Introduction to R - 1 day with Tidyverse

Lucy Liu

Introduction to course

1. Introduce self and helpers
2. Who has used R before? Who has used a different programming language?
3. Make sure everyone has R and RStudio
4. Go over outline for the day - on website. Note that cannot teach R in just one day. The idea is for everyone to be able to get started and familiar with R. And have the confidence to look things up yourself.

5. All course notes are available online
6. Google docs - there will be challenges to help you understand concepts. They are all in the google docs. If you don't see this outline, click on view > show document outline.

The 4 panels

Introduce the 4 windows: Script editor, Console, Environment/History, Files/Plots/Help/packages.

The first panel is the console. You can run commands here. **Whiteboard** Commands/code = instructions for the computer. Type in some maths and hit enter - runs the code. You may notice one thing though. You can't click back up here and edit what you wrote, in case you mistyped. You also can't save any of your commands - so you can look at them later.

This is where the top 'script editor' panel comes in. You can think about scripts as 'word' documents. A script is just a document with text - it's just that the text is code (instructions to the computer). You can make a new script %show% and this will be shown in this window. It is a good way to organise your work because you can save it - it will be a .r file - and refer to it or edit it later. If you haven't opened a script, I'll get you to open a new script and save it.

You can type some code here, then use Ctl/Cmd + Enter to make it run. E.g. 1+1.

The code and the result pops up down here. The difference is that you can edit your command up here and run it again.

In this panel here you have your environment - you can see variables that you have created. For now just think of this panel has showing the data that you have input into R. In the next tab you have your History - you can look at the history of the code you have run.

Down here you have a few tabs. One of the most important is plots. This is where your graphs will pop out when you make them.

Another one is files. This is just like the directory of files in your computer. You can go up to see which folder you are in and click into different files - just like the file directory or file explorer on your computer.

There is a thing called your 'working directory' - which is an odd concept at first. This is where you are in your computer. When you start R, it starts it in a partcular location on your computer (e.g. under Lucy and my R folder). A working directory is where R will automatically look for files that you want to import (like your data) and it's where it will automatically save your graphs that you make.. We'll talk about this later and hopefully it will make more sense.

It's a good idea set your working directory to somewhere that makes sense. So let's do that. First make a folder for this workshop, somewhere on your computer - whatever makes sense for you - your desktop or home folder etc.

Now lets make this folder your working directory. One way to do this to get the path to this folder (show in file explorer - here I am in lucy/documents/R. . . .). Now use the command:

```
#setwd("lucy/documents/R")
```

Your first challenge is to set your wd.

CHALLENGE: Make new folder and setwd to this folder

Useful tips

Cancelling commands

Once you have ran your command, look at the output here. If you've done it right, it gives you an answer. And note the like arrow, greater than sign here. This means that it is ready to take your next command.

If you ran a command and it gives you a plus - what do you think has happened here? You can see here that if you don't finish a command it will expect you to finish it - the plus here means that you haven't finished your command and it's waiting for you to finish your command.

You can either finish your command or press esc - which stops your command and returns the > - which means you can start writing a new command.

Comments

You can also leave yourself comments using #. Anything after a # is ignored by R. For example running the command below - R ignores my comment.

```
256 * 255 #doing some multiplication
```

```
## [1] 65280
```

Variables

Whiteboard

There are two more concepts I will introduce before we get to the fun stuff and import some data.

The first is a variable. This is a little bit like in algebra when we used letters like 'x' to represent numbers/values. A variable is just a thing that we have given a name to so we can refer to it later. For now, the thing is usually data like a number or a whole table of data like a excel spreadsheet.

To make a variable - you just assign the name to it using <- and you tell R what data you want to save under that name.

(You might have also seen people use '='. It is almost the same but it's better practice in R to use <-)

```
mydata <- 12
mydata
```

```
## [1] 12
```

```r
#what will happen if I do this?

mydata <- 12 + 3
#what does this give out
mydata
```

```
## [1] 15
```

Also notice that the variables come up here in your environment. This means its there for you to use.

Variable names

R is a bit particular about the names you can give variables. The name can't start with a number or a punctuation mark and they can't contain spaces at all. What do you think R will do if I do this:?

```r
2d <- 4
my data <- 6
```

Errors

It will return an error.

An error is when R cannot do what you asked it. It prints something out in red and it tries to tell you why it couldn't do what you asked it. Sometimes the message can be cryptic but you'll also get better at decoding these error messages.

Note that R also gives a warnings. A warning is when R could do what you asked it to - but it wants you to be careful of something. Often you can ignore warnings.

whiteboard error vs warnings. Error: R couldn't do what you asked it to do and you need to figure out why. Warning: R could do what you asked it to but be wary of something. Can be ignored often.

Functions

whiteboard function = does something. name(input) The second concept to introduce are functions. A fuction is just something that does something.

We have already used functions. setwd() was a function. It sets the working directory to be what you tell it.

sum() is another function. What do you think it does?

```r
sum(3,4,65)
```

```
## [1] 72
```

The names are pretty self explanatory.

A function looks like this name(input). We call these arguments.

Importing data

Let's finally get to the fun stuff and import some data in.

We will use the function read.csv(). Now what do you think we will need to put within the brackets?

```
read.csv("https://docs.google.com/uc?id=1ceiVo36_wxBDsg4irC1FZg-t7LQKtyjI&export=download")
```

```
##               Address Rooms   Price
## 1     25 Bloomburg St     2 1035000
## 2 18/659 Victoria St     3  230000
## 3        5 Charles St     3 1465000
## 4   40 Federation La     3  850000
## 5         55a Park St     4 missing
```

I see my data has popped out down here. But it hasn't shown up here in the environment. Why not??

```
house <- read.csv(
  "https://docs.google.com/uc?id=1ceiVo36_wxBDsg4irC1FZg-t7LQKtyjI&export=download")
```

You can see it has popped up here. This means that it has 'saved as a variable' and I can use it again. You can click on it and have a look at it here.

Your next challenge is to read the data in.

Now that you have read it in, you can have a look at it by clicking. You can also use the $ to look at specific columns.

I just want to mention that you can also ask R to import a file from your computer. You just have to put the file name here, instead of a URL.

read.csv() arguments

There is a lot more to the read.csv function. There are many inputs to this function, which change the way that data is imported in.

To learn more, we use the help function. If you want details about how to use any function, type in ? and then the function name (?read.csv) and you open the help file. It shows up in this panel and it is really useful. It is also really confusing when you start out. Don't worry you will get better at interpreting what this means.

Let's have a look at the read.csv help file. The title is data input - this sounds good as read.csv inputs data into R. Under usage its got read.csv but it's also got a number of other functions - read.table, read.csv2. These are here because they are all closely related - as you can imagine they all import data into R. Let's ignore the others and just look at read.csv. In the brackets its got all the possible inputs (arguments) it can take. These arguments let you specify how the function read.csv will do its job. The first thing in the bracketss is file. This means that this is the first input (remember from functions on the whiteboard) it expects. It makes sense because R needs to know what data you want it to import.

If we scroll down, there is more detail on file.

The next input (argument) is header. What does header mean? A logical value - it can either be TRUE or FALSE. Can anyone interpret this?

Now its your turn.

Challenge: If your data file had each data point separated by ";" (show on google image), what read.csv argument would you have to use to read the data in?

Take 10min break. Tea and coffee outside. The water cooler does give out hot water.

Data structures

We now need to learn about how our data is stored.

Data structures are containers for data - it's a way for R to store data. Unlike in excel, R has more rigid data containers. The 2 most often used are vectors and dataframes.

A dataframe holds 2D data (**whiteboard**) just like the data in your excel spreadsheet. R keeps track of the order of the data such that if you wanted the data in row 2, column 5, R can easily find it.

Vector is 1D and it is just like a row or column in excel. It is also ordered. If you wanted the 5th element, R can easily find it.

You can think of a dataframe as being composed of vectors.

You can make your own vector easily with the function:

```
vector <- c(1,2,3)
```

Now let's add another column to our data. We want to add a column that tells us how many bathrooms each of these houses has.

We can do this by first making a vector then adding it to our data frame. **whiteboard**

How long does the vector have to be?

Challenge: make a vector called bathroom, which has 5 numbers in it

---

Now you can add it to the dataframe house using the function cbind:

```
bathroom <- c(2,3,1,2,1)
house <- cbind(house,bathroom)
```

Data types

The other important concept to learn is data types.

There are 5 basic data types in R and every bit of data in R is labelled as a certain data type. R will look at it and give it a label.

Whiteboard 1. Logical TRUE or FALSE. What kind of data could be stored as TRUE or FALSE?
Next 3 are all numbers 2. Integer. Number without a decimal point. 3. Double. Number with a decimal point. 4. Complex. Remember i - the imaginary number from maths? Numbers with i. 5. Character. Sentences, words, letters. Always with quotes around it.

In a dataframe, R expects that all the values in one column are going to be the same data type (numbers/words). In a vector, R expects all the data to be of the same data type.

Note that it NEVER changes your data. If it is 45 it is always 45. 45 could be labelled as a number or it could force R to labeled it as a character, "45", but it is always 45. R just puts a label on it, so it knows what kind of data it is.

It's important for R to know what kind of data it is. This tells R what it can and can't do with it. For example it knows it can't log a character. What else could it NOT do on a character.

What could it do on a character data type and not on numbers? Search for a phrase e.g. find "street".

It also knows that adding a data labelled as a number to one labelled as character does not make sense. It also knows it can't take the log of data labelled as character. Can you think of something you can do with a character but not with numbers?

You can find out how R has labeled your data using class().

Let's look at a specific column. Can you remember how?

What do you think will be?

```
class(house$Rooms)
```

```
## [1] "integer"
```

Challenge: use class() to find out what kind of datatype the column 'Address' is

---

It's not what we expect. What is a factor?

A factor is another data type (not one of the basic ones) - it is used to store categorical data like red, blue, yellow or small, med, big. It is useful when the data is actually categorical because you can easily group the data into their categories (e.g. yellow, red, blue data) and perform functions - like taking the mean.

Each category is a level. And you can see this using str(). You can see all the possible categories (levels).

```
str(house$Address)
```

```
##  Factor w/ 5 levels "18/659 Victoria St",..: 2 1 5 3 4
```

```
levels(house$Address)
```

```
## [1] "18/659 Victoria St" "25 Bloomburg St"    "40 Federation La"
## [4] "55a Park St"        "5 Charles St"
```

But why has it done this?? Well when importing data into R, it labels all character data type (words,sentences) as a factor.

This is often annoying because in many cases like this one, characters are not factors. It can also be annoying to deal with data that is stored as factors. For example if I want to add another row:

```
myhouse <- c("1 Park st", 3, 500000)
rbind(house, myhouse)
```

```
## Warning in `[<-.factor`(`*tmp*`, ri, value = "1 Park st"): invalid factor
## level, NA generated

## Warning in `[<-.factor`(`*tmp*`, ri, value = "1 Park st"): invalid factor
## level, NA generated

##              Address Rooms   Price
## 1    25 Bloomburg St     2 1035000
## 2 18/659 Victoria St     3  230000
## 3       5 Charles St     3 1465000
## 4   40 Federation La     3  850000
## 5        55a Park St     4 missing
## 6               <NA>     3    <NA>
```

It shows a warning. Can you remember what a warn is?

Why have I gotten this? When R created this factor - it assumed that the all the possible categories were given to it. So it thinks that these addresses are factors (which they are not - they are not categorical data) and it thinks these 5 addresses are the only possible addresses. So it thinks values in this column can only be 25 bloomburg st etc… When we want to add another category it won't do it. This is annoying. The easiest way to solve this problem is to tell R not to make this column a factor when it reads in the data. We can always turn it into a factor later.

There is a setting in read.csv() that will makes R NOT label character columns as a factor.

Challenge: Use ?read.csv to figure out how to keep text columns as character vectors instead of factors. Then add another house (row) to our data using rbind(). Use the house details below: "1 Park st", 3, 500000

Any questions?

NA values

The last thing we will talk about this morning is missing values. In R missing values are called NAs. You can see that in our data, we have a missing value here and when they entered the data they have just put the word missing. This is a NA value.

When you read in the data, there is another input in read.csv(), that lets you specify what should be NA (999, xx). Remember that in the help file, the … means that read.csv has all the same possible inputs/arguments as read.table - it just isn't written out for brevity.

Challenge. use ?read.csv to help you figure out what argument you need to set for R to label the word "missing" as NA.

```
house <- read.csv(
  "https://docs.google.com/uc?id=1ceiVo36_wxBDsg4irC1FZg-t7LQKtyjI&export=download", stringsAsFactors =
```

Now that it is correctly coded as an NA, let's try to get the mean of prices. There is a handy function mean for that.

How do we get just one column?

```
mean(house$Price)
```

```
## [1] NA
```

Why has this happened? R says that it does not know what the mean is because the mean of 100000, ... and a number i dont know, is I don't know! Because I don't know this number (which could thus be anything), R can't tell you what the mean is.

You can tell the function mean to ignore NA values though.

Challenge: use ?mean to work out how to get the function mean to ignore NA values. Calculate the mean of the column price

---

What we covered: 1. Variable 2. Function and help 3. Error vs warning 4. Data structure (df and vector) 5. Data type (5 basic and factor) 6. NA

Questions?

Challenge: install dplyr and ggplot

```
#install.packages("dplyr")
#install.packages("ggplot2")

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
```

```
library(ggplot2)
```

LUNCH BREAK

New data

Let's take our training wheels off and use the whole set of data. We want to read it in.

Challenge: Read in the data from the new URL. Remember we don't want strings to be labelled as factors

---

```
house <- read.csv(
  "https://docs.google.com/uc?id=1wdSyZZ3U4tqMfgCKin7ogpTKaQ9gAc4t&export=download",
stringsAsFactors = FALSE)
```

The NA's have automatically been recorded. Why is that? If a cell/data point (in your original data) that is empty, it will automatically be recorded as NA.

Now I can see that there are lots of NA values. I'm going to remove them all with a function called na.omit. What is going to happen here?

```
house <- na.omit(house)
```

Let's have a look at the new data. The explanation is at the end of the google doc.

DPLYR

We are going to start to do some interesting things with our data now. We will use a package called dplyr. It's written by the chief scientist of RStudio - Hadley Wickham. It's very useful and intuitive for manipulating our data.

This package uses the %>% 'pipe' notation. You can think of it as send your data through a pipe. You start with your data (we've have called it house) and we pipe it to our next function where we might pick out some columns, then we might change the data in some way. The idea is that the data gets these functions performed on them sequentially.

Select

Let's start with the function select. **whiteboard**

You use it like this. What do you think this will do? It's made a new data frame.

```
house %>%
  select(Address, Rooms)
```

Challenge: Use select() to make a new data frame with only the columns Address and Landsize. Assign this new dataframe as a variable called 'selectHouse'.

Filter

Filter lets you pick which rows you want. (select: columns, filter: rows)

You use it like this: What do you think is going to happen here?

```
house %>%
  filter(Rooms == 2)
```

Why do we use == and not =? Remember how I said at the start that = pretty much does the same thing as <- So what = does is that it assigns a something to a variable. This data is called "house".

What == does is that it compares things. Here it looks through the rows and checks if Rooms is equal to 2, if it is, that row is kept and if it isn't we don't keep that row.

Challenge: Make a new dataframe with just the columns Postcode and Distance and just the rows where the Distance is 2.5.

```
house %>%
  select(Postcode, Distance) %>%
  filter(Distance == 2.5)
```

Challenge: Make a new dataframe with just the columns Rooms and Address and just the rows where the Type is 'h'

What's happened here? Remember when I said R does this in sequence. If we have selected just the columns Rooms and Address and then there is no column called Type - thus we can't pick just the rows where Type is 'h'. We have to pick the rows first and then select.

```
house %>%
  filter(Type == 'h') %>%
  select(Rooms)
```

What if we wanted the rows where Type is either 'h' or 't'? We can't do this:

```r
house %>%
  filter(Type == 'h' or 't')
```

R doesn't understand human language. We have to tell it specifically in this way:

```r
house %>%
  filter(Type %in% c('h','t'))
```

What this means is the Type has to be one of the things %in% here.

Challenge: Make a new dataframe with the columns Landsize and Postcode and the rows where the Suburb is "Carlton North" or "Parkville".

---

group_by and summarise

The next functions we'll learn about are group_by and summarise (write on whiteboard). These are great when you use them together. What will this do?

```r
house %>%
  group_by(Suburb) %>%
  summarise(mean = mean(Price))
```

Group by groups the rows so all the Abbotsford ones are together and all the Carlton ones are together. Summarise lets you summarise one group (e.g. the Abborstford group). Here we have worked out the mean Price.

How many rows does this new dataframe have? It only has 300 rows because there are 300 Suburbs altogether. Ther is now only 1 number for each Suburb - the mean price. And the name of this column we decided here.

Let's add another column. I want to know how many houses are in each suburb as well. What do you think I should do?

```r
house %>%
  group_by(Suburb) %>%
  summarise(mean = mean(Price), number = n())
```

Challenge: Find the mean price for the houses in each CouncilArea and the total number of houses in each CouncilArea Find the mean Landsize for the houses in each Suburb

---

Mutate

Mutate creates a new column.

What do you think will happen here?

```r
house %>%
  select(Address) %>%
  mutate(newcol = 'hello')
```

So it creates a new column and I can put anything I want in it. This isn't very useful. We have YearBuilt here. What if we another column with the age of the house - 2018-yearbuilt.

What do you think we would do?

```r
house %>%
  mutate(age = 2018 - YearBuilt) %>%
  select(age, YearBuilt)
```

Challenge: Create another column with the mean price in millions (e.g. 1 instead of 1000000)

**

Take a 10min break after this

ggplot

We are finally on to ggplot! It is one of the most popular packages in R. It was written by the same person that wrote dplyr and these 2 packages work well together as we will see later.

Whiteboard This is how to give instructions to R to tell it to make plots.

Let's make a scatter plot of Landsize vs yearbuilt. What do I start with? What should I put here?

So it doesn't look great. I don't really want to focus on this part of the graph. I can do that. Any further instructions are added on. Remember, like in dplyr, R looks at these instructions in sequence. Remember to give your instructions in the right order.

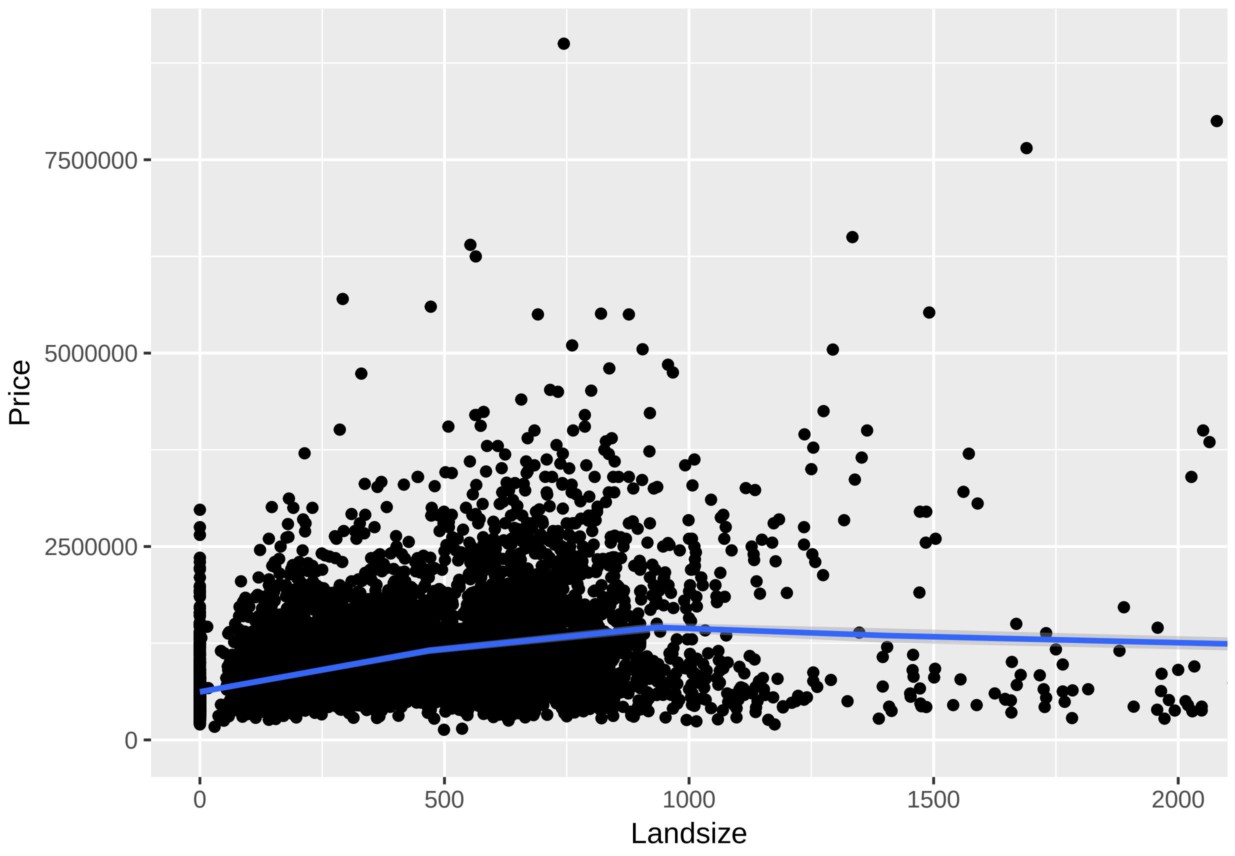
```r
ggplot(house, aes(x = YearBuilt, y = Landsize)) +
  geom_point() +
  coord_cartesian(xlim = c(1800,2050), ylim = c(0,10000))
```

CHALLENGE: Make a scatter plot of Landsize vs Price Change the coordinates of the y and x axis until you can visualise the data better.

Challenge: Add a line to your scatter plot of Landsize vs YearBuilt using geom_smooth()

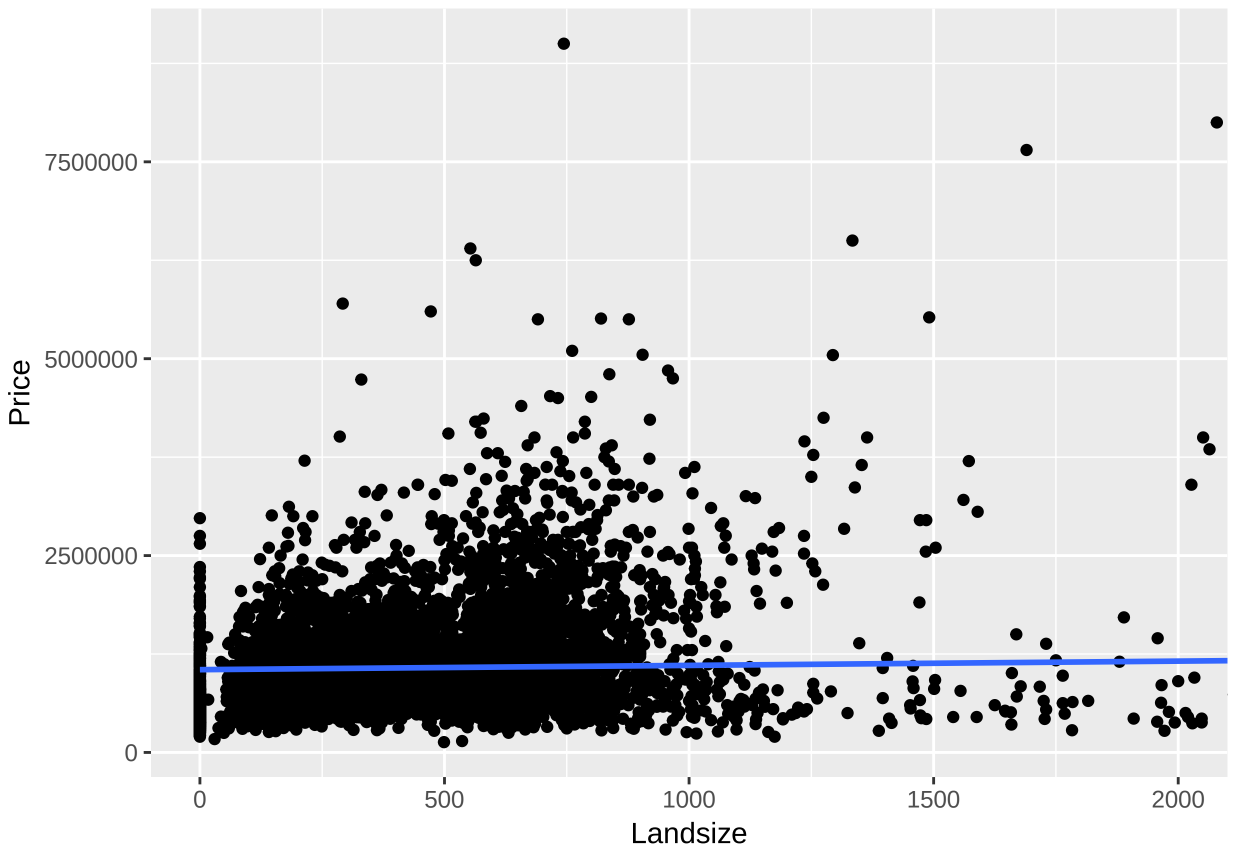
```r
ggplot(house, aes(x = Landsize, y = Price)) +
  geom_point() +
  geom_smooth() +
  coord_cartesian(xlim = c(0,2000))
```

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



You can change how the line is drawn. The default one is just a line that fits the data. Other optoms are lm (linear model), glm (general linear model).

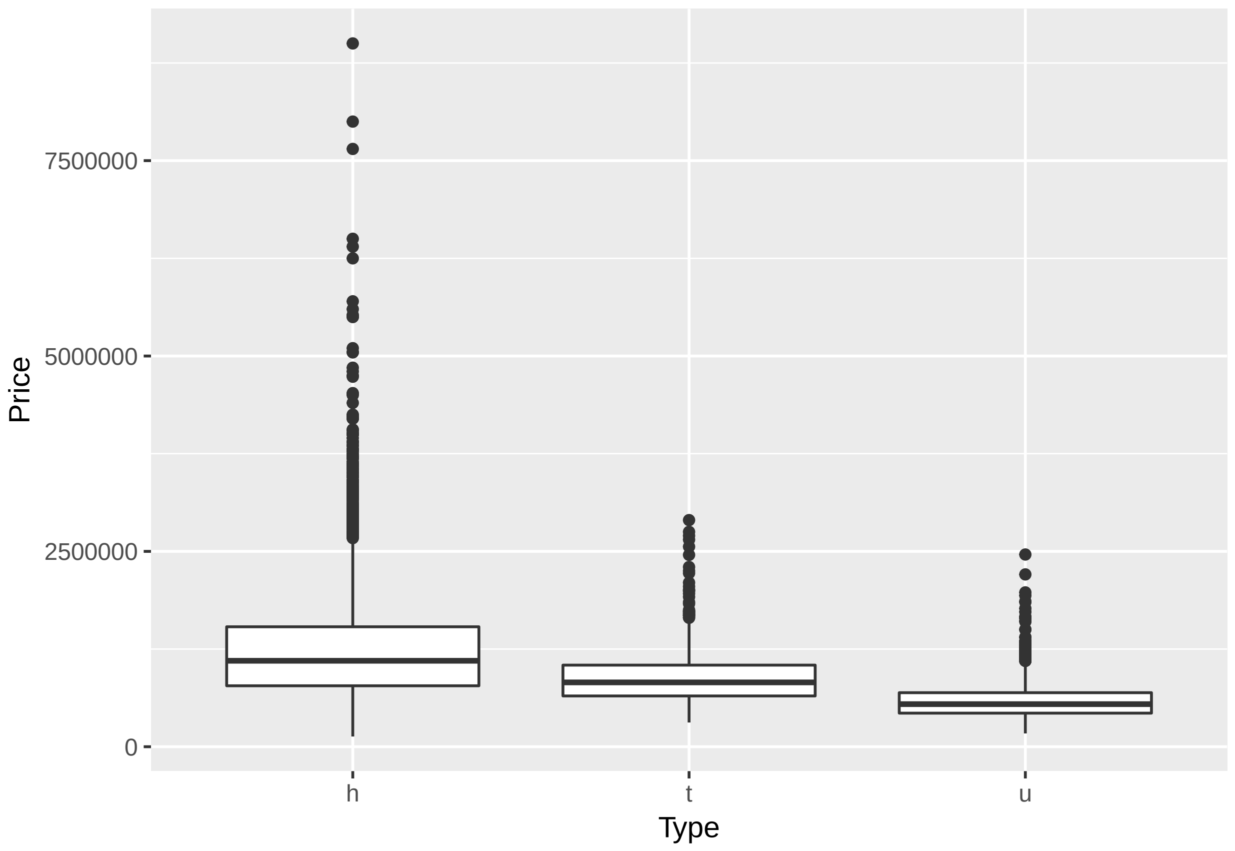
```r
ggplot(house, aes(x = Landsize, y = Price)) +
  geom_point() +
  geom_smooth(method = lm) +
  coord_cartesian(xlim = c(0,2000))
```



Boxplot

Let's move on to Boxplots.

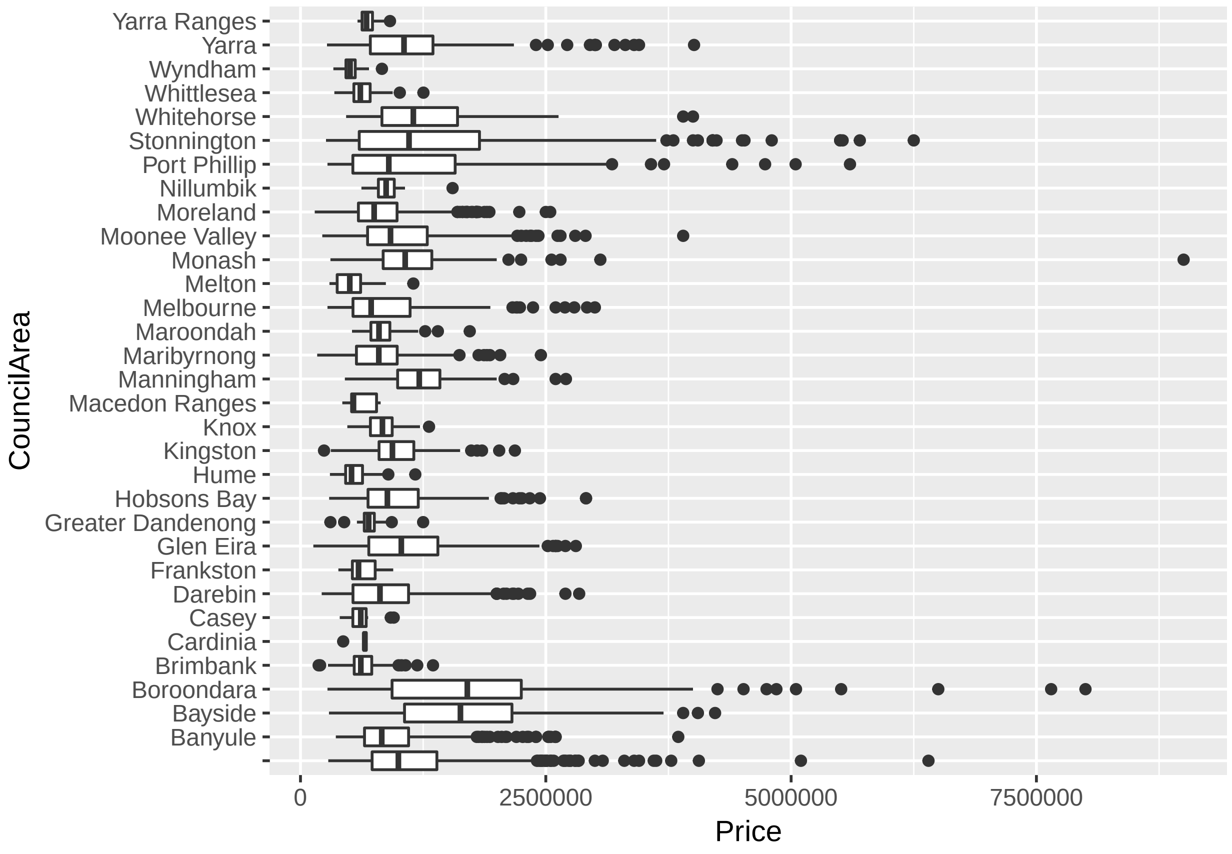
```r
ggplot(house, aes(y = Price, x = Type)) +
  geom_boxplot()
```



CHALLENGE: Make a boxplot of Price for each CouncilArea

We can't really read the labels. To fix this we can switch the axis.

```
ggplot(house) +
  geom_boxplot(aes(y = Price, x = CouncilArea)) +
  coord_flip()
```



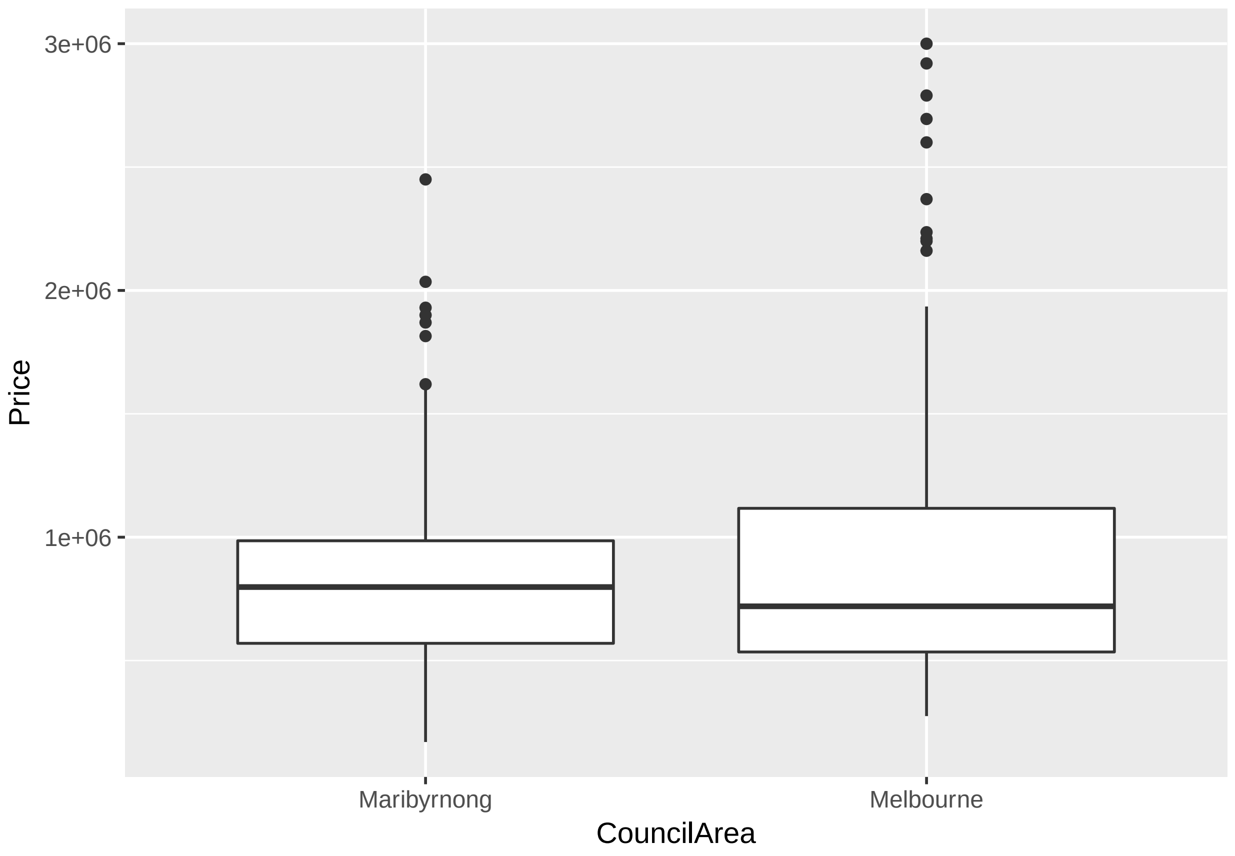
So this is getting a bit messy. We are only interested in some areas anyway. We can combine dplyr (the select and filter we were using before) with ggplot.
We start filtering using:

```
house %>%
  filter(CouncilArea %in% c("Maribyrnong", "Melbourne"))
```

I can send this data to ggplot. From here it passes it to ggplot. Usually I have to tell ggplot what data I am plotting but here I don't as it is getting the data piped in from here.

Do note that with dplyr you use %>% but with ggplot you need to use +.

```
house %>%
  filter(CouncilArea %in% c("Maribyrnong", "Melbourne")) %>%
  ggplot(aes(y = Price, x = CouncilArea)) +
  geom_boxplot()
```

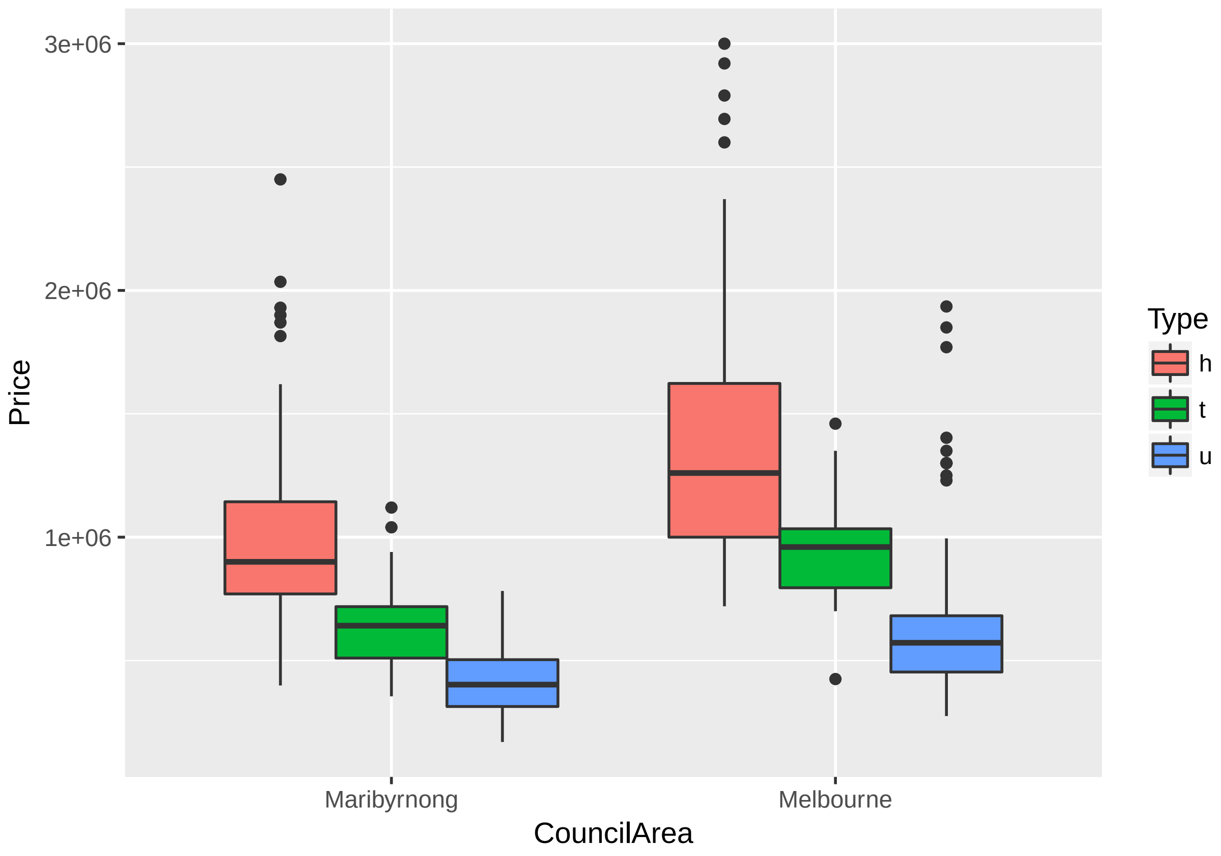


We can add another variable to out plot. We do this by plotting a side by side boxplot.

Say for every CouncilArea, we want to know the price for each type of house. **whiteboard** this is the plot that we want.

We just have to add fill = Type to aesthetics.

```
house %>%
  filter(CouncilArea %in% c("Maribyrnong", "Melbourne")) %>%
  ggplot() +
  geom_boxplot(aes(y = Price, x = CouncilArea, fill = Type))
```

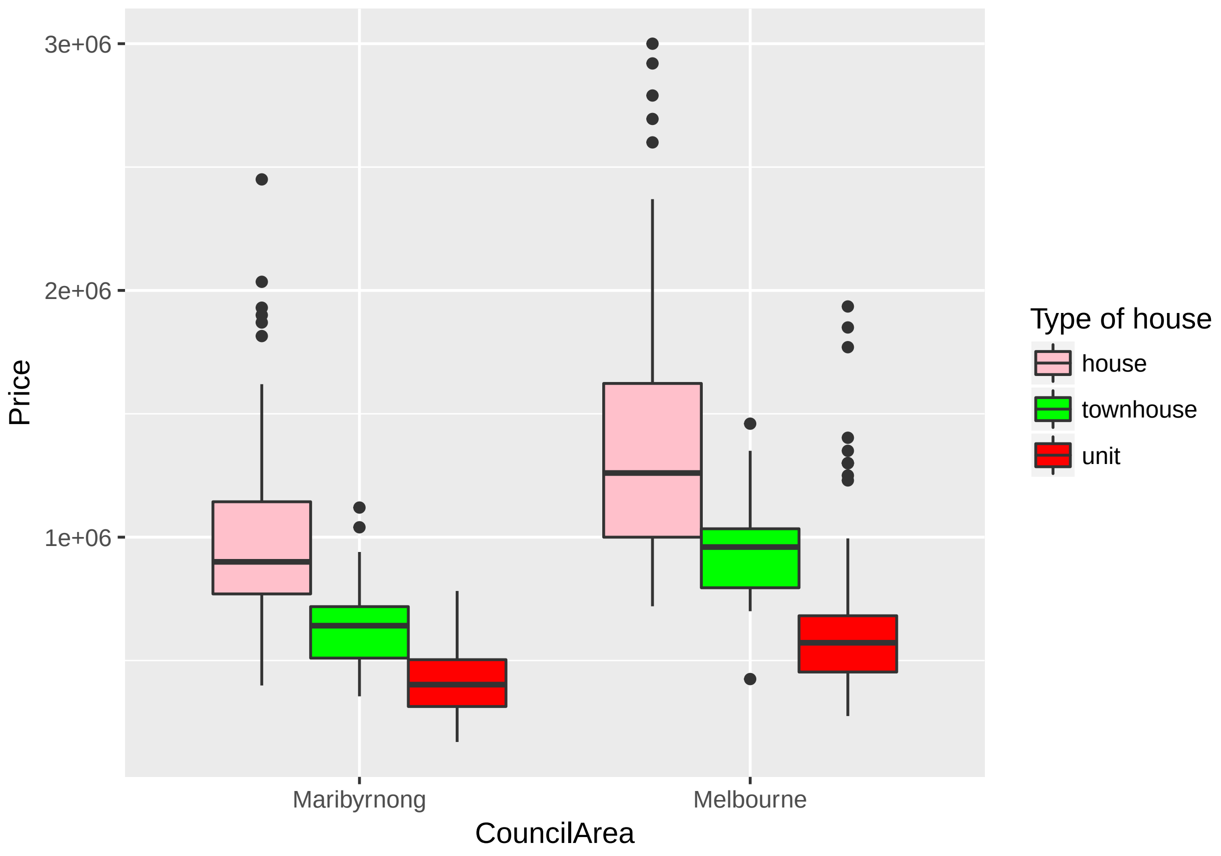


Challenge: Pick 3 CouncilAreas and make your own side by side boxplot

---

We can also change the colour. These are the default ones. scale_fill_manual lets us specify what the legend will look like.

```
house %>%
  filter(CouncilArea %in% c("Maribyrnong", "Melbourne")) %>%
  ggplot() +
  geom_boxplot(aes(y = Price, x = CouncilArea, fill = Type)) +
  scale_fill_manual(values = c("pink", "green", "red"), name = "Type of house", labels = c("house","tow
```



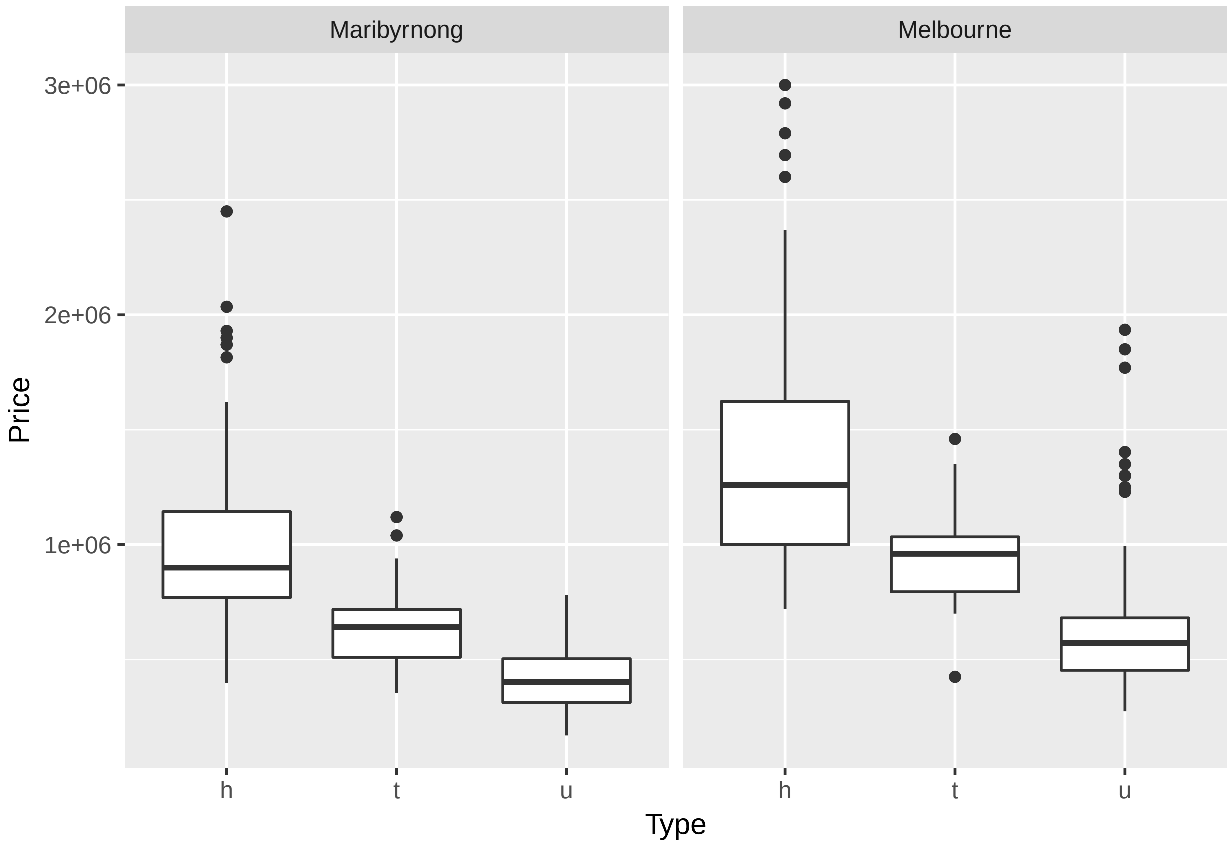
CHALLENGE: Use scale_fill_manual() to change the colours of the boxes and the labels.

Facet grid

We can also separate out these plots using something called facets.

How do I start.

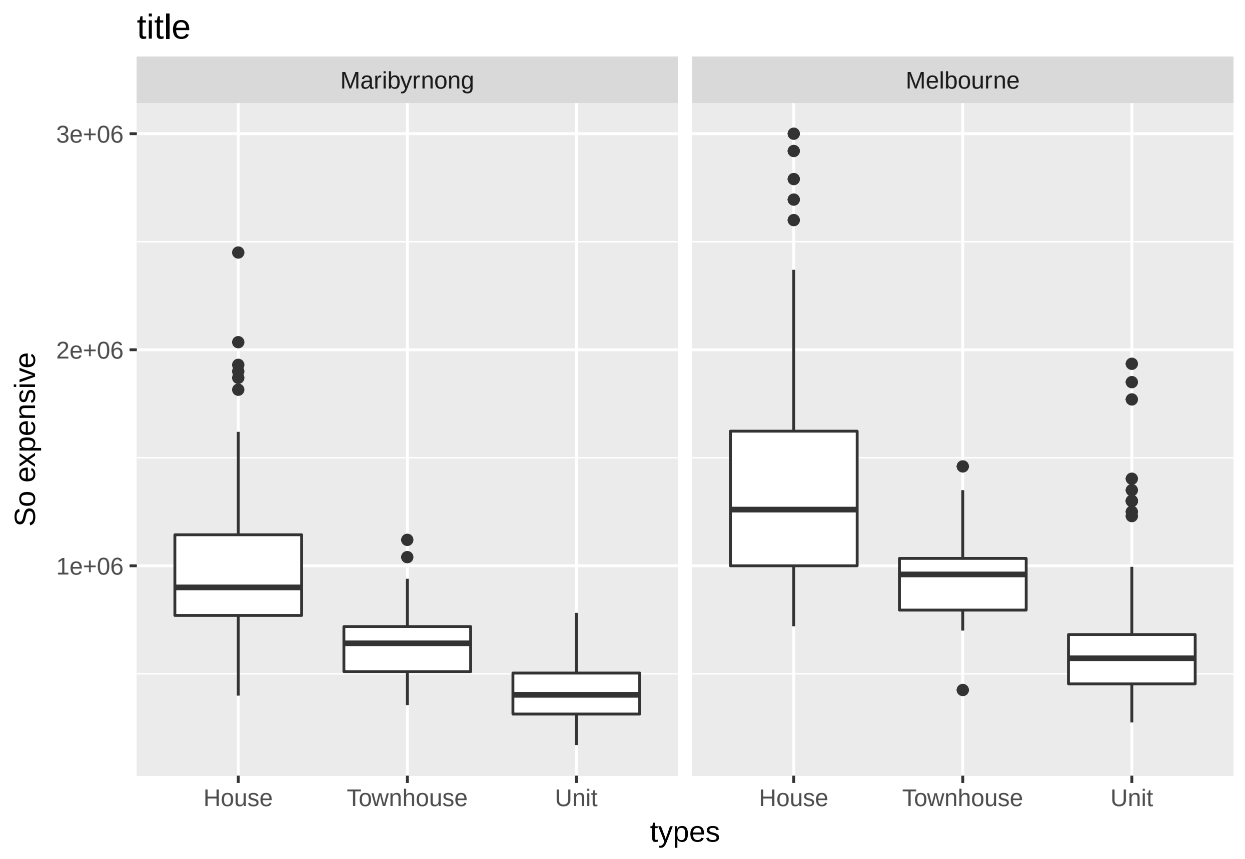
```r
house %>%
  filter(CouncilArea %in% c("Maribyrnong", "Melbourne")) %>%
  ggplot() +
  geom_boxplot(aes(y = Price, x = Type)) +
  facet_grid(.~CouncilArea)
```



What does the .~ mean? Here you tell it what should be faceted vertical ~ horizontal.

Lets make this look better. First we can add titles using scale_x_discrete and labs

```
house %>%
  filter(CouncilArea %in% c("Maribyrnong", "Melbourne")) %>%
  ggplot() +
  geom_boxplot(aes(y = Price, x = Type)) +
  facet_grid(.~CouncilArea) +
  scale_x_discrete(labels = c("House", "Townhouse", "Unit")) +
  labs(title = "title", y = "So expensive", x = "types")
```



Challenge: Add labels to your plot using scale_x_discrete() and labs()

Challenge: Change the code so that we have price in million on the y axis. Then change x axis label so that it tells us the price is in millions.

Hint: Make a new column with mutate that is the price in million.

Saving plots

So I like this plot and I want to save it. Show how to save plots in plots using Export.

Another way to save is to use pdf(). This is good if you are making 10 plots and you don't want to click through for each one.

```r
pdf("myPlot.pdf", width = 10, height = 8)

houses %>%
  ggplot() +
  geom_boxplot(aes(y = Price, x = Type))

dev.off
```

Bar plots

Let's make bar plots now! **%Whiteboard%** I want to plot mean price for each number of rooms. First I need to work out the mean price for each number of rooms. Which one of these would I use?

```
house %>%
  group_by(Bedroom2) %>%
  summarise(meanPrice = mean(Price))
```

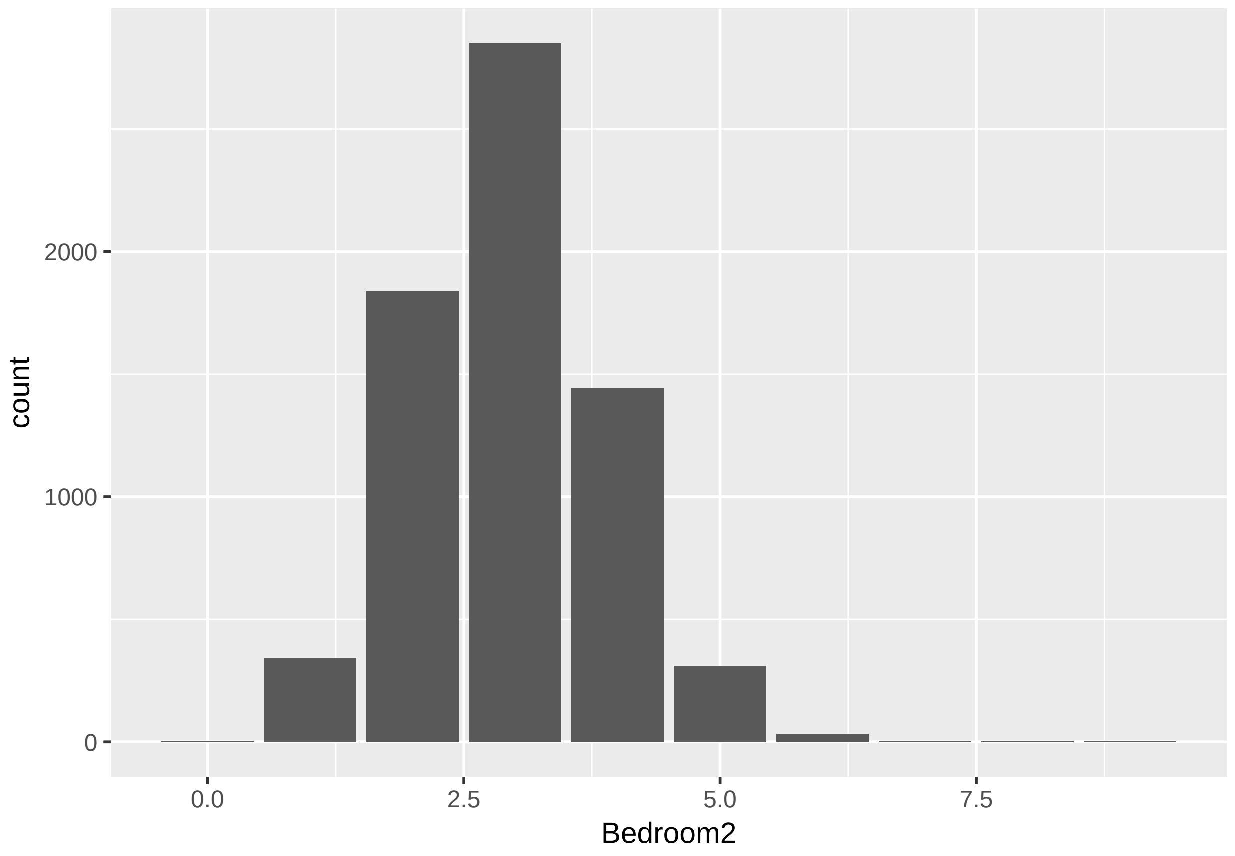
```
## # A tibble: 10 x 2
##    Bedroom2 meanPrice
##       <int>     <dbl>
##  1        0  1182000
##  2        1   423362.
##  3        2   746501.
##  4        3  1070408.
##  5        4  1469442.
##  6        5  1892509.
##  7        6  2000844.
##  8        7  1888000
##  9        8   755500
## 10        9  1487000
```

Lets plot this. Note that I send the data through to ggplot.

```
houses %>%
  group_by(Bedroom2) %>%
  summarise(meanPrice = mean(Price, na.rm = TRUE)) %>%
  ggplot() +
    geom_bar(aes(y = meanPrice, x = Bedroom2))
```

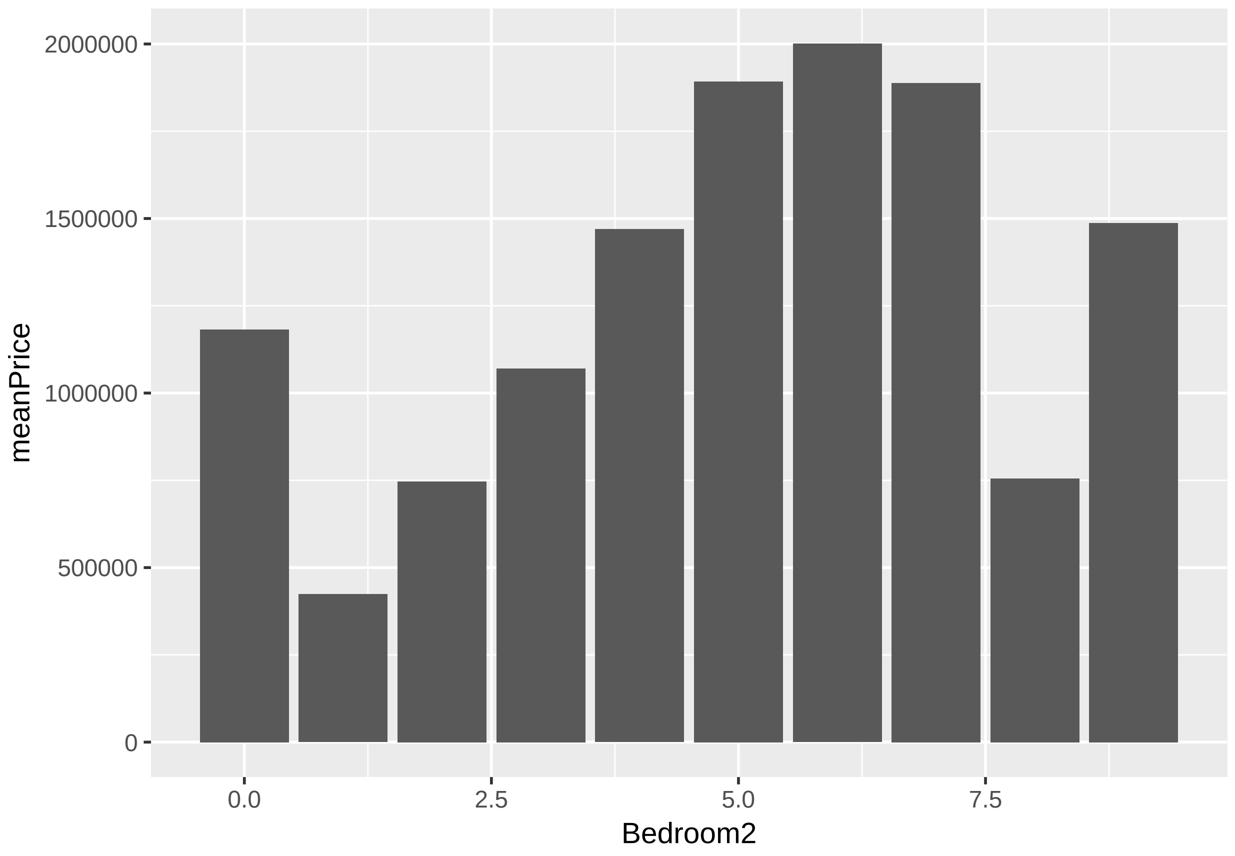
Why have we gotten this error??
Let's look at a bar plot that doesn't give an error.

```
ggplot(house) +
  geom_bar(aes(x = Bedroom2))
```



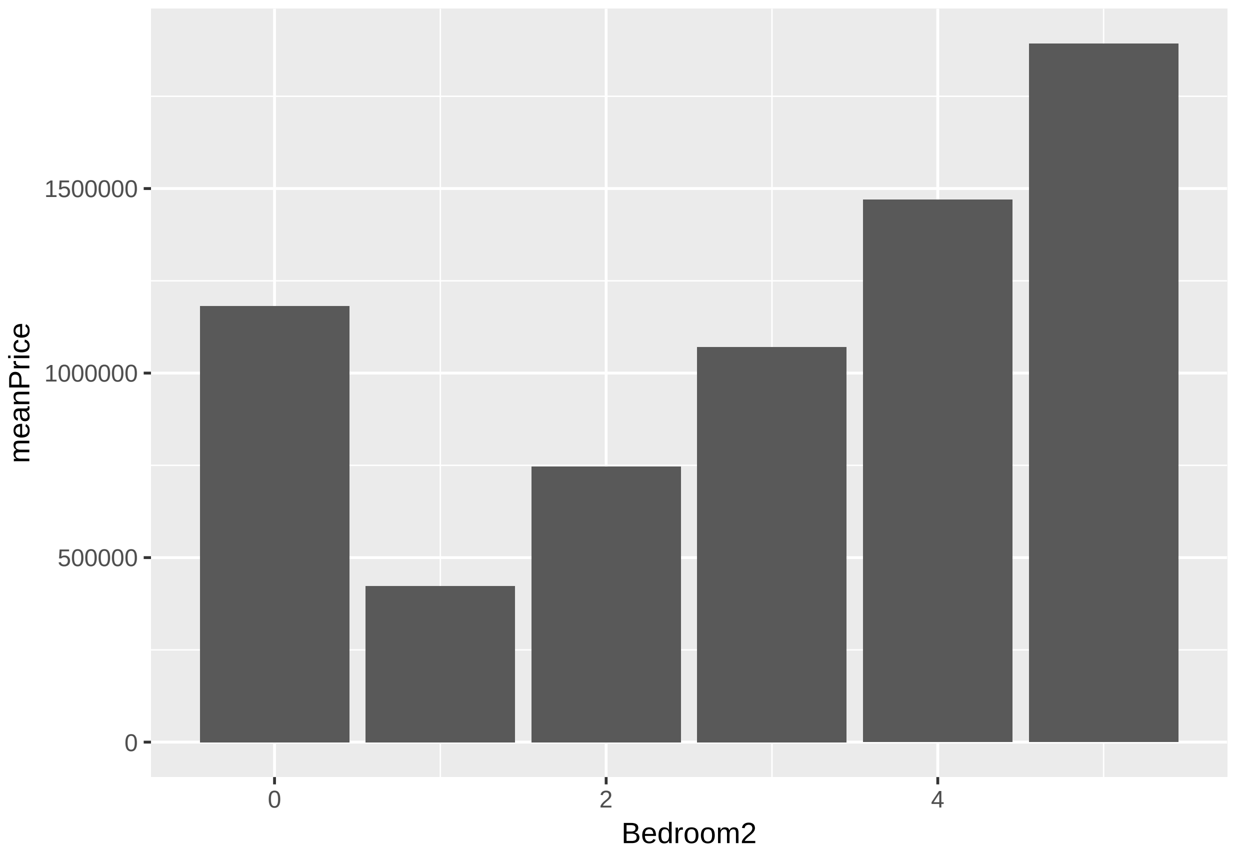
What is this plot showing? y = number of houses that had 1 bedroom, 2bds. . . .
What this function has done, is automatically counted how many houses/rows have 1 bd, 2 bd. . .
Notice how we only gave it 1 variable - because it's just counting up how many times there is a 1 bd house and putting the count herre - in the y axis.
Lets look at our previous data again. We are not plotting how many - we want it to plot actually this number here. We can tell it to do so with stat = "identity"

```
house %>%
  group_by(Bedroom2) %>%
  summarise(meanPrice = mean(Price, na.rm = TRUE)) %>%
  ggplot() +
    geom_bar(aes(y = meanPrice, x = Bedroom2), stat = "identity")
```



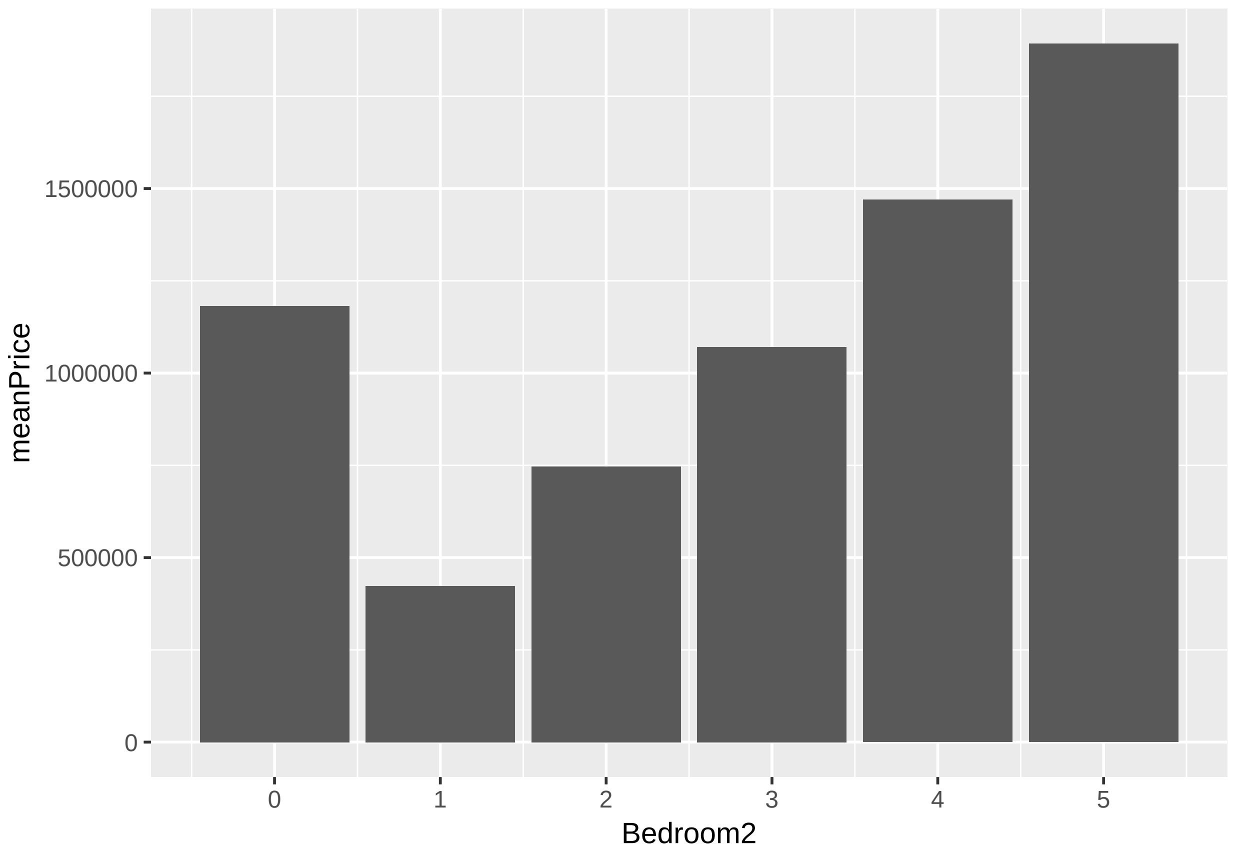
So I only want to look at houses that have 5 bedrooms or less - I'm a poor PhD student. Remember should I use == or %in% ??

```
house %>%
  filter(Bedroom2 %in% 0:5) %>%
  group_by(Bedroom2) %>%
  summarise(meanPrice = mean(Price, na.rm = TRUE)) %>%
  ggplot() +
    geom_bar(aes(y = meanPrice, x = Bedroom2), stat = "identity")
```



Okay back to our plot. I want the ticks here for each bedroom.

```
house %>%
  filter(Bedroom2 %in% 0:5) %>%
  group_by(Bedroom2) %>%
  summarise(meanPrice = mean(Price, na.rm = TRUE)) %>%
  ggplot() +
    geom_bar(aes(y = meanPrice, x = Bedroom2), stat = "identity") +
    scale_x_continuous(breaks = 0:5, labels = 0:5)
```

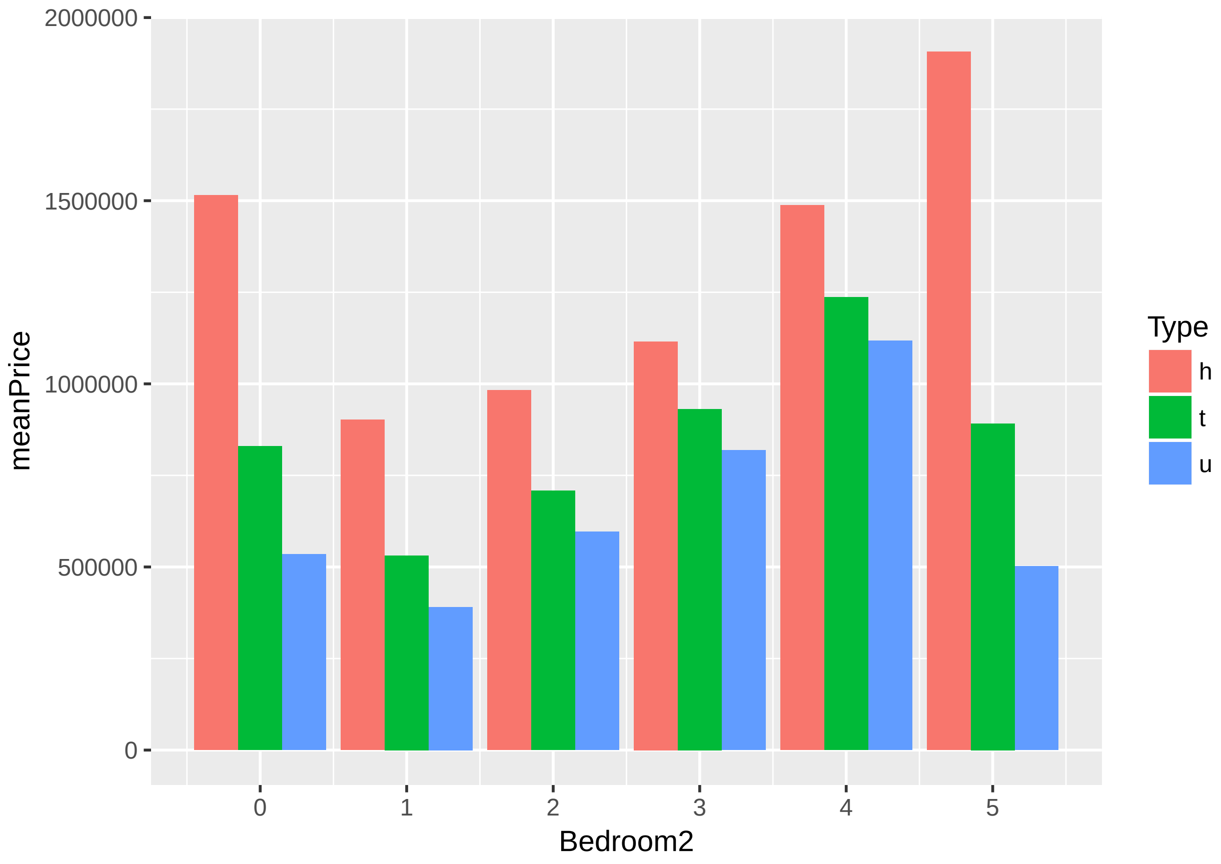


```
#breaks mean the points at which the grid lines appear
```

CHALLENGE: Make a side by side bar plot so for every bedroom number you have average price for each house Type.

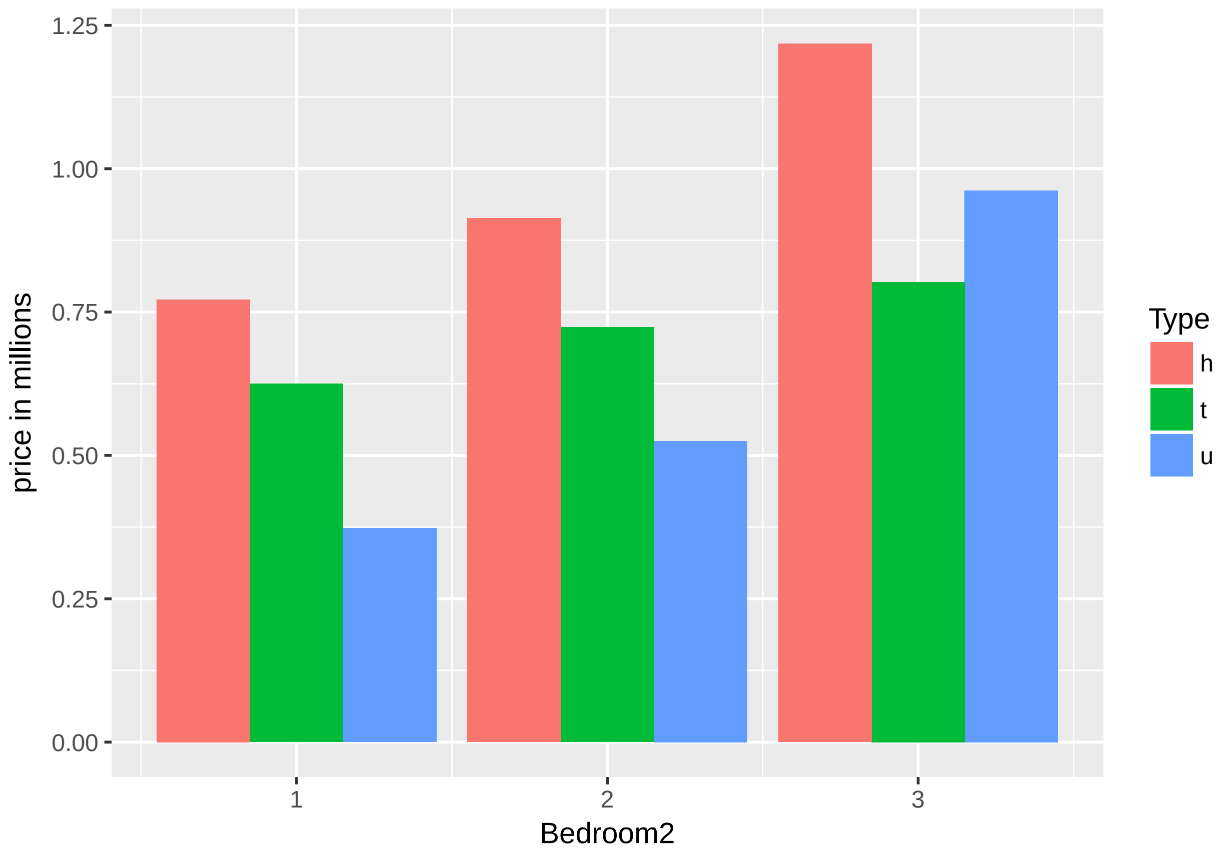
Hint: You will need to make a dataframe that shows mean price for each house type AND bedroom number then give this to ggplot.
Use position = "dodge" in geom_bar() to get the bars to be side by side Extra: Change the colours of the bars with scale_fill_manual()

```
house %>%
  filter(Bedroom2 %in% 0:5) %>%
  group_by(Bedroom2, Type) %>%
  summarise(meanPrice = mean(Price, na.rm = TRUE)) %>%
  ggplot() +
    geom_bar(aes(y = meanPrice, x = Bedroom2, fill = Type), stat = "identity", position = "dodge") +
    scale_x_continuous(breaks = 0:5, labels = 0:5)
```



This is not very good looking but you can change the labels and the colours if you want.
Now I want one of these plots for Brunswick. And I only want 1-3 bedrooms.

```
house %>%
  filter(Bedroom2 %in% 1:3 & Suburb %in% c("Brunswick")) %>%
  group_by(Bedroom2, Type) %>%
  summarise(meanPrice = mean(Price, na.rm = TRUE) / 1000000) %>%
  ggplot() +
    geom_bar(aes(y = meanPrice, x = Bedroom2, fill = Type), stat = "identity", position = "dodge") +
    labs(y = "price in millions")
```



What we covered: 1. Dplyr - key functions 2. ggplot - how to make a plot

Any questions?

Please fill out survey. Happy to hear your comments.

Would any of you be interested in attending a meetup next month, where you can talk about any problems you have with using R on your own data?