STAT350 Tutorial 1

Dylan Maciel

11/09/2020

For this course we will be using R to implement the concepts learned in class. This tutorial covers downloading and installing R and RStudio for Mac and Windows. Then we'll go over some basic R commands and do an example simple linear regression on a data set.

Installing R and RStudio

R is a free open-source software used for statistical computing; the current version is 4.0.2. You can code directly in the program, but using an IDE like RStudio, which is also free, allows you to write more than one line at a time and integrates many other useful tools that make using R more efficient.

The order in which you install these programs is important. For Mac users you'll have to download and install XQuartz first. Then, for both Windows and Mac install R before you install RStudio. For RStudio you'll want to download the free version of RStudio Desktop. I recommend keeping all the defaults when installing these programs. If done in the proper order, when you first open RStudio it will detect R on your computer and start the R console itself.

For a more in-depth introduction to R and Rstudio I recommend reading through sections 1 to 4 of A (very) short introduction to R by Torfs and Brauer. It also goes through installation, but will also introduce you to some basic R functions and uses of RStudio.

If you have any troubles with these installations, I will be holding online office hours over zoom next week on Tuesday at 5pm and Wednesday at 10 am. I can also answer any other R questions you may have.

An Initial Example

In what follows I will briefly go over reading in data and fitting a simple linear regression model. To get the most out of this example I would open a new R script in RStudio and go through the code below. Copy the code, make sure you can get it working on your computer and try to understand what each line is doing.

Data Entry

Typically, we use data from external files in R. The function to read in data from an ".csv" or excel file is `read.csv`. This file can either be located on you local disk or we can read a file directly from the internet as seen below .

```
forbes_data <- read.csv(file = "http://users.stat.umn.edu/~sandy/alr4ed/data/Forbes.csv",
                        header = TRUE,
                        sep = ',',
                        row.names = 1)
```

This function has many input. The first input `file` tells R the name of file containing the dataset. If this file was on your computer the way to ensure that R can find the file is to put the whole path to the file within the quotes.

We set the `header` input to `TRUE` if the first row of the dataset contains variable names, otherwise it should be set to `FALSE`. If this is not explicitly set it will default to `TRUE`.

The `sep` input tells the function how the columns of data are separated within the file. It's default value is `','`.

The `row.names` input names can be used to specify the column number that contains the row names. Many data sets do not have row names, for those this input does not need to be specified.

Here I have saved the data to the object named `forbes_data`. Now anytime I want to work with this specific dataset I just need to call it by that name. The name you choose for the dataset is important and should be informative.

Looking at the data.

We can look at the first few rows of the data using the `head()` function.

```
head(forbes_data)
```

```
##      bp  pres  lpres
## 1 194.5 20.79 131.79
## 2 194.3 20.79 131.79
## 3 197.9 22.40 135.02
## 4 198.4 22.67 135.55
## 5 199.4 23.15 136.46
## 6 199.9 23.35 136.83
```

So, we see that the dataset contains three variable; `bp`, `pres`, and `lpres`. Before fitting a model to the data it is good practice to do some initial data analysis. This is done to ensure the data looks the way we expect it to look. If the data has mistakes or missing values they could negatively impact the performance of our model in the end. Datasets used in tutorials will be clean and ready for analysis, but you should keep this in mind in your future work.

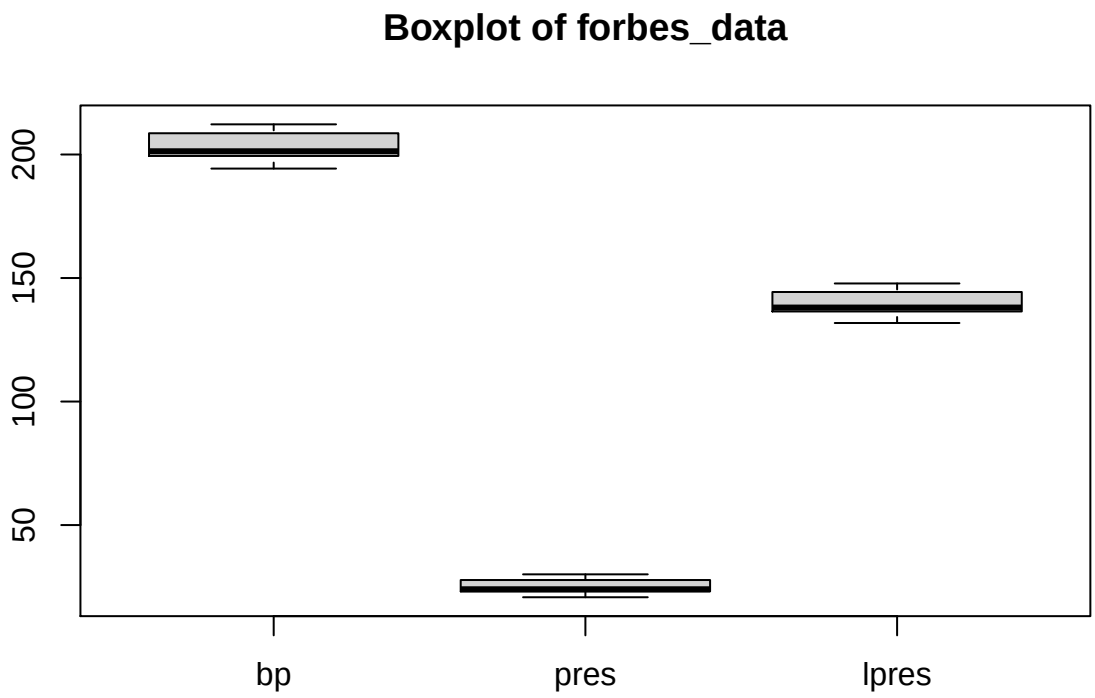
We can get univariate numerical summaries of the data using the `summary()` function.

```
summary(forbes_data)
```

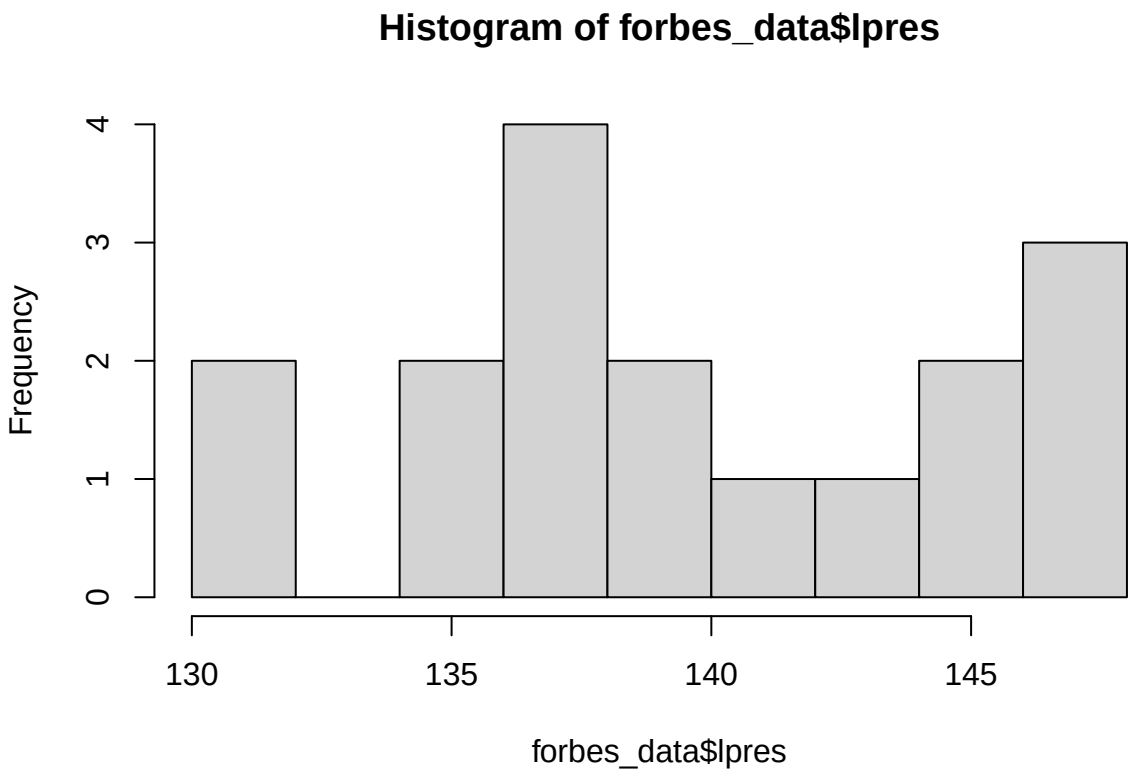
```
##        bp             pres           lpres
##  Min.   :194.3   Min.   :20.79   Min.   :131.8
##  1st Qu.:199.4   1st Qu.:23.15   1st Qu.:136.5
##  Median :201.3   Median :24.01   Median :138.0
##  Mean   :203.0   Mean   :25.06   Mean   :139.6
##  3rd Qu.:208.6   3rd Qu.:27.76   3rd Qu.:144.3
##  Max.   :212.2   Max.   :30.06   Max.   :147.8
```

We can also make graphical summaries. Below, I use the `boxplot()` function to create a boxplot for each variable.

```
boxplot(forbes_data,
        main = 'Boxplot of forbes_data')
```



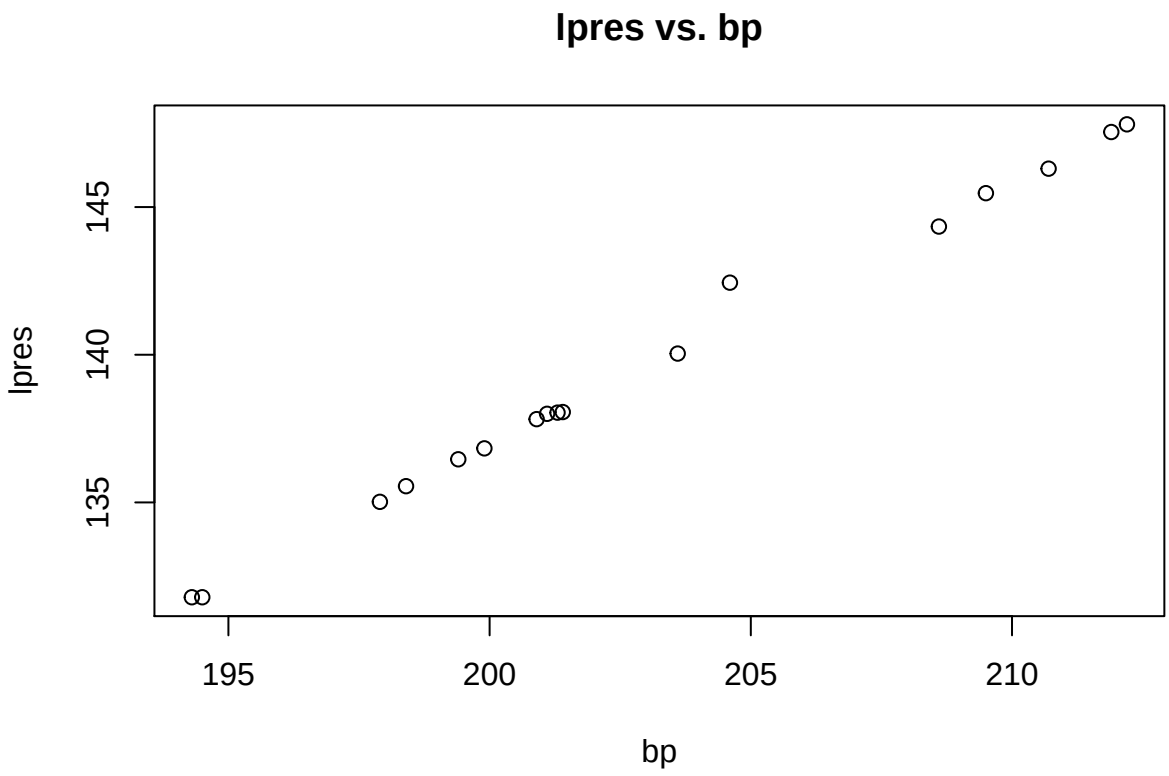

```
hist(forbes_data$lpres)
```



Simple Linear Regression

For our example `lpres` will be the response and `bp` will be the predictor. First we'll create a basic scatter plot of the data using the `plot()` function.

```
plot(formula = lpres ~ bp,
     data = forbes_data,
     main = "lpres vs. bp",
     xlab = "bp",
     ylab = "lpres")
```



The first input, `formula`, tells R how to arrange the data on the plot. The response goes to the left of the `~` and the predictor to the right. The next argument tells R where the data is located. This is followed by arguments that let us set the main title, x-axis label, and y-axis label, respectively. There are other arguments to this function that let us further customize how we want the plot to look.

Looking at the plot, we see that a linear regression model would be suitable for this data and can proceed with creating the model.

In R we use the function `lm()` to fit linear regression models. The first two arguments for this function follow the same format that we used in the `plot()` function above.

```
my_mdl <- lm(formula = lpres ~ bp,
             data = forbes_data)
```

Here I have saved the resulting object from using this command to the name `my_mdl`. If we want brief summary of the model we can call it by name.

```
my_mdl
```

```
##
```

```
## Call:
## lm(formula = lpres ~ bp, data = forbes_data)
##
## Coefficients:
## (Intercept)           bp
##    -42.1378       0.8955
```

With this we see how the linear model was defined, but more importantly we get the estimated coefficients of the model. Here the estimated intercept is -42.138 and the estimated slope is 0.895. If we want to extract the coefficient values from the model object we use `$` followed by the name.

```
my_mdl$coefficients
```

```
## (Intercept)          bp
## -42.1377793   0.8954937
```

To get a single value, we then add the index within square brackets to the code above. If we want the intercept, it's the first element and therefore indexed by 1.

```
my_mdl$coefficients[1]
```

```
## (Intercept)
##   -42.13778
```

We can get a more elaborate summary of the model we can use the `summary()` function on the model object.

```
summary(my_mdl)
```

```
##
## Call:
## lm(formula = lpres ~ bp, data = forbes_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.32220 -0.14473 -0.06664  0.02184  1.35978
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
```

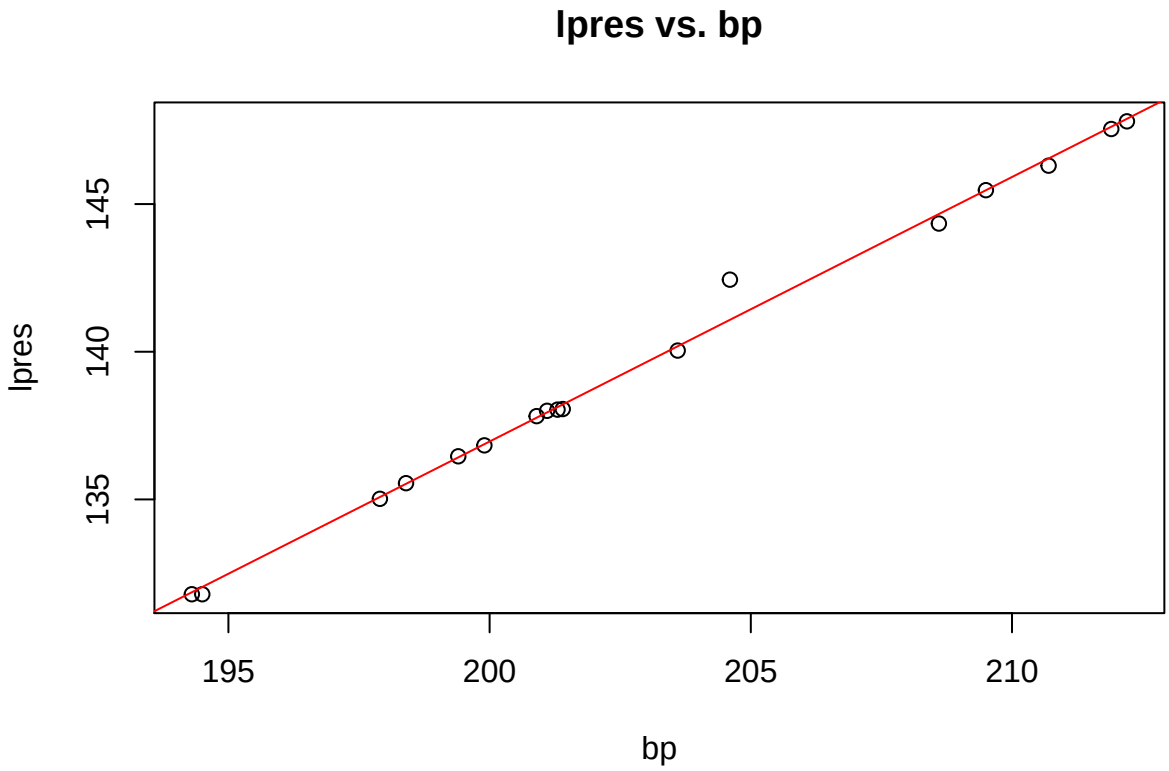
```
## (Intercept) -42.13778    3.34020  -12.62 2.18e-09 ***
## bp            0.89549    0.01645   54.43  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.379 on 15 degrees of freedom
## Multiple R-squared:  0.995,  Adjusted R-squared:  0.9946
## F-statistic:  2963 on 1 and 15 DF,  p-value: < 2.2e-16
```

The added information here will be useful in the coming weeks when we cover confidence intervals, hypothesis tests, and assessing the model.

Now that the model is fit we can add the regression line to the scatter plot of the data. This is done with the function `abline()` whose input `reg` is the model object defined above. Alternatively, we could also manually set the regression line values using the input `a` for the intercept and `b` for the slope. This function places a straight line over top of a scatter plot and should therefore be called after the `plot()` function. Here I've also included the argument `col` to specify the colour of the line.

```
plot(formula = lpres ~ bp,
     data = forbes_data,
     main = 'lpres vs. bp',
     xlab = 'bp',
     ylab = 'lpres')

abline(reg = my_mdl,
       col = 'red')
```


lpres vs. bp
lpres
135
140
145
195
200
205
210
bp