Introduction to R Notebooks

Matthew Alampay Davis

September 28, 2021

This is an R Markdown Notebook, which will be the format for your problem set and exam submissions if you choose to use R. When you execute code within the notebook, the results appear beneath the code. We'll get to the code later. For now, we are just using this file as a notebook. Type here in the way you would on a Word document, for example.

Let's start by creating headings and subheadings. These will be useful for problem set submissions so you can easily distinguish questions, sub-questions, and sub-sub-questions:

Part 1: Introduction to headings

The pound sign (#) here created this subheading. You can't see it if you're reading this in RStudio, but once you render this notebook (i.e. tell R to convert the notebook into a pdf file), it will appear as bolder and larger than standard text. You can use multiple pound signs to create a subheading (##) or a sub-subheading (###) and so on:

Part 1a: Introduction to subheadings

Like so!

To see this difference, click on the *Knit* button at the top and select *Knit to pdf*. Knitting is just what R calls the process of converting a notebook to a properly formatted document. This is the format we'll want for our problem set submissions.

Using asterisks like this makes that text appear as italicized. We can also include math symbols and expressions by putting them in between dollar signs like this: $a^2 = b^2 + c^2$ to write an equation in line or using double dollar signs to write a full equation like so:

$$y_i = \beta_0 + \beta_1 x_i + e_i$$

To be able to write in R notebooks, you should run the following expression in the console panel of RStudio:

tinytex::install_tinytex()

That is to say, just copy it into the console panel of RStudio (you'll notice RStudio's interface is divided into four panels) and press enter to begin a download. You only ever need to do this once.

To get really good at writing mathematical expressions, you'll want to learn a symbolic typesetting grammar called TeX or LaTeX, but don't worry about it for this course. For our purposes, knowing that an underscore _ produces subscripts and a carat sign ^ produces superscripts will probably be more than enough.

Part 2: Code

To start coding in an R Notebook, just write a "chunk" like below:

```
2+2
```

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

Anything in this chunk is a command. Here, we've commanded R to compute 2+2 and if you run this command (either by pressing *Cmd+Enter* after you've clicked on the line of code in the chunk or by clicking on the *Run* button at the top right of this panel, you can see the answer that R outputs. When we convert this notebook into a pdf, we'll be able to see both the code and the output in-line.

Let's do something a tiny bit more complicated. "cars" is a practice dataset that's built into R just as an example of a dataset. It consists of two variables, "speed" and "dist". We can see a quick preview of this dataset by using the *head* command, which shows us the first six rows of a dataset:

```
head(cars)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

Alternatively, we might want to just see the first three observations:

```
head(cars, 3)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
```

or the last three observations:

```
tail(cars, 3)
```

```
##    speed dist
## 48    24   93
## 49    24  120
## 50    25   85
```

Or we might want to get summary statistics about each variable:

```
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

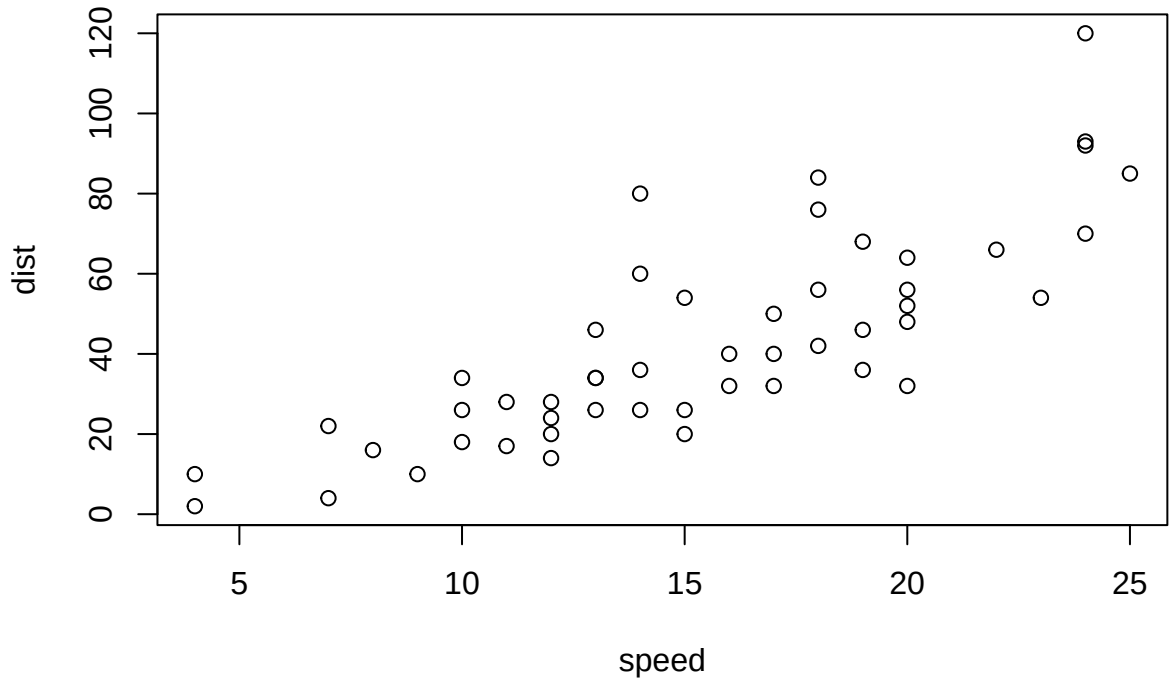
In these examples, we are using different functions (head, tail, summary). These functions take "arguments" in parentheses that are separated by commas. The first argument was the dataset we wanted to use (cars) and the second is a number n for the number of observations we wanted to see. Different functions have different arguments. If you ever need to know what arguments a given function has, just type a question mark followed by the function into the console:

```
?head
```

Part 3: 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 (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.

Part 4: 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 a lot. To download a package, for example the the "ggplot2" package, run the following commmand:

install.packages('ggplot2')

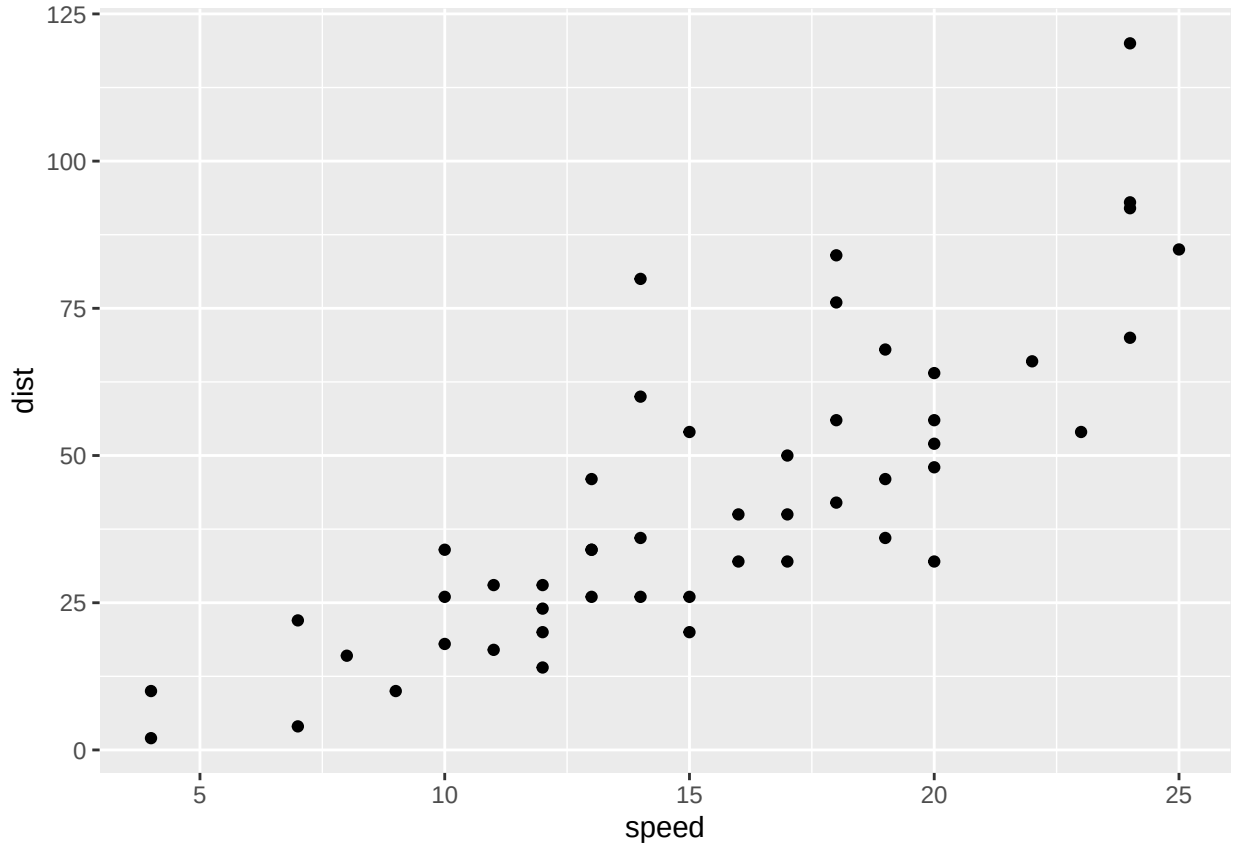
When installing packages, like we did before, run this command in the console panel rather than in the notebook panel. You'll only ever need to do this once per package.

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.

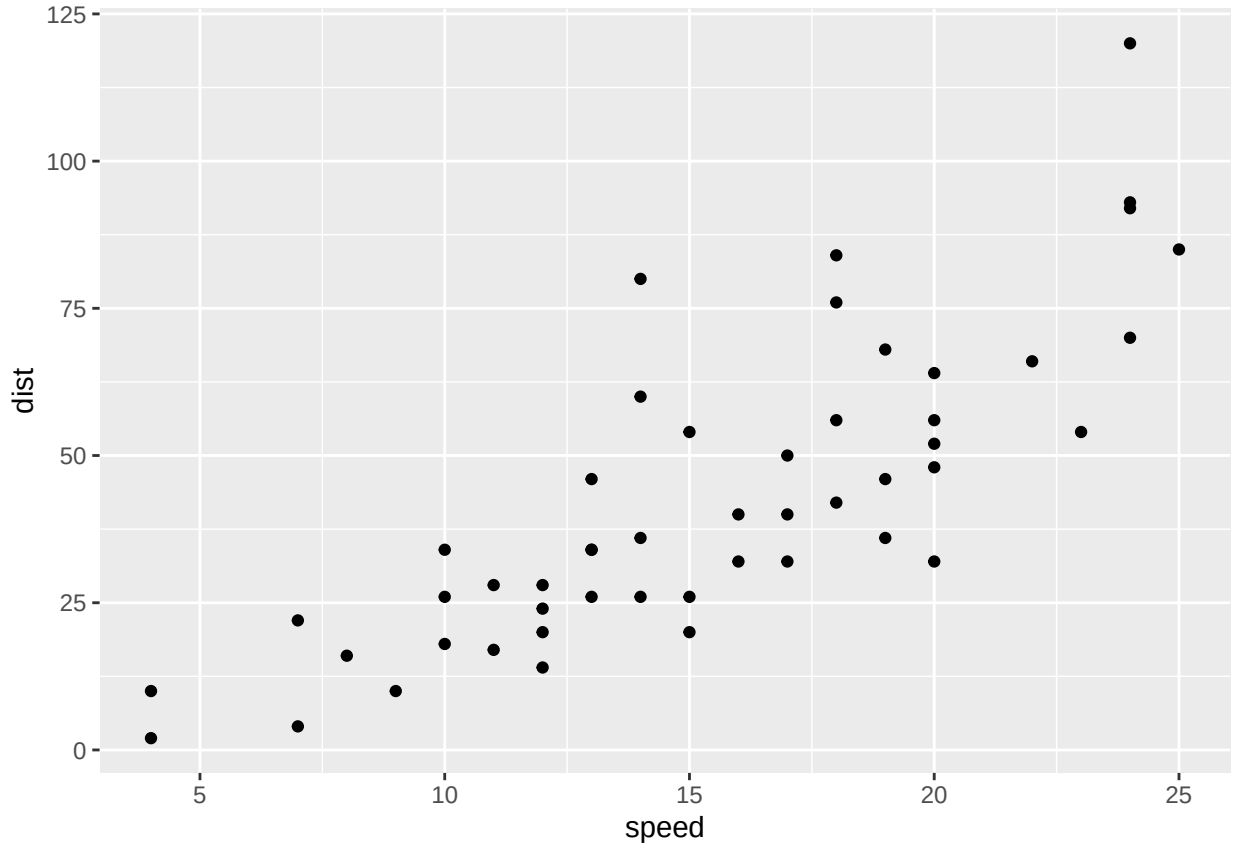
We end the line with a "+" to denote we want to add an additional plot element. Here, we wanted a scatterplot so we use the geom_point() function with no argument (nothing in its parentheses). This may seem much more complicated than the standard plot function, but once you get a hang of this sort of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

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:

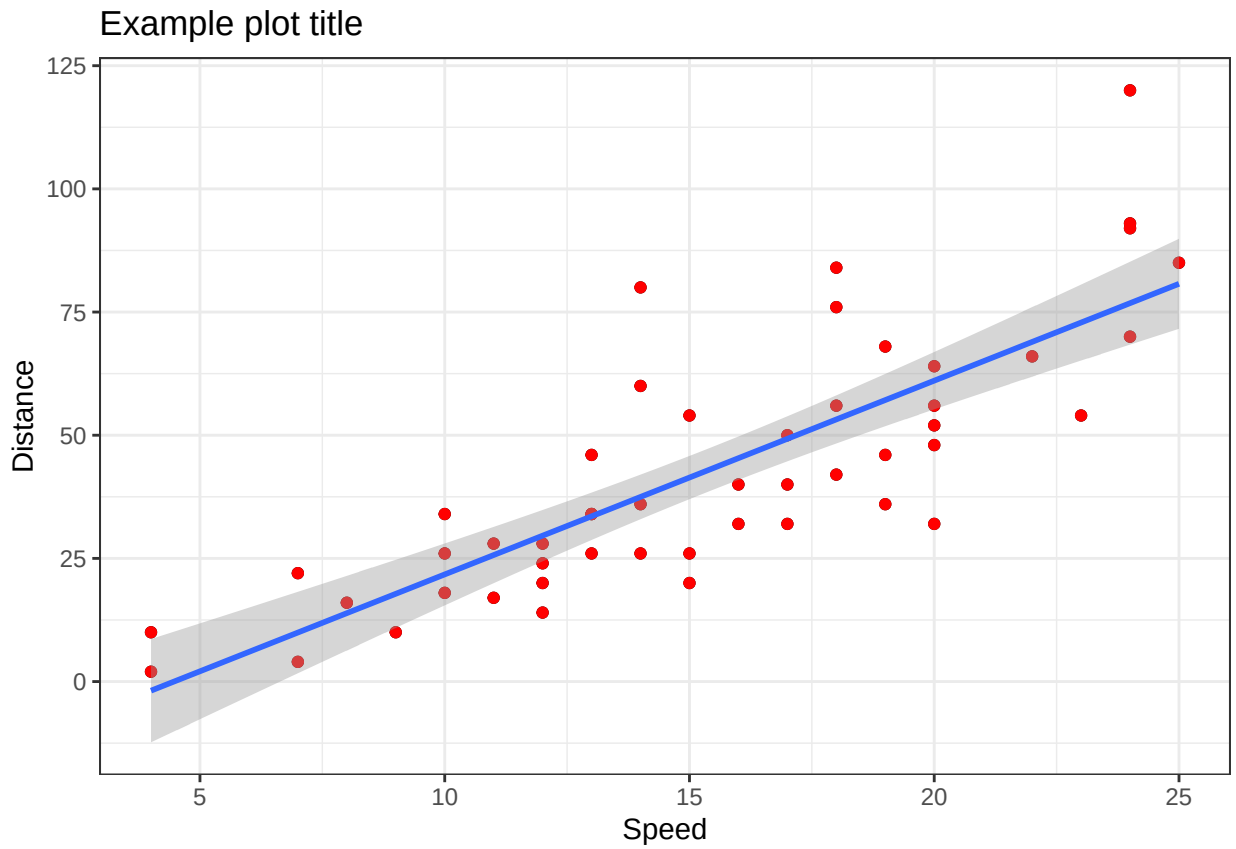
```
print(test.plot)
```



And we can make some new modifications, thanks to the grammar of ggplot2:

```
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)
```

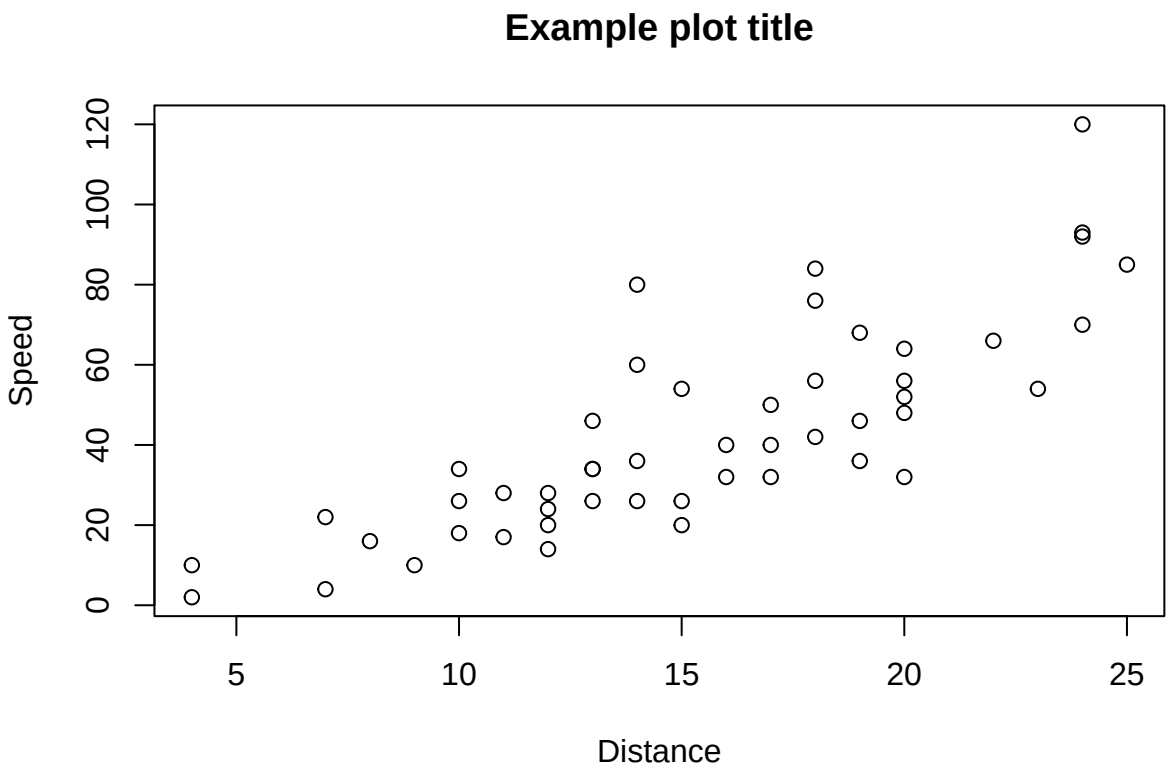
```
## `geom_smooth()` using formula 'y ~ x'
```



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:

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



Again, ggplot may seem much more complicated than the simple plot command, but learning ggplot grammar will pay off later on when we do other types of graphs or want to modify them because its modifiability is intuitive.

Part 5: 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 a package that can. Again, 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 load the file. Let's call the dataset "animals" by using the assignment notation from before:

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

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

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

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

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

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

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

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

```
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
```

```
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", 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:

```
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 6: Regressions

The most important thing we'll do in this course is run regression models of various types so I'll end this introduction with quick intro. 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 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 predicted/fitted values
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
```