should come from. This command then gives us an object called "cars.mod" that is a model. If we wanted to print the regression output, we would use the "summary" function on this object:

```
summary(cars.mod)
```

```
##
## Call:
## lm(formula = speed ~ dist, data = cars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.5293 -2.1550  0.3615  2.4377  6.4179
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  8.28391    0.87438   9.474 1.44e-12 ***
## dist         0.16557    0.01749   9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.156 on 48 degrees of freedom
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

We can clearly see this gives us a y-intercept (i.e. a constant) of 8.28391 and a coefficient estimate of 0.16557 on distance. We interpret this as saying an increase in distance by 1km (or whatever unit distance is in) is associated with an increase in speed of 0.16557. The intercept and distance coefficients each have standard errors, t-values, and p-values clearly associated with them, which we care about for inference.

We can do some more things with our model object, for example:

```
# If we only cared about the coefficients:
coef(cars.mod)
```

```
## (Intercept)        dist
##   8.2839056   0.1655676
```

```
# If we want to extract the residuals (don't worry if you don't know what these are for yet)
cars.mod$residuals
```

```
##           1           2           3           4           5           6
## -4.61504079 -5.93958139 -1.94617594 -4.92639228 -2.93298684 -0.93958139
##           7           8           9          10          11          12
## -1.26412199 -2.58866258 -3.91320318 -0.09855441 -1.91979773  1.39814831
##          13          14          15          16          17          18
##  0.40474287 -0.25752743 -0.91979773  0.41133742 -0.91320318 -0.91320318
##          19          20          21          22          23          24
## -2.90001408  1.41133742 -0.24433833 -4.21796012 -7.52931161  3.40474287
##          25          26          27          28          29          30
##  2.41133742 -2.22455467  2.41793197  1.09339137  3.41793197  2.09339137
##          31          32          33          34          35          36
##  0.43771563  2.76225622  0.44431018 -2.86704131 -4.19158191  4.75566167
##          37          38          39          40          41          42
##  3.09998592 -0.54250072  6.41793197  3.76885078  3.10658048  2.44431018
##          43          44          45          46          47          48
##  1.11976958  2.78863443  5.77544533  4.12636413  0.48387749  0.31830992
##          49          50
## -4.15201460  2.64285051
```

```
# If we want to extract the model's predicted/fitted values of y
cars.mod$fitted.values
```

```
##         1         2         3         4         5         6         7         8
##  8.615041  9.939581  8.946176 11.926392 10.932987  9.939581 11.264122 12.588663
##         9        10        11        12        13        14        15        16
## 13.913203 11.098554 12.919798 10.601852 11.595257 12.257527 12.919798 12.588663
##        17        18        19        20        21        22        23        24
## 13.913203 13.913203 15.900014 12.588663 14.244338 18.217960 21.529312 11.595257
##        25        26        27        28        29        30        31        32
## 12.588663 17.224555 13.582068 14.906609 13.582068 14.906609 16.562284 15.237744
##        33        34        35        36        37        38        39        40
## 17.555690 20.867041 22.191582 14.244338 15.900014 19.542501 13.582068 16.231149
##        41        42        43        44        45        46        47        48
## 16.893420 17.555690 18.880230 19.211366 17.224555 19.873636 23.516123 23.681690
##        49        50
## 28.152015 22.357149
```

```
# And if you want to save any of these as separate objects to be referred to later:
cars.coefs <- coef(cars.mod)
cars.resid <- cars.mod$residuals
cars.fit <- cars.mod$fitted.values

summary(cars.mod)
```

```
##
## Call:
## lm(formula = speed ~ dist, data = cars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.5293 -2.1550  0.3615  2.4377  6.4179
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  8.28391    0.87438   9.474 1.44e-12 ***
## dist         0.16557    0.01749   9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.156 on 48 degrees of freedom
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

Part 5: A note about working directories, R Notebooks, and knitting

It is very common to begin an R script or R notebook with a chunk that sets your working directory and loads the packages that we are going to use in the script or Notebook. An example might be something like

```
setwd('~/Documents/Grad School/Columbia/Y3/Metrics TA/Recitation 3')
library(readstata13)
library(ggplot2)
```

setwd() here sets my working directory to the given folder (obviously adapt that line to the folder you like for your computer). This tells R to look in that folder whenever we refer to a file (usually a dataset). For example, this folder is where this Notebook is saved in my computer and it's also where a dataset called 'Growth.dta', which we use in this problem set is saved. Now I can load this dataset easily:

```
growth <- read.dta13('Growth.dta')
head(growth)
```

```
##    country_name    growth oil    rgdp60 tradeshare yearsschool rev_coups
## 1         India 1.9151679   0  765.9998  0.1405020        1.45 0.1333333
## 2     Argentina 0.6176451   0 4462.0015  0.1566230        4.99 0.9333333
## 3         Japan 4.3047590   0 2953.9995  0.1577032        6.71 0.0000000
## 4        Brazil 2.9300966   0 1783.9999  0.1604051        2.89 0.1000000
## 5 United States 1.7122649   0 9895.0039  0.1608150        8.66 0.0000000
## 6    Bangladesh 0.7082631   0  951.9998  0.2214584        0.79 0.3064815
##   assasinations
## 1     0.8666667
## 2     1.9333333
## 3     0.2000000
## 4     0.1000000
## 5     0.4333333
## 6     0.1750000
```

Critically, when we use R Notebooks to produce pdfs by ‘Knitting’, it essentially runs R from scratch as if we had never loaded any packages or defined any objects. It also assumes the working directory is the folder where the R Notebook is saved even if you’ve used ‘setwd()’. In practice, this might be different from the working directory of your environment. It’s generally not the default working directory when you open RStudio, for example.

Secondly, knitting runs the code in our code chunks in the order that they’re written here. If you’ve defined an object called ‘data’ and at some point you run a command like ‘mean(data)’, you have to make sure the object data is defined earlier than the command mean(data) is run or else R won’t know what ‘data’ is and it won’t produce the pdf. They can be in different chunks, it’s just the order of appearance that matters. Similarly, if you defined an object but you didn’t do so in your Notebook, R won’t know what that object is when knitting and it won’t produce a pdf.

If you run into any trouble related to this, do email me or post a question on Piazza and if this causes any issues. Google is also your friend here. I think this is the main thing that might cause confusion the first time we use R Notebooks so do let me know!

Part 6: Creating a data.frame

Often we’ll be interested in building a dataset manually rather than load an existing dataset into R. We’ll do this by constructing what are called data.frame objects, which is basically just a spreadsheet. Rows are observations and columns are variables.

I can’t do the example from the problem set for you, but let’s look at another dataset, one that is inbuilt into R.

(As an aside, if you want to see the list of all datasets that are inbuilt into R, you can type “data()” as a command)

```
head(ToothGrowth)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

This one is called “ToothGrowth”. For context, these are the results from an experiment studying the effect of vitamin C on tooth growth in 60 Guinea pigs. Each animal received one of three dose levels of vitamin C (0.5, 1, and 2 mg/day) by one of two delivery methods, (orange juice or ascorbic acid (a form of vitamin C and coded as VC).

That isn’t important for our purposes, we just want to use it to learn about creating datasets by hand. Let’s create these first six rows by defining objects called vectors. We use the “c” command to do this.

```
x1 <- c(4.2, 11.5, 7.3, 5.8, 6.4, 10.0)
x2 <- c('VC', 'VC', 'VC', 'VC', 'VC', 'VC')
x3 <- c(0.5, 0.5, 0.5, 0.5, 0.5, 0.5)
```

Notice that when we input words, we have to put them in quotation marks but numbers are fine as they are.

We can then easily combine these into a data.frame object that we’ll call tooth.data

```r
tooth.data <- data.frame(x1, x2, x3)
tooth.data
```

```
##     x1 x2  x3
## 1  4.2 VC 0.5
## 2 11.5 VC 0.5
## 3  7.3 VC 0.5
## 4  5.8 VC 0.5
## 5  6.4 VC 0.5
## 6 10.0 VC 0.5
```

Which is a nice recreation of the original dataset (at least its first six rows). We also could've very easily assigned names to the variables:

```r
tooth.data <- data.frame(len = x1,
                         supp = x2,
                         dose = x3)
tooth.data
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

This makes it convenient when we want to perform some commands on the data. We simply refer to the data.frame *tooth.data* and its variable *len* or *supp* or *dose* by combining them with a dollar sign symbol: *tooth.data$len*.

Let's look at some applications of this, but let's use the full set of observations in the dataset instead of just the first 6. This means just redefining tooth.data as the original dataset:

```r
tooth.data <- ToothGrowth
```

Some summary statistics

```r
# Finding the mean
mean(tooth.data$len)
```

```
## [1] 18.81333
```

```r
# Finding the standard deviation and variance
sd(tooth.data$len)
```

```
## [1] 7.649315
```

```
var(tooth.data$len)
```

```
## [1] 58.51202
```

```
# A general summary of the data
summary(tooth.data)
```

```
##       len        supp         dose      
##  Min.   : 4.20   OJ:30   Min.   :0.500  
##  1st Qu.:13.07   VC:30   1st Qu.:0.500  
##  Median :19.25           Median :1.000  
##  Mean   :18.81           Mean   :1.167  
##  3rd Qu.:25.27           3rd Qu.:2.000  
##  Max.   :33.90           Max.   :2.000
```

```
# Correlation between length and dose
cor(tooth.data$len, tooth.data$dose)
```

```
## [1] 0.8026913
```

Back to regressions

And let's do some basic regression commands again:

```
tooth.model <- lm(len ~ dose, data = tooth.data)
# Look at the model output
summary(tooth.model)
```

```
## 
## Call:
## lm(formula = len ~ dose, data = tooth.data)
## 
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -8.4496 -2.7406 -0.7452  2.8344 10.1139 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept)   7.4225     1.2601    5.89 2.06e-07 ***
## dose          9.7636     0.9525   10.25 1.23e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 4.601 on 58 degrees of freedom
## Multiple R-squared:  0.6443, Adjusted R-squared:  0.6382 
## F-statistic: 105.1 on 1 and 58 DF,  p-value: 1.233e-14
```

If we want robust standard errors (which we often will), we will need another package called estimatr (so install this if you haven't already):

```
library(estimatr)
tooth.model.robust <- lm_robust(len ~ dose, data = tooth.data,
                                se_type = 'stata')
summary(tooth.model.robust)
```

```
##
## Call:
## lm_robust(formula = len ~ dose, data = tooth.data, se_type = "stata")
##
## Standard error type:  HC1
##
## Coefficients:
##             Estimate Std. Error t value  Pr(>|t|) CI Lower CI Upper DF
## (Intercept)    7.423      1.285   5.775 3.196e-07     4.85    9.995 58
## dose           9.764      0.881  11.082 6.011e-16     8.00   11.527 58
##
## Multiple R-squared:  0.6443 ,    Adjusted R-squared:  0.6382
## F-statistic: 122.8 on 1 and 58 DF,  p-value: 6.011e-16
```

The "SE type" argument here makes sure we get the same standard errors that we would using Stata's robust command.

Maybe we want to add the residuals and fitted/predicted values from this regression to the original dataset:

```
# The original
head(tooth.data)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

```
# Now let's create variables called residuals and predictions
tooth.data$residuals <- tooth.model$residuals
tooth.data$predictions <- tooth.model$fitted.values

# New dataset
head(tooth.data)
```

```
##    len supp dose  residuals predictions
## 1  4.2   VC  0.5 -8.1042857    12.30429
## 2 11.5   VC  0.5 -0.8042857    12.30429
## 3  7.3   VC  0.5 -5.0042857    12.30429
## 4  5.8   VC  0.5 -6.5042857    12.30429
## 5  6.4   VC  0.5 -5.9042857    12.30429
## 6 10.0   VC  0.5 -2.3042857    12.30429
```

We also could've done the same thing in the following way:

```
tooth.residuals <- tooth.model$residuals
tooth.predictions <- tooth.model$fitted.values
tooth.data <- data.frame(ToothGrowth,
                         residuals = tooth.residuals,
                         predictions = tooth.predictions)
head(tooth.data)
```

```
##    len supp dose  residuals predictions
## 1  4.2   VC  0.5 -8.1042857    12.30429
## 2 11.5   VC  0.5 -0.8042857    12.30429
## 3  7.3   VC  0.5 -5.0042857    12.30429
## 4  5.8   VC  0.5 -6.5042857    12.30429
## 5  6.4   VC  0.5 -5.9042857    12.30429
## 6 10.0   VC  0.5 -2.3042857    12.30429
```

Let's return to the predicted linear model:

```
summary(tooth.model)
```

```
##
## Call:
## lm(formula = len ~ dose, data = tooth.data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.4496 -2.7406 -0.7452  2.8344 10.1139
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   7.4225     1.2601    5.89 2.06e-07 ***
## dose          9.7636     0.9525   10.25 1.23e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.601 on 58 degrees of freedom
## Multiple R-squared:  0.6443, Adjusted R-squared:  0.6382
## F-statistic: 105.1 on 1 and 58 DF,  p-value: 1.233e-14
```

We can also summarize it through

```
library(lmtest) # Install this package if you haven't already
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
coeftest(tooth.model)
```

```
##
## t test of coefficients:
##
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  7.42250    1.26008  5.8905 2.064e-07 ***
## dose         9.76357    0.95253 10.2501 1.233e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Let's create a 95% confidence interval for our coefficient estimates using the *confint* function:

```
confint(tooth.model)
```

```
##                2.5 %    97.5 %
## (Intercept) 4.900171  9.944829
## dose        7.856870 11.670273
```

```
confint(tooth.model.robust)
```

```
##                2.5 %    97.5 %
## (Intercept) 4.849592  9.995408
## dose        7.999997 11.527146
```

We could also do this for different confidence levels. For example, a 99% confidence interval:

```
confint(tooth.model, level = 0.99)
```

```
##                0.5 %   99.5 %
## (Intercept) 4.066538 10.77846
## dose        7.226703 12.30044
```

A little bit about typesetting equations in R Notebooks using TeX

We can write the implied regression model very nicely in R using the language of TeX as shown in the first week:

$$\hat{Length} = 7.4225 + 9.7636 \times Dosage_i$$

TeX might take a while to get used to but the basics are that an underscore _ puts everything in the curly brackets as a subscript. We can also use ˆ to superscript (for example, for exponents) and we use the hat command to signify something is a predicted quantity.

Now let's get to the practice problems!

Practice Problem 5: SW Empirical Exercise 4.1

Part a

Load the growth dataset

```
library(readstata13) # If you've already loaded the package earlier, you don't need this line
growth <- read.dta13('Growth.dta') # File names are case sensitive
head(growth)
```

```
##    country_name    growth oil    rgdp60 tradeshare yearsschool rev_coups
## 1         India 1.9151679   0  765.9998  0.1405020        1.45 0.1333333
## 2     Argentina 0.6176451   0 4462.0015  0.1566230        4.99 0.9333333
## 3         Japan 4.3047590   0 2953.9995  0.1577032        6.71 0.0000000
## 4        Brazil 2.9300966   0 1783.9999  0.1604051        2.89 0.1000000
## 5 United States 1.7122649   0 9895.0039  0.1608150        8.66 0.0000000
## 6    Bangladesh 0.7082631   0  951.9998  0.2214584        0.79 0.3064815
##   assasinations
## 1     0.8666667
## 2     1.9333333
## 3     0.2000000
## 4     0.1000000
## 5     0.4333333
## 6     0.1750000
```

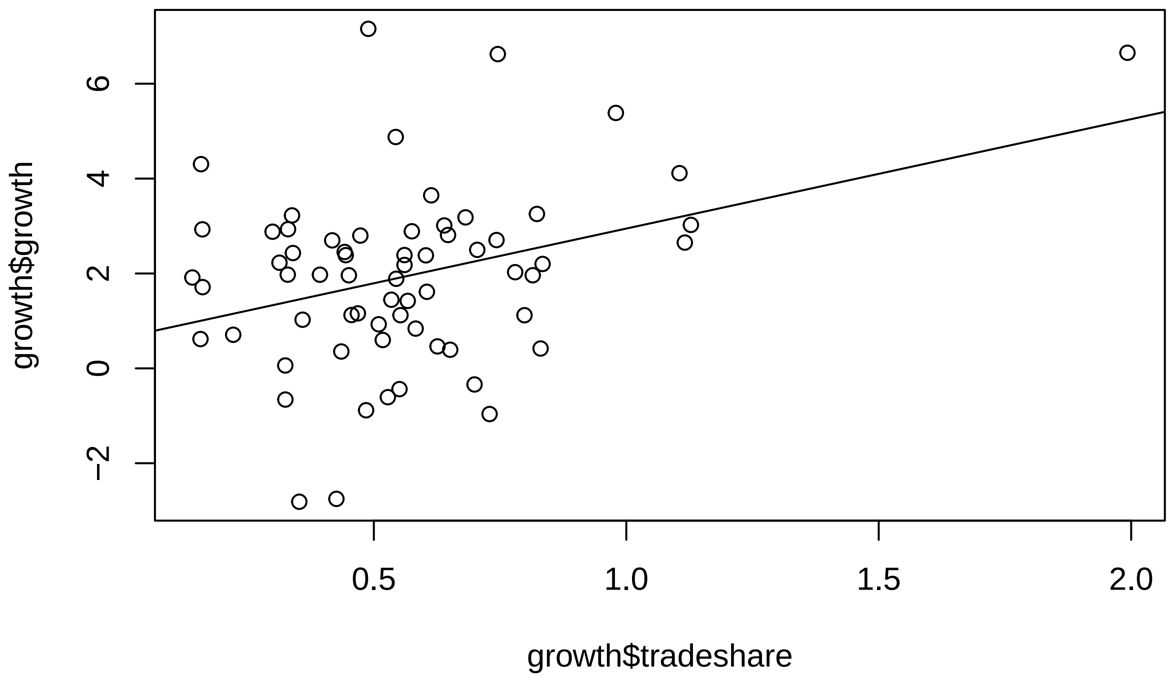
We want to construct a scatterplot of growth on average tradeshare:

We can do this in a couple of ways.

Method 1: Base R

```
# Define the linear model
growth.model <- lm(growth ~ tradeshare, data = growth)

{plot(growth$tradeshare, growth$growth)
abline(growth.model)}
```

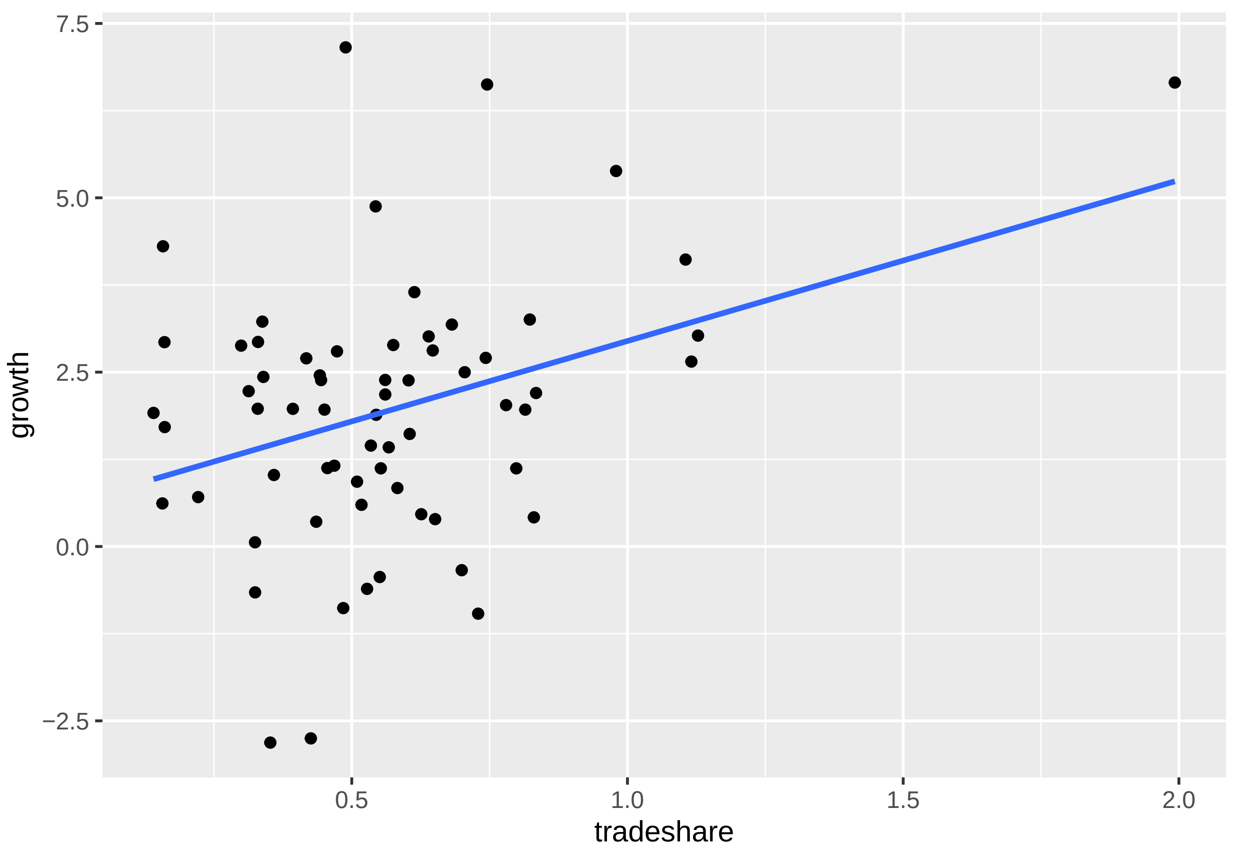


Notice that using Base R, we have to combine the plot and abline functions using curly brackets. This is just a quirk of R Notebooks.

Method 2: ggplot2

```
library(ggplot2) # If you've already loaded the package earlier, you don't need this line
ggplot(growth, aes(x = tradeshare, y = growth)) +
  geom_point() +
  geom_smooth(method = 'lm', se = F)
```

```
## ‘geom_smooth()‘ using formula ’y ~ x’
```



Part b

Notice that there is a very anomalous value here whose tradeshare is far greater than any other country. We might want to drop it.

Part c

```
summary(growth.model)
```

```
##
## Call:
## lm(formula = growth ~ tradeshare, data = growth)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.3739 -0.8864  0.2329  0.9248  5.3889
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.6403     0.4900   1.307  0.19606
## tradeshare    2.3064     0.7735   2.982  0.00407 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 1.79 on 63 degrees of freedom
## Multiple R-squared:  0.1237, Adjusted R-squared:  0.1098
## F-statistic: 8.892 on 1 and 63 DF,  p-value: 0.00407
```

Countries with tradeshares of 0.5 and 1.0 will have predicted growth rates

```
0.6403 + 2.3*0.5
```

```
## [1] 1.7903
```

```
0.6403 + 2.3*1
```

```
## [1] 2.9403
```

Part d

Let's use the *dplyr* package (install if you haven't) to modify the dataset. dplyr is one of the most popular packages in R so we'll probably be using it a lot.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
growth2 <- filter(growth, tradeshare <= 1.5)
growth.model2 <- lm(growth ~ tradeshare, data = growth2)
```

This creates a new dataset that is the same as the original dataset but filters out any observations whose tradeshare is less than or equal to 1.5.

Alternatively we could do this in one line without defining a new dataset:

```
growth.model2 <- lm(growth ~ tradeshare,
                    data = filter(growth, tradeshare <= 1.5))
```

Then we have:

```
summary(growth.model2)
```

```
##
## Call:
## lm(formula = growth ~ tradeshare, data = filter(growth, tradeshare <=
##     1.5))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.4247 -0.9383  0.2091  0.9265  5.3776
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.9574     0.5804   1.650   0.1041
## tradeshare    1.6809     0.9874   1.702   0.0937 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.789 on 62 degrees of freedom
## Multiple R-squared:  0.04466,    Adjusted R-squared:  0.02925
## F-statistic: 2.898 on 1 and 62 DF,  p-value: 0.09369
```

```
0.9574 + 1.6809*0.5
```

```
## [1] 1.79785
```

```
0.9574 + 1.6809*1
```

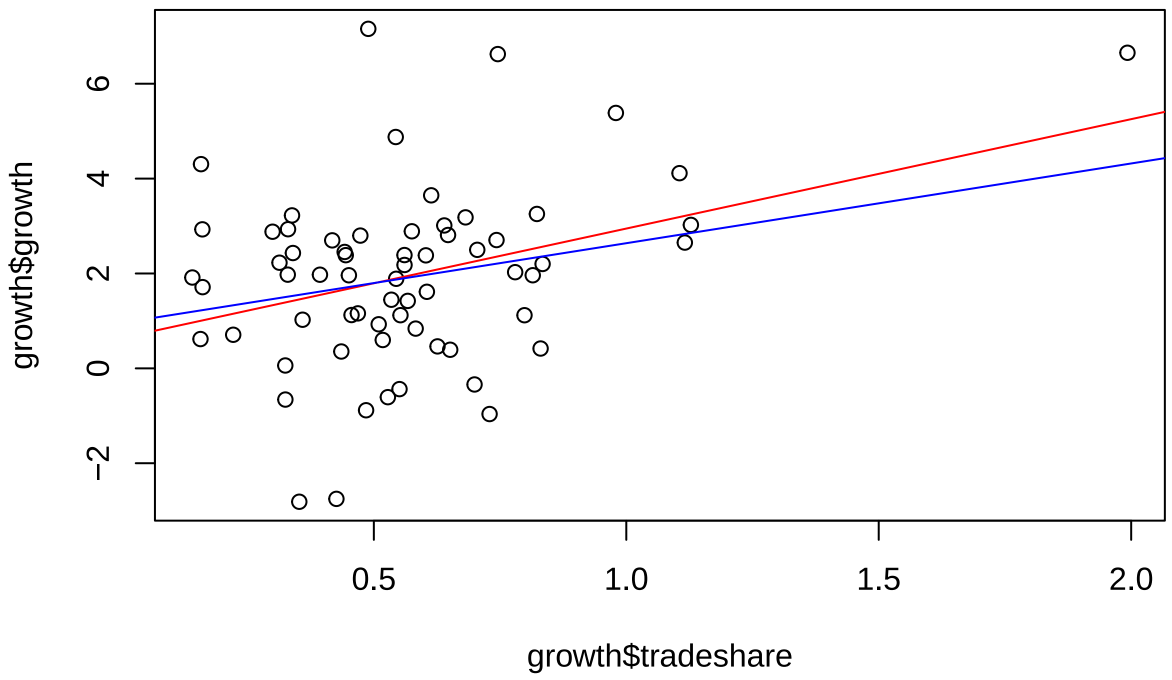
```
## [1] 2.6383
```

Part e

We'll plot both lines of best fit on the original scatterplot

Method 1:

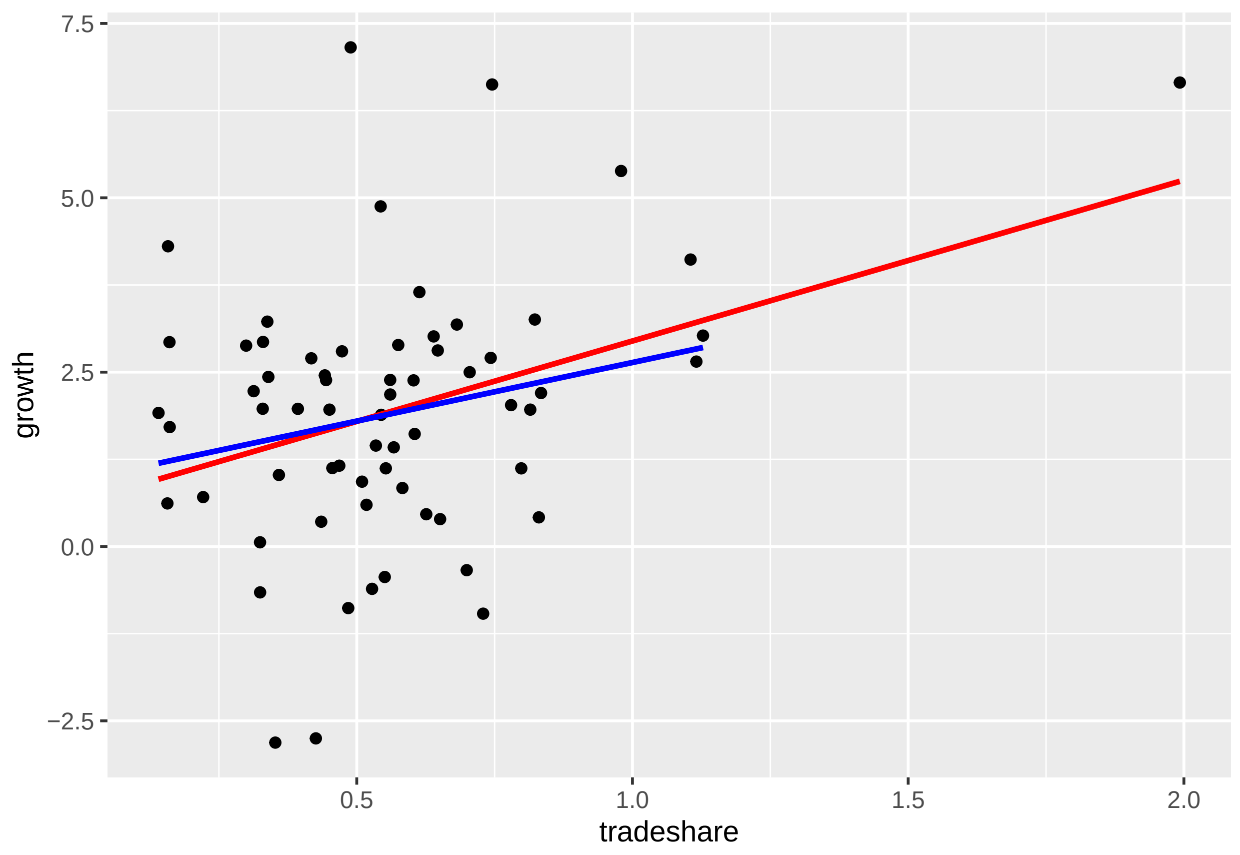
```
{plot(growth$tradeshare, growth$growth)
abline(growth.model, col = 'red')
abline(growth.model2, col = 'blue')}
```



Method 2: ggplot2

```r
ggplot(growth, aes(x = tradeshare, y = growth)) +
  geom_point() +
  geom_smooth(method = 'lm', se = F, color = 'red') +
  geom_smooth(method = 'lm', se = F, color = 'blue', data = growth2)
```

```
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
```



Here, we specified the filtered growth dataset as an argument for the second line of best fit. The blue line is shorter because the Malta point is not in that dataset. In comparison, the base R graph just has the lines extending indefinitely in both directions.

Part f

Something about Malta being a small island nation, etc.

Practice Problem 6: SW Empirical Exercise 4.2

Let's load the new dataset

```
earnings <- read.dta13('Earnings_and_Height.dta',
                       generate.factors = T,
                       nonint.factors = T)
head(earnings)
```

```
##        sex age                  mrd educ           cworker      region
## 1 0:female  48 1:Married, sps in hh   13 1:Private company     3:South
## 2 0:female  41      6:Never Married   12 1:Private company   2:Midwest
## 3 0:female  26 1:Married, sps in hh   16 1:Private company 1:Northeast
## 4 0:female  37 1:Married, sps in hh   16 1:Private company   2:Midwest
## 5 0:female  35      6:Never Married   16 1:Private company 1:Northeast
## 6 0:female  25      6:Never Married   15 1:Private company      4:West
##                   race earnings height weight occupation
## 1 non-hispanic white 84054.75     65    133          1
## 2 non-hispanic white 14021.39     65    155          1
## 3 non-hispanic white 84054.75     60    108          1
## 4 non-hispanic white 84054.75     67    150          1
## 5 non-hispanic white 28560.39     68    180          1
## 6 non-hispanic white 23362.87     63    101          1
```

Part a

```
median(earnings$height)
```

```
## [1] 67
```

Part b

There are a number of ways to do these conditional statistic

Part b-i

Average earnings for workers whose height is at most 67 inches

```
# Method 1
mean(filter(earnings, height <= 67)$earnings)
```

```
## [1] 44488.44
```

```
# Method 2
earnings2 <- filter(earnings, height <= 67)
mean(earnings2$earnings)
```

```
## [1] 44488.44
```

```
# Method 3 (a bit more advanced)
earnings %>%
  filter(height <= 67) %$%
  earnings %>%
  mean
```

```
## [1] 44488.44
```

This last method breaks up the process in a very clear way but it takes some getting used to the grammar here. The '%>%' part is what is called a pipe. On the LHS is an object and on the RHS is a function. The pipe puts what's on the LHS as the first argument in the function on the RHS. The first pipe is equivalent to the first line in Method 2.

Similarly, the LHS of the '%$%' pipe is an object (here it is the output from the first pipe). The RHS of the is the name of a variable in the LHS object.

Finally, the last pipe takes this variable from the previous pipe and uses it as the first argument in the function on the RHS.

If this is confusing, then just ignore Method 3, but it is very convenient and clear to read once you gets used to it.

Part b-ii

Similar but for a different condition:

```
mean(filter(earnings, height > 67)$earnings)
```

```
## [1] 49987.88
```

Part b-iii

Let's define a dummy variable called *tall* to split the sample by height so we can compare tall-people earnings to short-people earnings. Then run a t-test:

```
earnings$tall <- earnings$height > 67
t.test(earnings ~ tall, data = earnings)
```

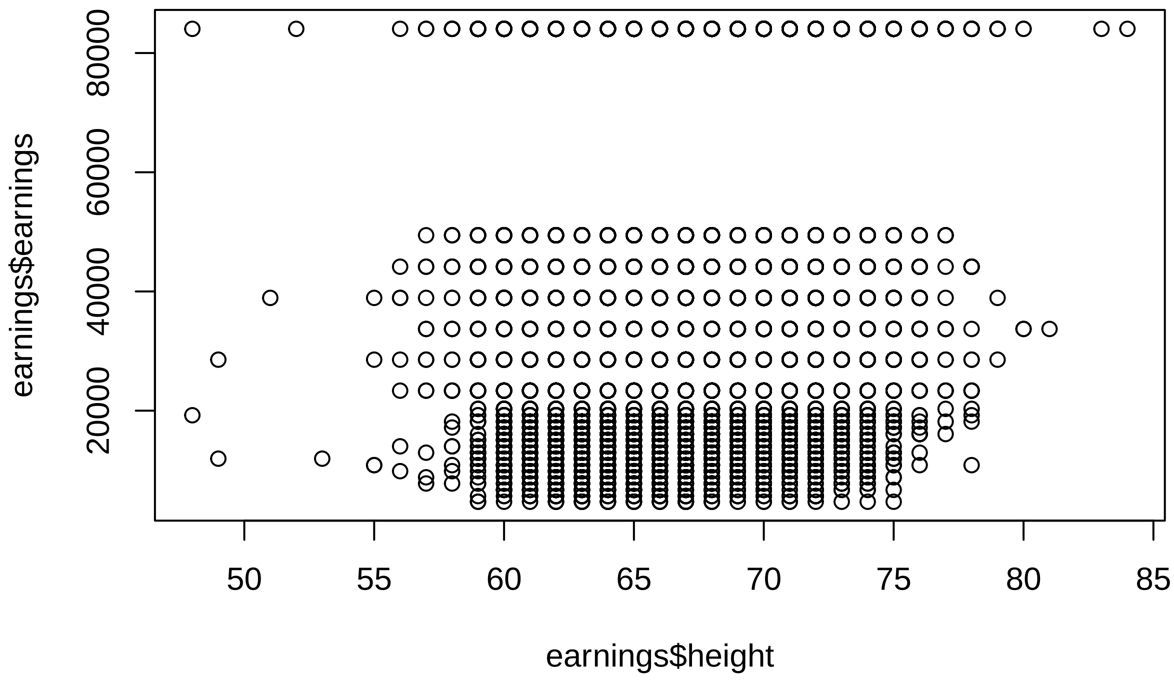
```
##
##  Welch Two Sample t-test
##
## data:  earnings by tall
## t = -13.59, df = 16624, p-value < 2.2e-16
## alternative hypothesis: true difference in means between group FALSE and group TRUE is not equal to
## 95 percent confidence interval:
##  -6292.643 -4706.237
## sample estimates:
## mean in group FALSE  mean in group TRUE
##            44488.44            49987.88
```

This gives us a 95% confidence interval for the difference in average earnings (4706.2 to 6292.6) and t-statistics and a p-value to test for significance in differences.

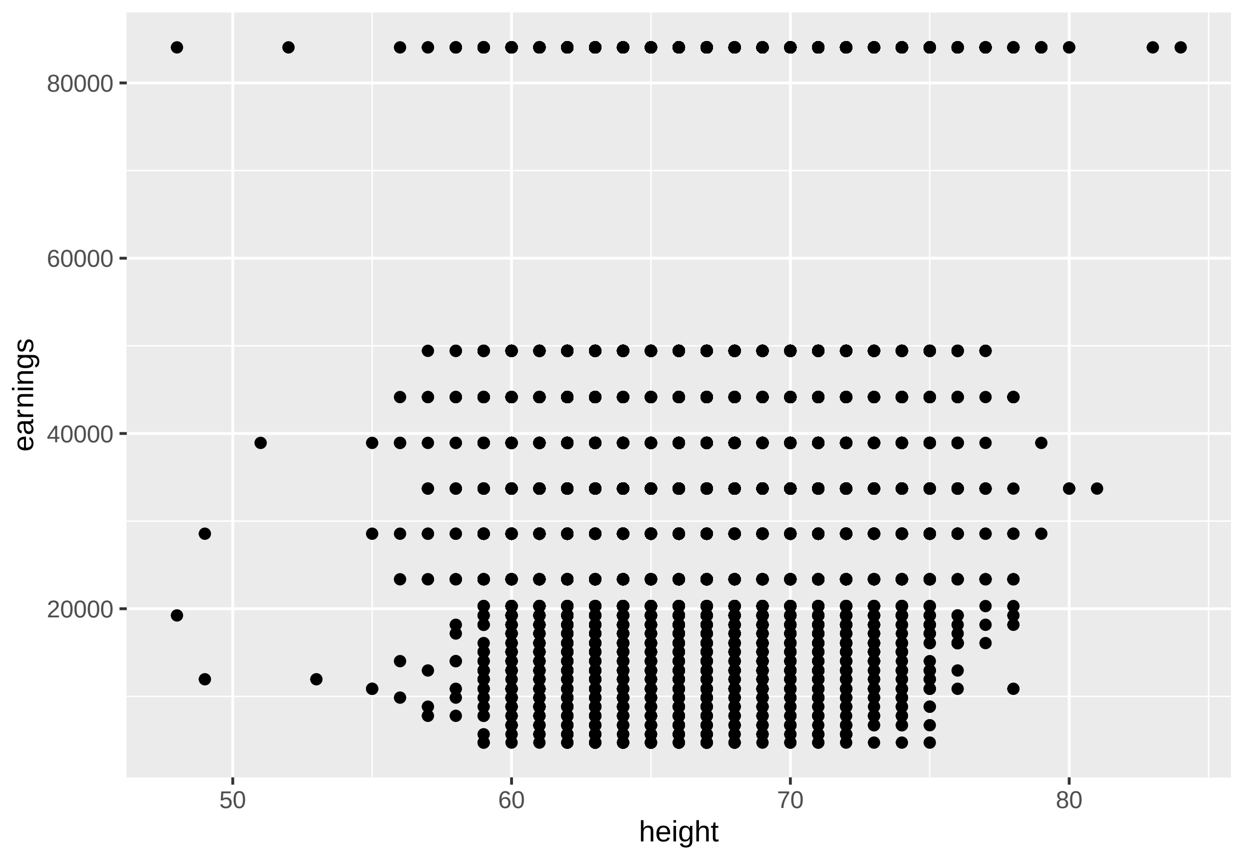
Part c

Scatterplot earnings and height

```
plot(earnings$height, earnings$earnings)
```



```
ggplot(earnings, aes(x = height, y = earnings)) +
  geom_point()
```



Part d

Regress earnings on height

```r
height.mod <- lm_robust(earnings ~ height, data = earnings)
summary(height.mod)
```

```
##
## Call:
## lm_robust(formula = earnings ~ height, data = earnings)
##
## Standard error type:  HC2
##
## Coefficients:
##             Estimate Std. Error t value  Pr(>|t|) CI Lower CI Upper    DF
## (Intercept)   -512.7     3380.0 -0.1517 8.794e-01  -7137.9   6112.4 17868
## height         707.7       50.4 14.0419 1.490e-44    608.9    806.5 17868
##
## Multiple R-squared:  0.01088 ,   Adjusted R-squared:  0.01082
## F-statistic: 197.2 on 1 and 17868 DF,  p-value: < 2.2e-16
```

```r
# Part d-i
coefficients(height.mod)
```

```
## (Intercept)      height
##   -512.7336    707.6716
```

```
# Part d-ii
-512.7336 + 707.67*67
```

```
## [1] 46901.16
```

```
-512.7336 + 707.67*70
```

```
## [1] 49024.17
```

```
-512.7336 + 707.67*65
```

```
## [1] 45485.82
```

Part e

Let's convert the height to centimeters

```
earnings$heightcm <- earnings$height*2.54

# Parts e-i and e-ii
cm.mod <- lm_robust(earnings ~ heightcm, data = earnings)
coefficients(cm.mod)
```

```
## (Intercept)    heightcm
##   -512.7336    278.6108
```

```
# Part e-iii
cm.mod$r.squared
```

```
## [1] 0.0108753
```

```
# Part e-iv
summary(cm.mod)
```

```
##
## Call:
## lm_robust(formula = earnings ~ heightcm, data = earnings)
##
## Standard error type:  HC2
##
## Coefficients:
##             Estimate Std. Error t value  Pr(>|t|) CI Lower CI Upper    DF
## (Intercept)   -512.7    3380.00 -0.1517 8.794e-01  -7137.9   6112.4 17868
## heightcm       278.6      19.84 14.0419 1.490e-44    239.7    317.5 17868
##
## Multiple R-squared:  0.01088 ,   Adjusted R-squared:  0.01082
## F-statistic: 197.2 on 1 and 17868 DF,  p-value: < 2.2e-16
```

Parts f and g

```
f.mod <- lm_robust(earnings ~ height, data = filter(earnings, sex == '0:female'))
m.mod <- lm_robust(earnings ~ height, data = filter(earnings, sex != '0:female'))
summary(f.mod)
```

```
##
## Call:
## lm_robust(formula = earnings ~ height, data = filter(earnings,
##     sex == "0:female"))
##
## Standard error type:  HC2
##
## Coefficients:
##             Estimate Std. Error t value  Pr(>|t|) CI Lower CI Upper   DF
## (Intercept)  12650.9     6299.9   2.008 4.466e-02    301.7  25000.0 9972
## height         511.2       97.6   5.238 1.655e-07    319.9    702.5 9972
##
## Multiple R-squared:  0.002672 ,  Adjusted R-squared:  0.002572
## F-statistic: 27.44 on 1 and 9972 DF,  p-value: 1.655e-07
```

```
summary(m.mod)
```

```
##
## Call:
## lm_robust(formula = earnings ~ height, data = filter(earnings,
##     sex != "0:female"))
##
## Standard error type:  HC2
##
## Coefficients:
##             Estimate Std. Error t value  Pr(>|t|) CI Lower CI Upper   DF
## (Intercept)   -43130    6926.25  -6.227 4.995e-10   -56708   -29553 7894
## height          1307      98.87  13.217 1.826e-39     1113     1501 7894
##
## Multiple R-squared:  0.02086 ,   Adjusted R-squared:  0.02074
## F-statistic: 174.7 on 1 and 7894 DF,  p-value: < 2.2e-16
```

If a randomly selected woman is 1 inch taller, we predict her earnings to be greater by $511.2. For men, by $1307. (Note that if we want to use the dollar sign as a dollar sign in R notebooks, we put a backslash before it. Otherwise, it thinks we want to write an equation.)

Part h

Height may be correlated with other factors that cause earnings. For example, height may be correlated with strength which in some occupations makes workers more productive. There are other potential factors correlated with height that have a causal relationship with earnings. This type of relationship is called endogeneity and will be of interest to us throughout the course.

Practice Problem 7: SW Empirical Exercise 5.3

```r
smoking <- read.dta13('birthweight_smoking.dta')
head(smoking)
```

```
##   nprevist alcohol tripre1 tripre2 tripre3 tripre0 birthweight smoker unmarried
## 1       12       0       1       0       0       0        4253      1         1
## 2        5       0       0       1       0       0        3459      0         0
## 3       12       0       1       0       0       0        2920      1         0
## 4       13       0       1       0       0       0        2600      0         0
## 5        9       0       1       0       0       0        3742      0         0
## 6       11       0       1       0       0       0        3420      0         0
##   educ age drinks
## 1   12  27      0
## 2   16  24      0
## 3   11  23      0
## 4   17  28      0
## 5   13  27      0
## 6   16  33      0
```

Part a

```r
# Part a-i
mean(smoking$birthweight)
```

```
## [1] 3382.934
```

```r
# Part a-ii and a-iii
# Method 1
filter(smoking, smoker == 1) %$%
  birthweight %>%
  mean
```

```
## [1] 3178.832
```

```r
filter(smoking, smoker == 0) %$%
  birthweight %>%
  mean
```

```
## [1] 3432.06
```

```r
# Method 2
t.test(birthweight ~ smoker, data = smoking)
```

```
##
##  Welch Two Sample t-test
##
## data:  birthweight by smoker
## t = 9.4414, df = 887.15, p-value < 2.2e-16
## alternative hypothesis: true difference in means between group 0 and group 1 is not equal to 0
## 95 percent confidence interval:
##  200.5882 305.8685
## sample estimates:
## mean in group 0 mean in group 1
##        3432.060        3178.832
```

Part b

We can use the same t-test function to answer part b-i.

```
weight.test <- t.test(birthweight ~ smoker, data = smoking)
# Differences
diff(weight.test$estimate)
```

```
## mean in group 1
##       -253.2284
```

```
# Standard error of difference
weight.test$stderr
```

```
## [1] 26.82106
```

```
# 95% confidence interval on differences
weight.test$conf.int
```

```
## [1] 200.5882 305.8685
## attr(,"conf.level")
## [1] 0.95
```

Part c

```
weight.model <- lm_robust(birthweight ~ smoker, data = smoking)
summary(weight.model)
```

```
##
## Call:
## lm_robust(formula = birthweight ~ smoker, data = smoking)
##
## Standard error type:  HC2
##
## Coefficients:
##             Estimate Std. Error t value  Pr(>|t|) CI Lower CI Upper   DF
## (Intercept)   3432.1      11.89 288.675 0.000e+00   3408.7   3455.4 2998
## smoker        -253.2      26.82  -9.441 7.148e-21   -305.8   -200.6 2998
##
## Multiple R-squared:  0.0286 ,    Adjusted R-squared:  0.02828
## F-statistic: 89.14 on 1 and 2998 DF,  p-value: < 2.2e-16
```

Part c-i

The intercept is the average birthweight for non-smokers

The slope is the difference between average birthweights for smokers and non-smokers

Part c-ii

The standard errors are the same (if we use the robust standard errors)

Part c-iii

```
confint(weight.model)
```

```
##                 2.5 %    97.5 %
## (Intercept) 3408.7485 3455.3714
## smoker      -305.8179 -200.6388
```