Plotting

Andrew Roth

2024-05-20

Plotting in R

One of the most important things you will want to do with your data is to visualize it. R has powerful plotting capabilities for visualizing data. By default R includes the `graphics` library with support for plotting. In practice virtually no one uses this library and instead they work with the addon library ggplot. Most often ggplot is used alongside a collection of packages called the tidyverse. In class we have installed the tidyverse package, and I will assume you have it for the remainder of the tutorial.

Getting started with ggplot

The first step to our exploration of plotting will be to get some data loaded up to work with. Here we will use the diabetes dataset we saw in class. I have converted it to a csv file, but you could also directly load the excel file if you prefer. Remember that RStudio has the option to import data for you under the File menu.

Let's load the data and take a look at it. In the code below you will need to replace the file "path" in read.csv with one that points to file on your own computer.

```
my_data <- read.csv("/home/andrew/Desktop/path/Diabetes_Full.csv")
head(my_data)
```

```
##   Random.Blood.Glucose.mg.dL Random.Blood.Glucose.Binary
## 1                        151                         Low
## 2                         75                         Low
## 3                        141                         Low
## 4                        206                        High
## 5                        135                         Low
## 6                         97                         Low
##   Random.Blood.Glucose.Ordinal Age Sex  BMI  BP Total.Cholesterol   LDL HDL TCH
## 1                       Medium  59   2 32.1 101               157  93.2  38   4
## 2                          Low  48   1 21.6  87               183 103.2  70   3
## 3                          Low  72   2 30.5  93               156  93.6  41   4
## 4                         High  24   1 25.3  84               198 131.4  40   5
## 5                          Low  50   1 23.0 101               192 125.4  52   4
## 6                          Low  23   1 22.6  89               139  64.8  61   2
##      LTG Fasting.Glucose
## 1 4.8598              87
## 2 3.8918              69
## 3 4.6728              85
## 4 4.8903              89
## 5 4.2905              80
## 6 4.1897              68
```

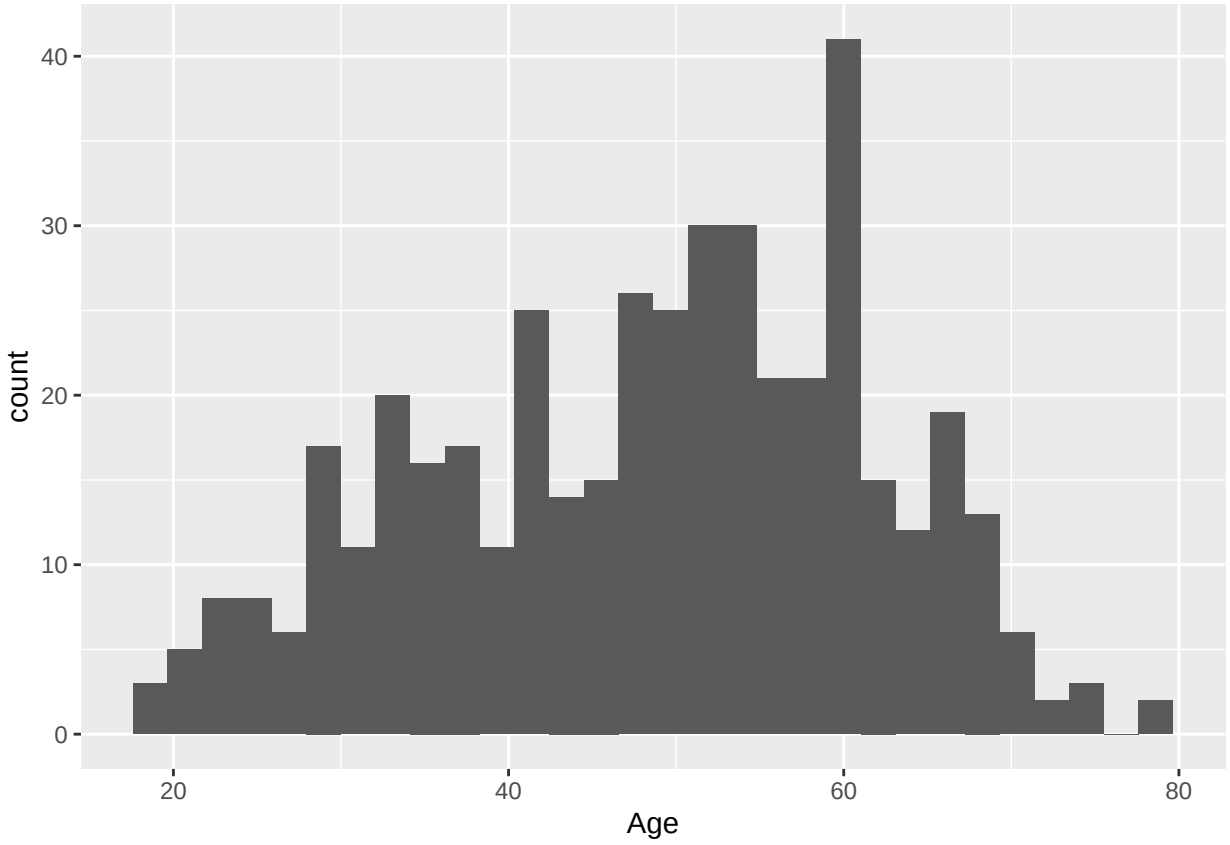
Let's make our first attempt at a plot in R. First we will load the ggplot library.

```
library(ggplot2)
```

Now let's make a simple histogram to visualize the age of the patients in the dataset.

```
ggplot(my_data, aes(x=Age)) + geom_histogram()
```

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



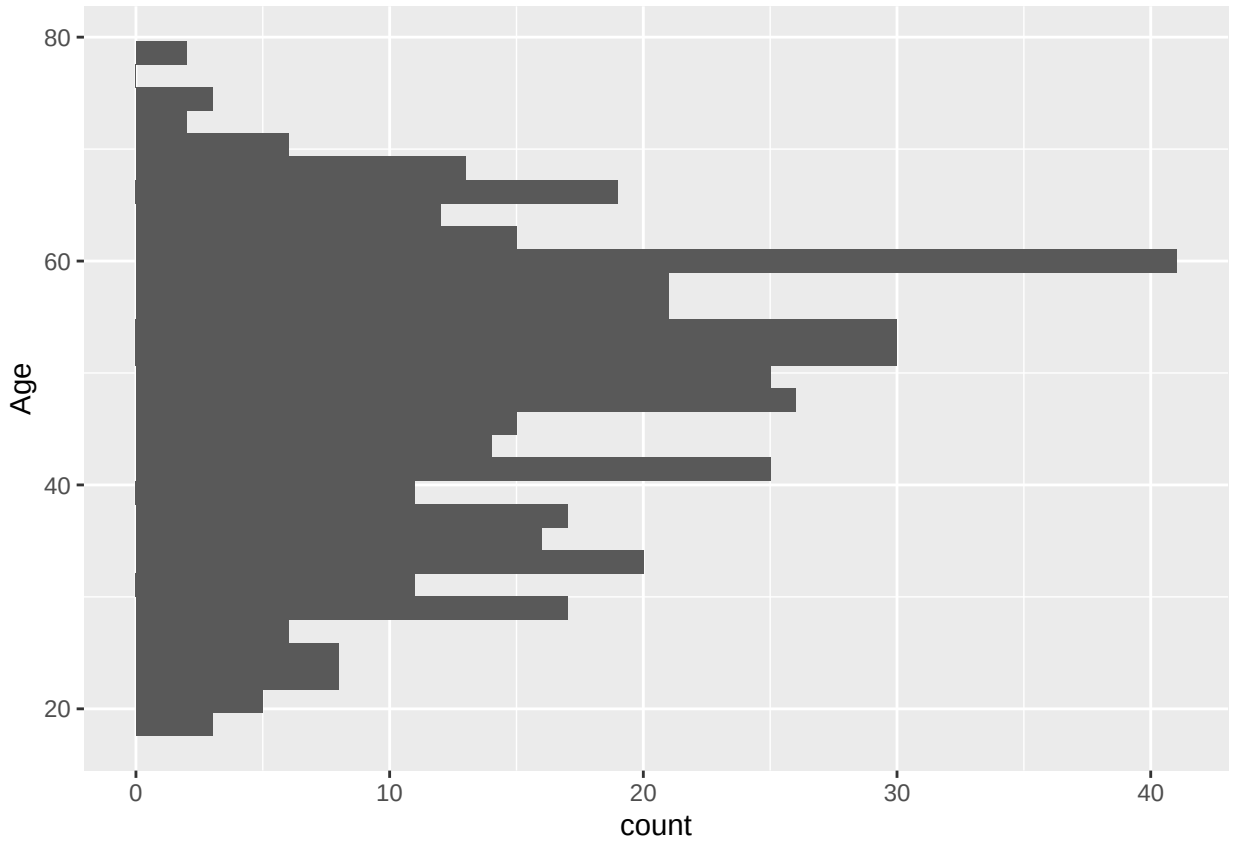
This simple example illustrates a few key concepts. First we call the `ggplot` function and pass the data as the first argument. The `ggplot` function essentially creates a blank plot. It also controls how the plot interacts with your data. You will almost always follow this pattern of having `ggplot` as the first command. We also pass a second argument which is the `aes` function with the argument `x=Age` to `ggplot`. The `aes` function is the aesthetic mapping function. Here we are telling `aes` that we want to plot `Age` on the x-axis.

Next we add the `geom_histogram()` to the `ggplot` function. This tells the ggplot library that we want to make a histogram and add it to our plot. More precisely it is telling ggplot to add a "geometry" object to the plot, in this case the geometry is a histogram. What we are plotting in the histogram is defined previously by the `aes` argument to the `ggplot` function.

Sticking with this simple example we could change the orientation of the histogram by letting ggplot know we want to associate age with the y-axis as follows.

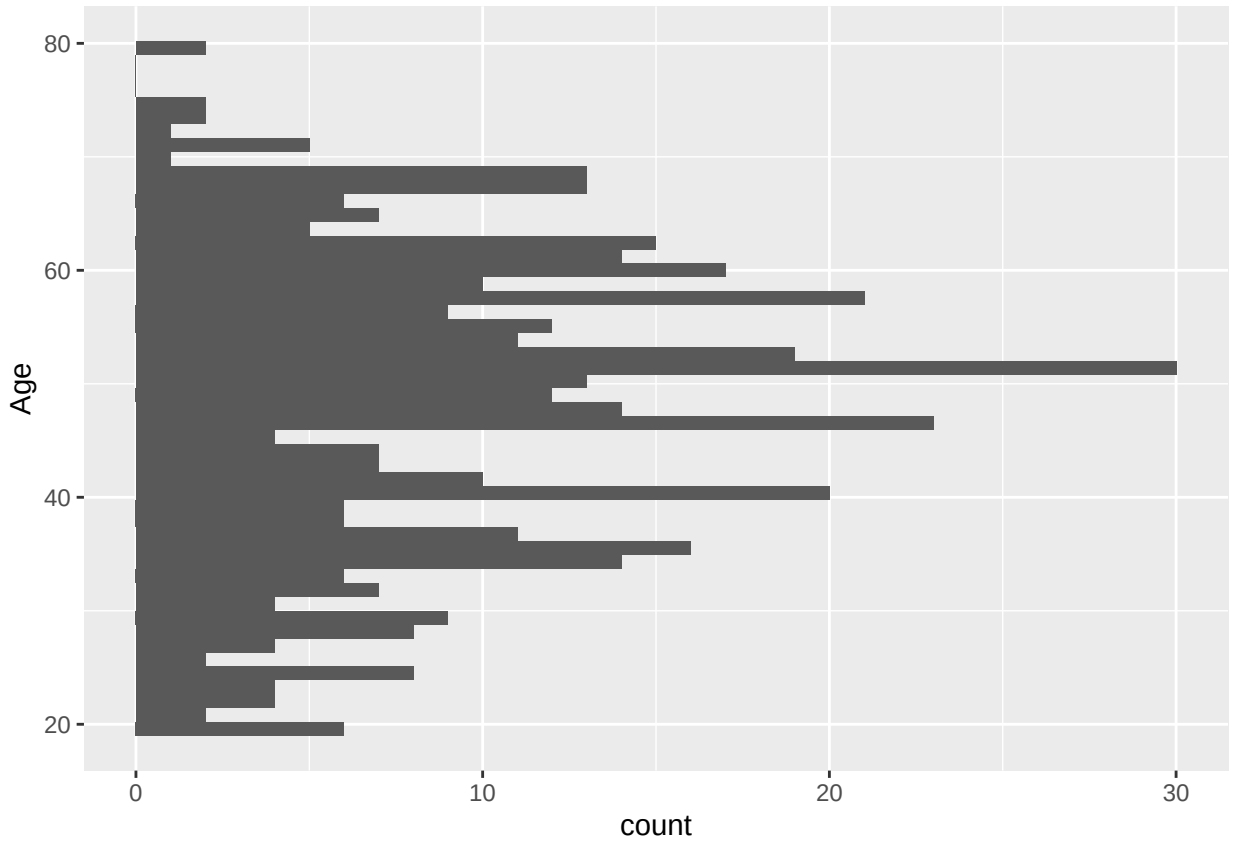
```
ggplot(my_data, aes(y=Age)) + geom_histogram()
```

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



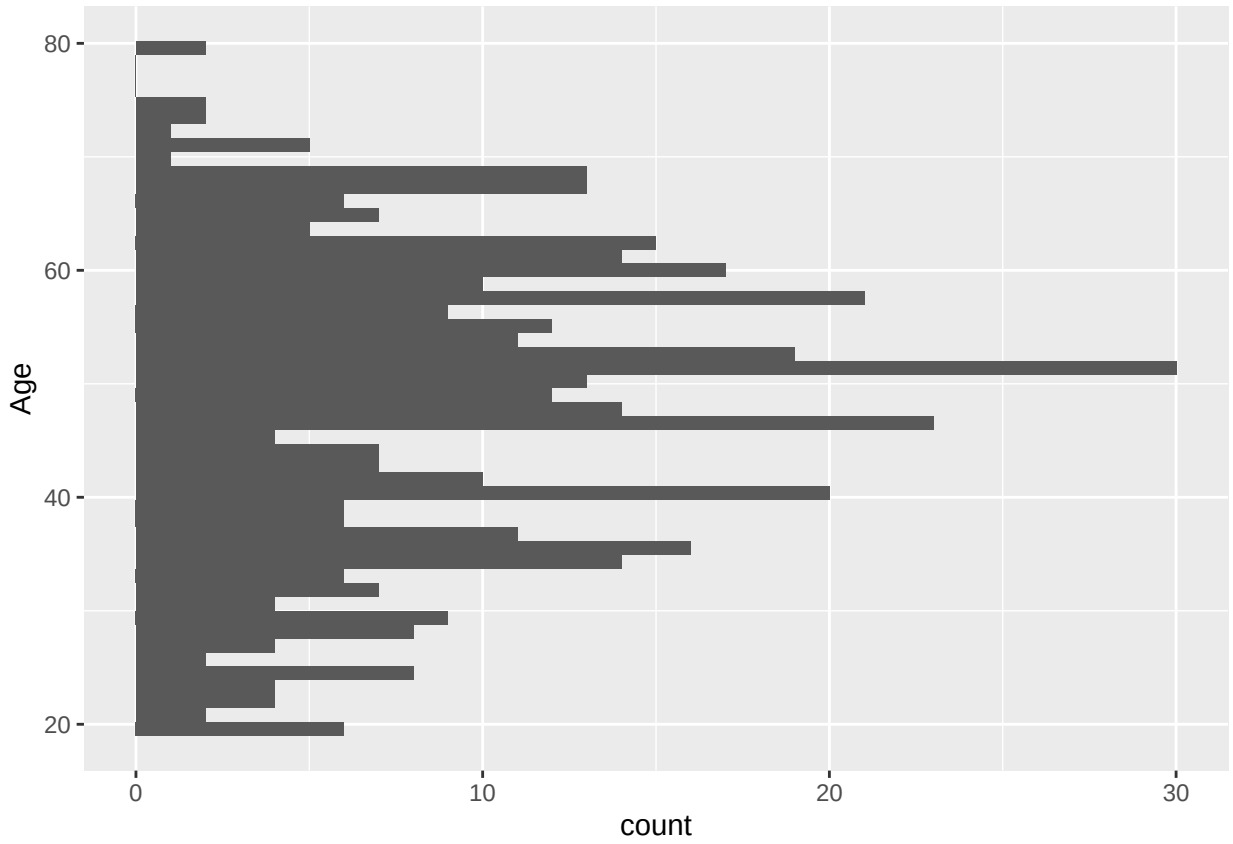
You'll see we are getting a warning about using a default bin size. We can get rid of this by choosing the number of bins we use. In the code below we will use 50 bins.

```
ggplot(my_data, aes(y=Age)) + geom_histogram(bins=50)
```



In general you will pass the `aes` function to the `ggplot` function. But as you get into more sophisticated plots you may want to have different mappings for different geometries. This can be accommodated by passing the `aes` function directly to your geometry functions.

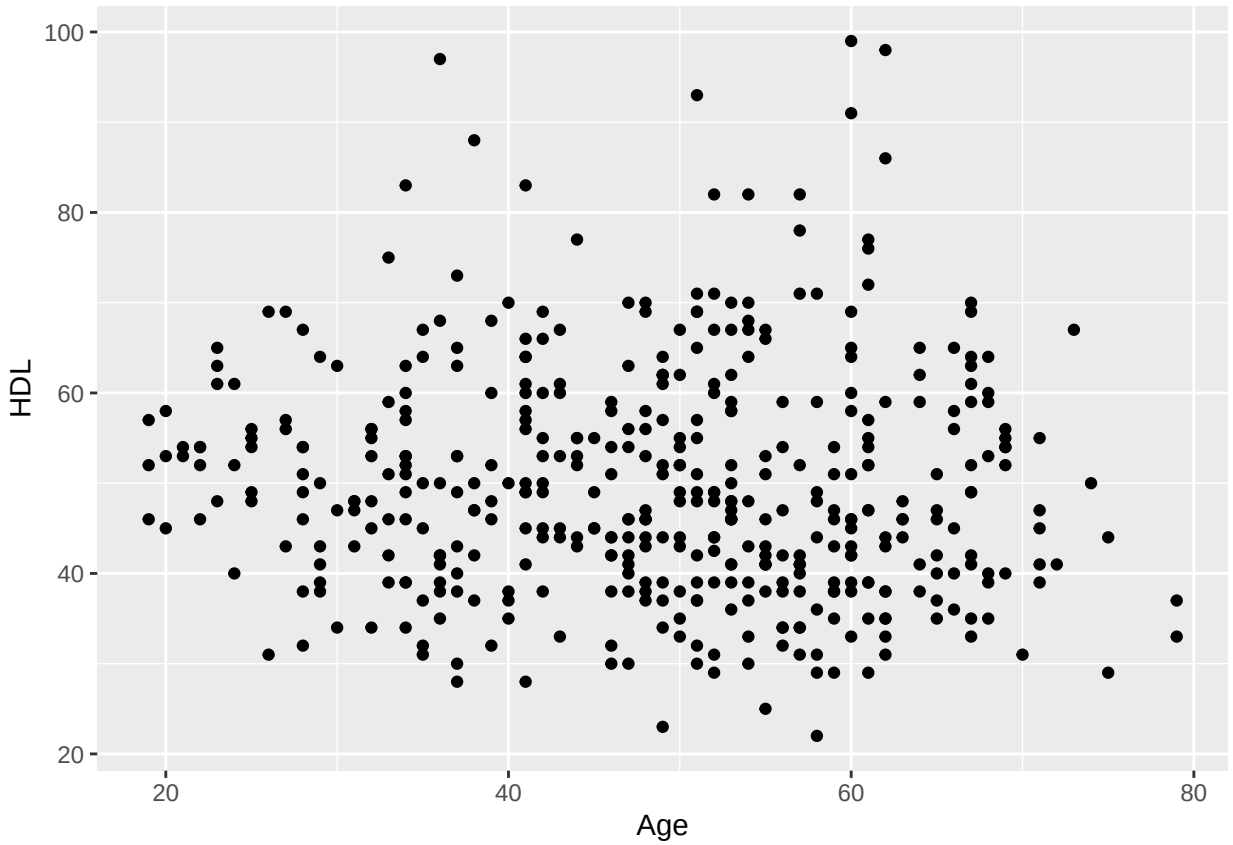
```
ggplot(my_data) + geom_histogram(bins=50, mapping=aes(y=Age))
```



Plotting two variables

So far we have created a plot of a single variable. But often you would like to plot two variables together to see their relationships. In the next example we will plot Age and HDL together using a scatter plot to explore the relationship between the variables.

```
ggplot(my_data, aes(x=Age, y=HDL)) + geom_point()
```

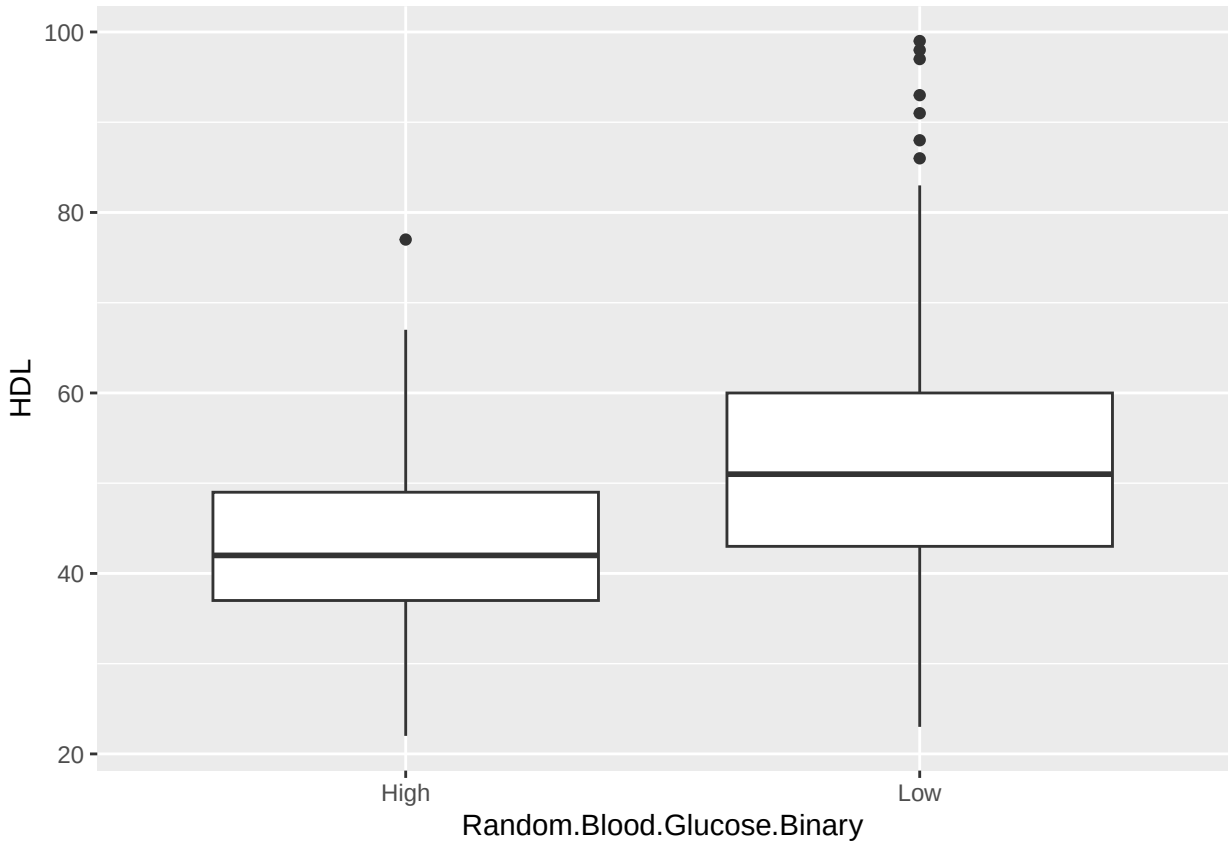


This looks a lot like the histogram code, but we have two new elements. First, we are specifying two mappings in the `aes` function. We are saying the x-axis should be the Age column and the y-axis the HDL column. Second, we are using a new geometry type with the `geom_point` function. This adds points to the plot, and in our case lets us create a scatter plot.

Often we will want to explore whether different "conditions" in our data have different distributions. For example, you way want to see if the weight of mice is different depending on treatment or diet. A quick and easy way to do this using boxplots.

In this dataset we have a column called Random.Blood.Glucose.Binary which is a binary variable which separates patients into high/low blood glucose. Let's see how the HDL values distribute based on this variable.

```
ggplot(my_data, aes(x=Random.Blood.Glucose.Binary, y=HDL)) + geom_boxplot()
```



ggplot has done something smart for us in this case. Namely it has realized that Random.Blood.Glucose.Binary which is a column with character type actually refers to different types or factors. We can see what type R thinks the column is as using the `typeof` function.

```
typeof(my_data$Random.Blood.Glucose.Binary)
```

```
## [1] "character"
```

In practice we should be more explicit and let R know that we have the categorical "factors" in our data. This can be done as follows.

```
my_data$Random.Blood.Glucose.Binary <- factor(my_data$Random.Blood.Glucose.Binary)
```

The above code is really saying create a new column vector with the type factor with the code `factor(my_data$Random.Blood.Glucose.Binary)`. We then overwrite the previous value of Random.Blood.Glucose.Binary using the `<-` assignment operator.

Now we see that R thinks that Random.Blood.Glucose.Binary is an integer.

```
typeof(my_data$Random.Blood.Glucose.Binary)
```

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

This is a bit confusing, since we really want to know if R thinks it is a factor. We can explicitly check with `is.factor` function.

```
is.factor(my_data$Random.Blood.Glucose.Binary)
```

```
## [1] TRUE
```

Which confirms that we have successfully let R know that our column is a factor.

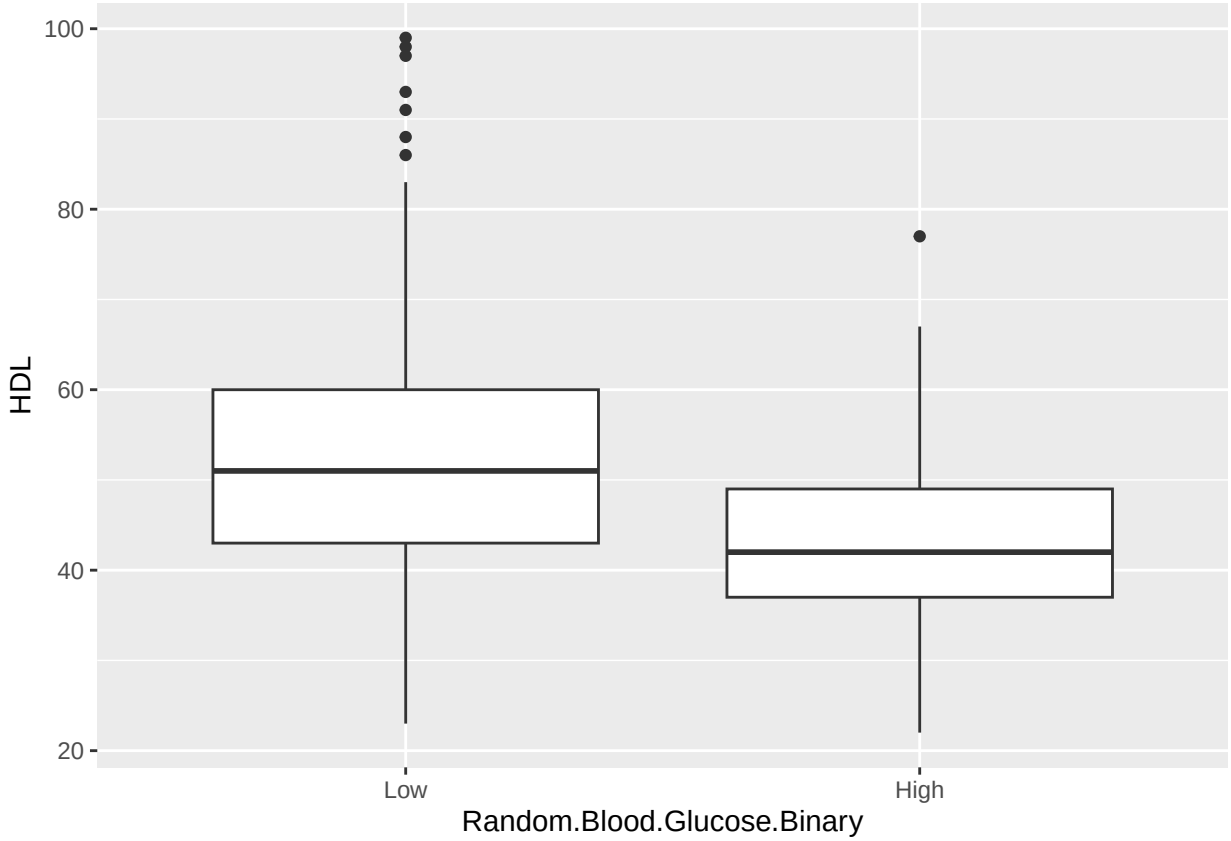
If you are using RStudio you can also look at your data frame in the Environment tab.

Being explicit about factors will become essential when doing statistical tests later. But it also has some implications for plotting. You will notice in our plots that High appears to right of Low in the plot. Usually when you have variables like this with an order, you would want the smaller values on the right. We can do this by using the `levels` argument as follows.

```
my_data$Random.Blood.Glucose.Binary <- factor(
  my_data$Random.Blood.Glucose.Binary,
  levels=c("Low", "High")
  )
```

The previous code tells R that the levels should be ordered as Low then High. We use the combine function `c` to explicitly pass a vector with the factor levels in the order we want. Let's try our boxplots again.

```
ggplot(my_data, aes(x=Random.Blood.Glucose.Binary, y=HDL)) + geom_boxplot()
```



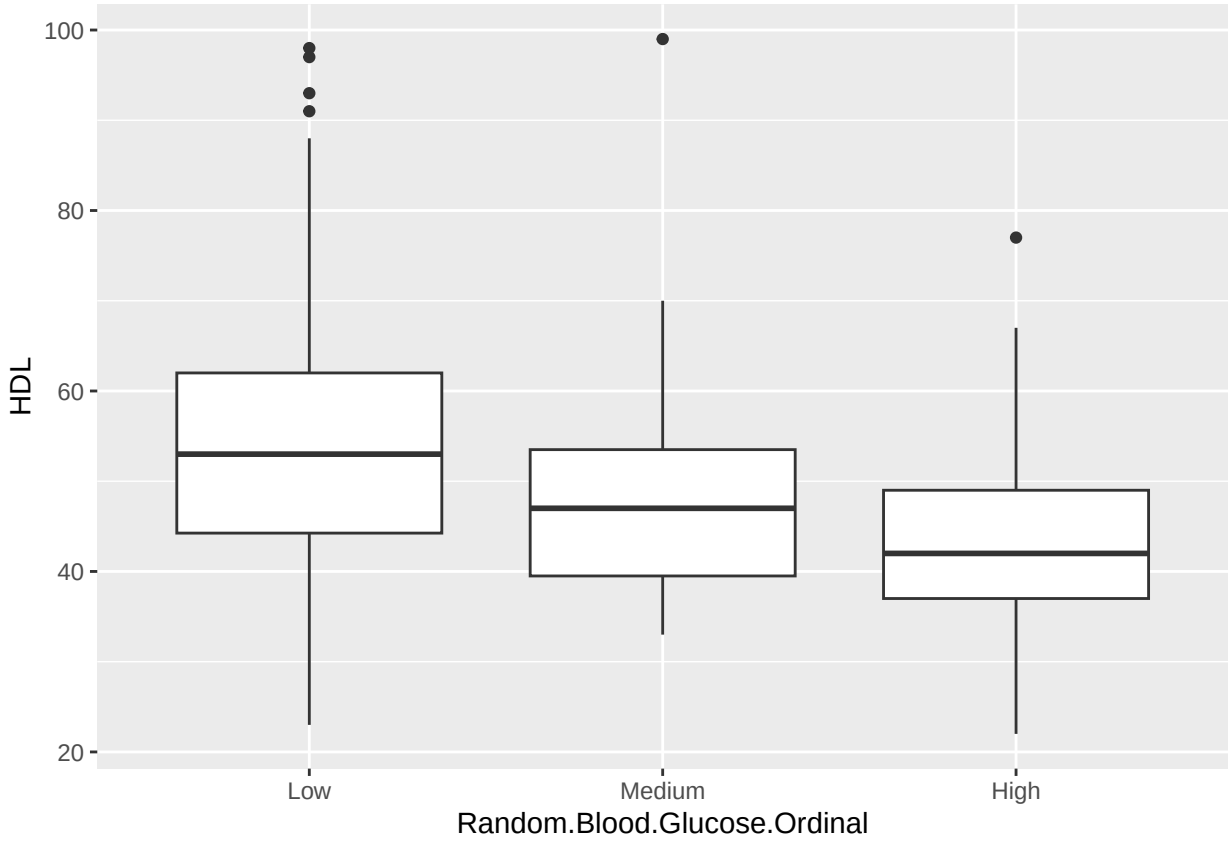
As expected we know get Low to the left of High in the plot! This dataset also has another categorical column related to blood glucose called Random.Blood.Glucose.Ordinal. Instead of low and high, this column has three possible levels low, medium, high. Currently R thinks this is a string column. Before turning it into a factor and setting the levels, we should make sure the values are what we expect. We can look at all possible unique values in the columns as follows.

```
unique(my_data$Random.Blood.Glucose.Ordinal)
```

```
## [1] "Medium" "Low"    "High"
```

Here we see that as expected there are three values "Medium", "Low" and "High" Let's convert this column to a factor, set the levels and make boxplot to look at HDL.

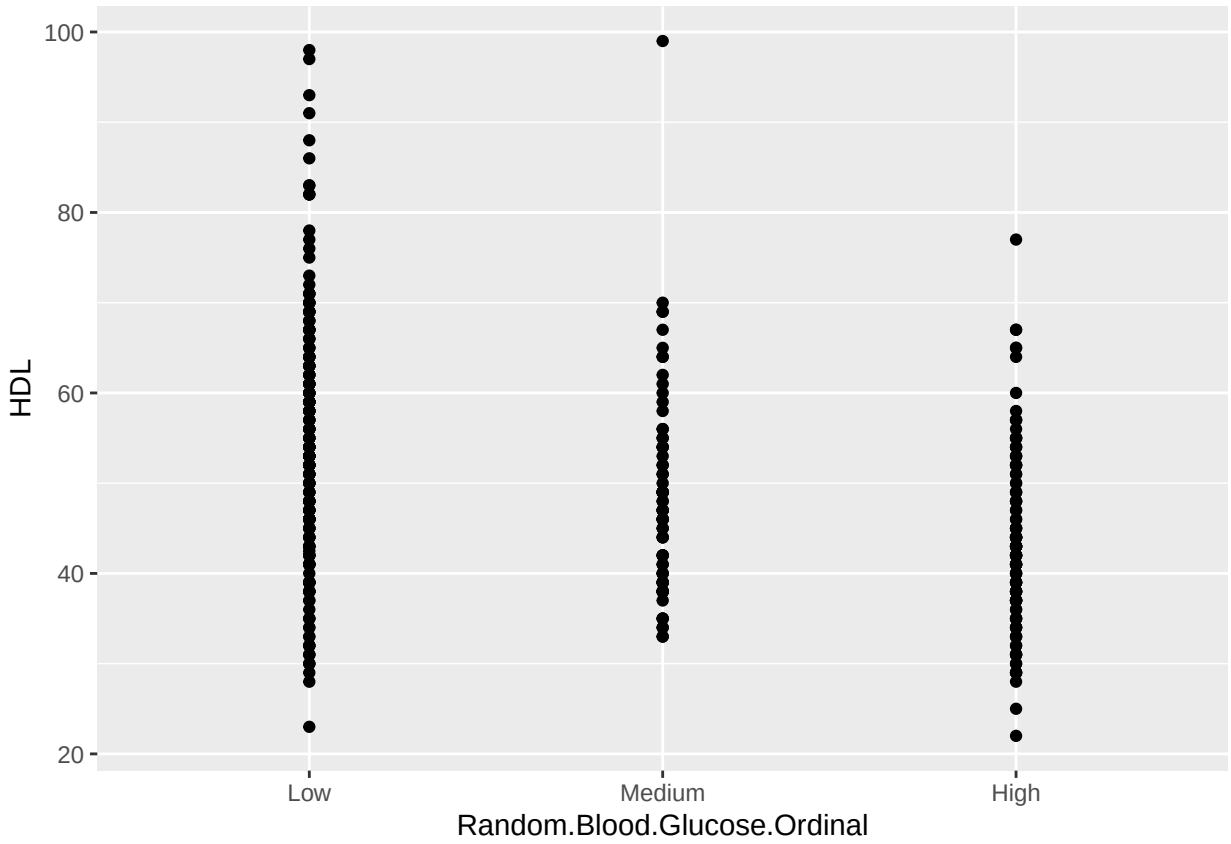
```
my_data$Random.Blood.Glucose.Ordinal <- factor(
  my_data$Random.Blood.Glucose.Ordinal,
  levels=c("Low", "Medium", "High")
  )
ggplot(my_data, aes(x=Random.Blood.Glucose.Ordinal, y=HDL)) + geom_boxplot()
```



This looks good, we are plotting things in the order we expect!

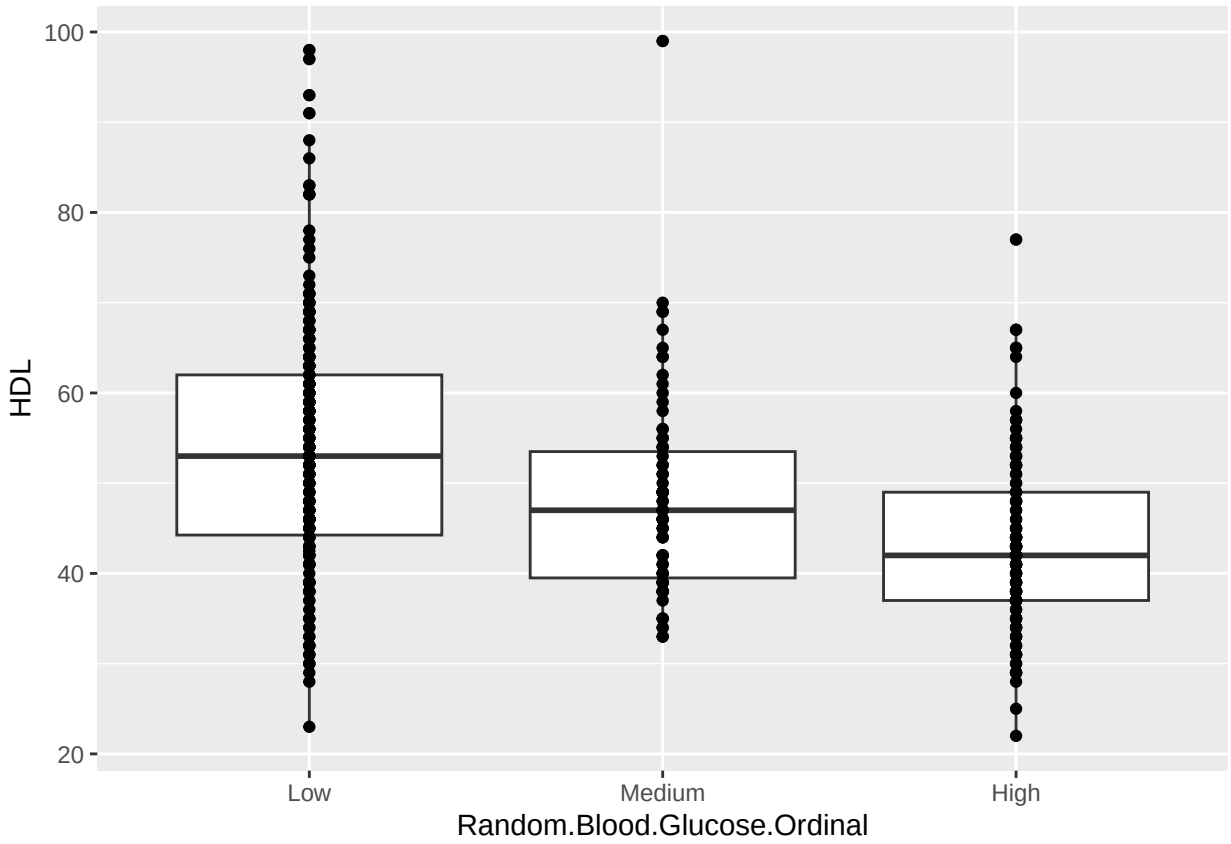
Now boxplots are great, but they can sometimes hide the variability in the data. One way to see this variability is to use a point plot instead. To do this we simply switch the geometry we are using from `geom_boxplot` to `geom_point`.

```
ggplot(my_data, aes(x=Random.Blood.Glucose.Ordinal, y=HDL)) + geom_point()
```



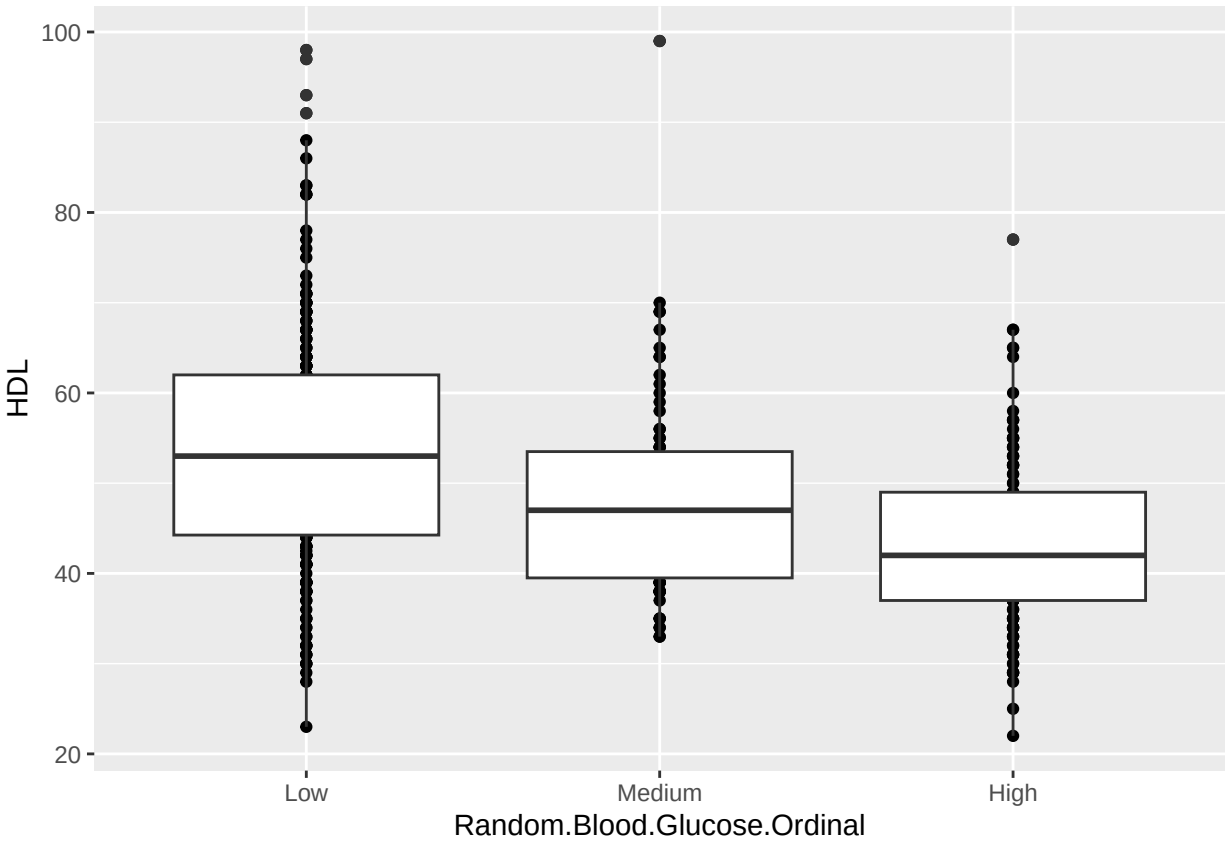
We can also combine the two and have our points on-top of the boxplot.

```
ggplot(my_data, aes(x=Random.Blood.Glucose.Ordinal, y=HDL)) + geom_boxplot() + geom_point()
```



Note what happens if we switch the order that we add `geom_boxplot` and `geom_point`.

```
ggplot(my_data, aes(x=Random.Blood.Glucose.Ordinal, y=HDL)) + geom_point() + geom_boxplot()
```



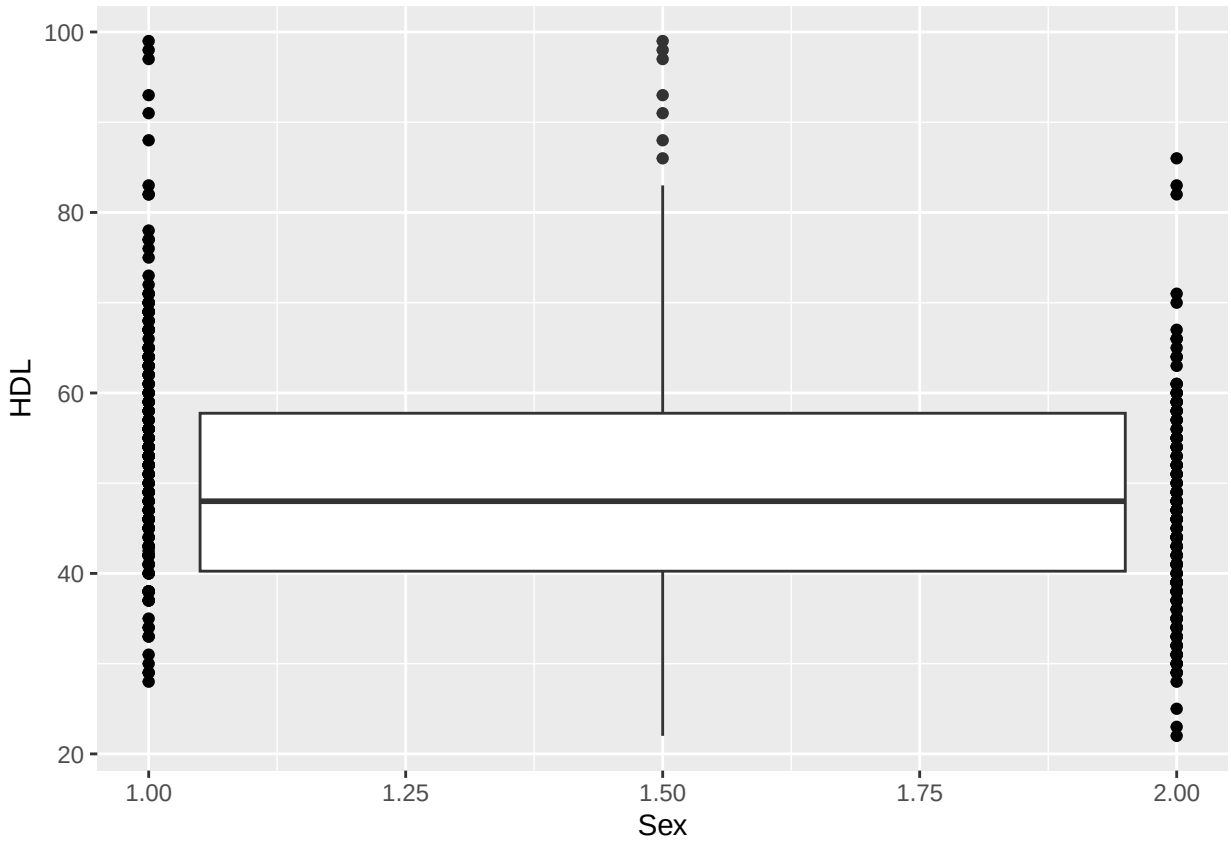
Now the boxplots are in front of the points. This illustrates the concept of layering on plot elements in ggplot. Things get added on-top of the previous things in the plots. In this case the points are plotted first and the boxplots are added on-top. In this case we would probably prefer the points be on-top of the boxplots as in the first code example.

Plotting three variables

Often we may have additional structure beyond the condition we are testing. For example in this dataset there is a sex variable. Let's start simple and look at how HDL distributes by sex.

```
ggplot(my_data, aes(x=Sex, y=HDL)) + geom_boxplot() + geom_point()
```

```
## Warning: Continuous x aesthetic
## i did you forget `aes(group = ...)`?
```



This does not produce the expected result. The issue is that ggplot does not know that sex is a categorical factor. Let's see what R thinks it is and what values it has.

```
typeof(my_data$Sex)
```

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

```
unique(my_data$Sex)
```

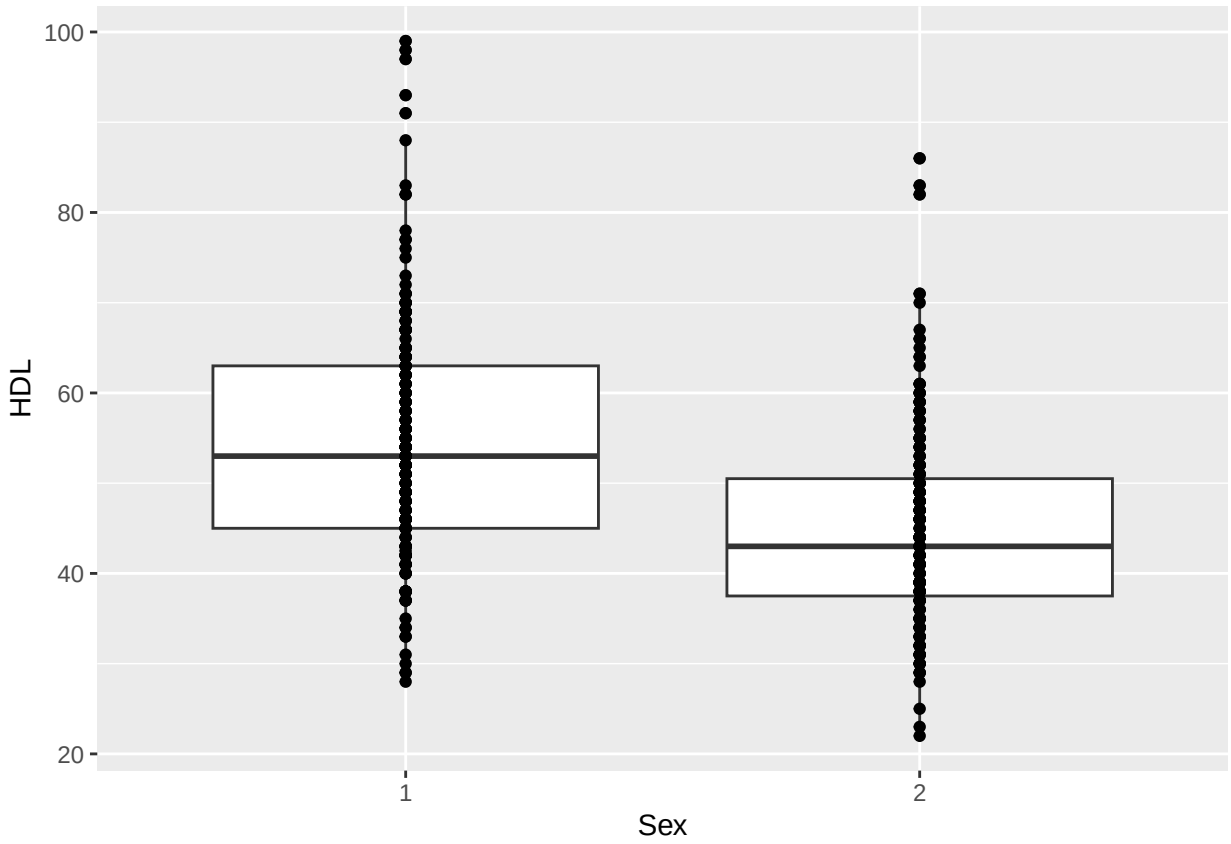
```
## [1] 2 1
```

So R thinks that the Sex column is an integer. There are two values in this case for Sex. We have already seen how to fix this issue using the `factor` function, so let's do it.

```
my_data$Sex <- factor(my_data$Sex)
```

Let's try plotting again.

```
ggplot(my_data, aes(x=Sex, y=HDL)) + geom_boxplot() + geom_point()
```



That looks better. But having sex be 1 or 2 might not be what we want for a publication. I will assume here that 1 refers to male and 2 refers to female. There are many ways to solve this. Later in the data wrangling tutorial we will see how `dplyr` can help us. For now I will stick to a solution using builtin R functionality.

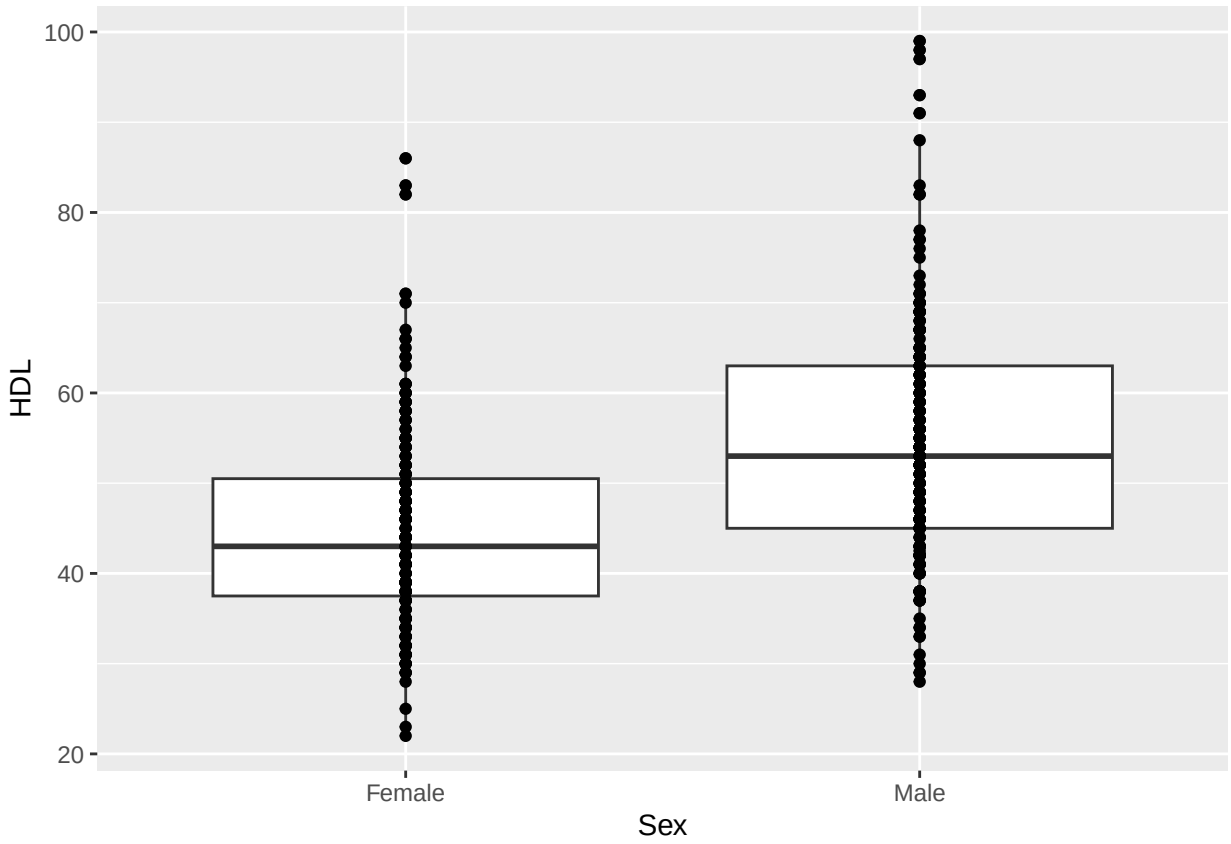
```
my_data$Sex <- as.character(my_data$Sex)
my_data$Sex[my_data$Sex == 1] <- "Male"
my_data$Sex[my_data$Sex == 2] <- "Female"
my_data$Sex <- factor(my_data$Sex, levels=c("Female", "Male"))
```

This is not the nicest solution but it works. First, I have told R to interpret the Sex column as a character in the first line using the `as.character` function. This is just like the way we tell R to think about a column as a factor using the `factor` function. Next, I have used R indexing to first find all places the column has a 1 value and change it to Male. I do the same thing for the value 2 and female. Finally, I turn the column back into a factor and set the levels. Here the levels aren't really important as there is not an order, but it is helpful for plotting.

> I had to turn the Sex column into a character vector because R will not let you specify values in a factor column that it does not know about. There other ways around this using levels I think, but the current solution works.

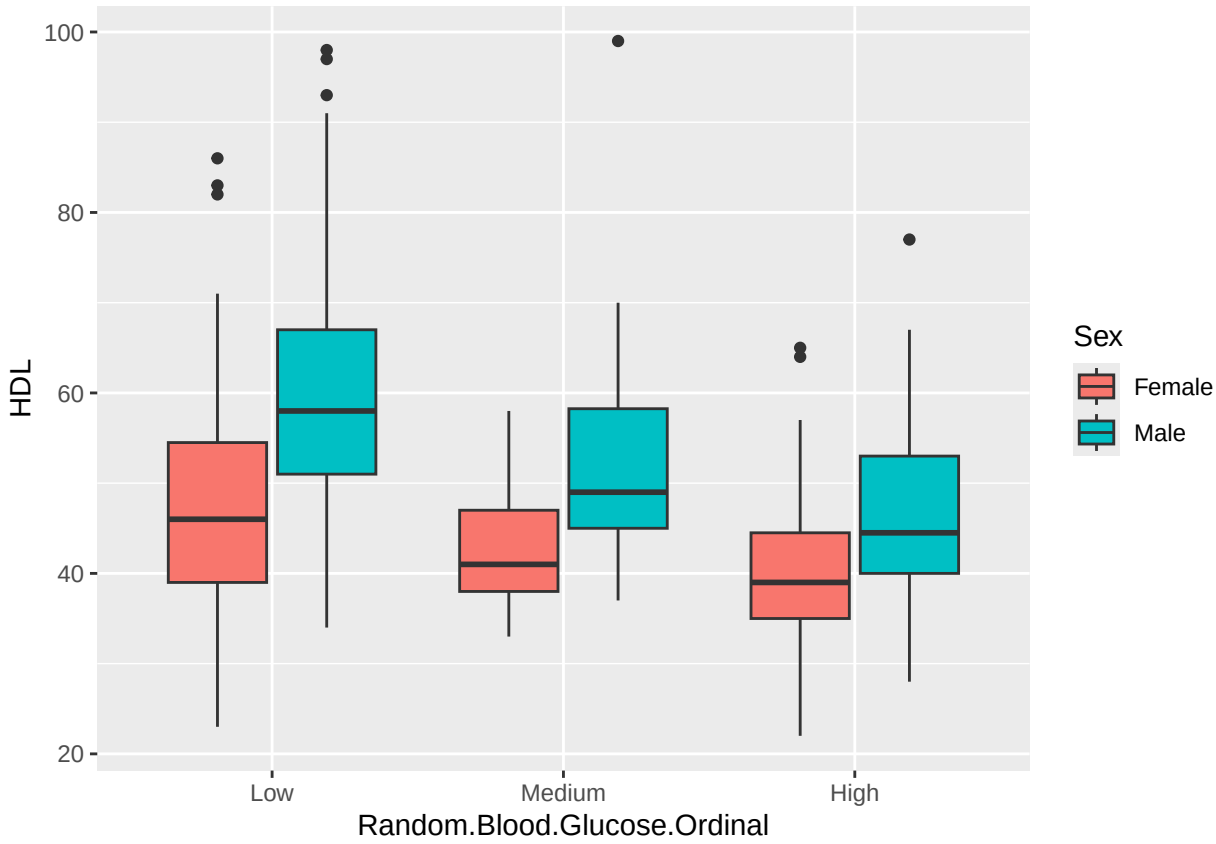
Let's see how it looks after that work.

```
ggplot(my_data, aes(x=Sex, y=HDL)) + geom_boxplot() + geom_point()
```



Now it looks like HDL is higher in males. But maybe blood glucose is different in males or females, and somehow drives this result. We could plot that blood glucose and sex together to explore this. But maybe we can consider all three variables together.

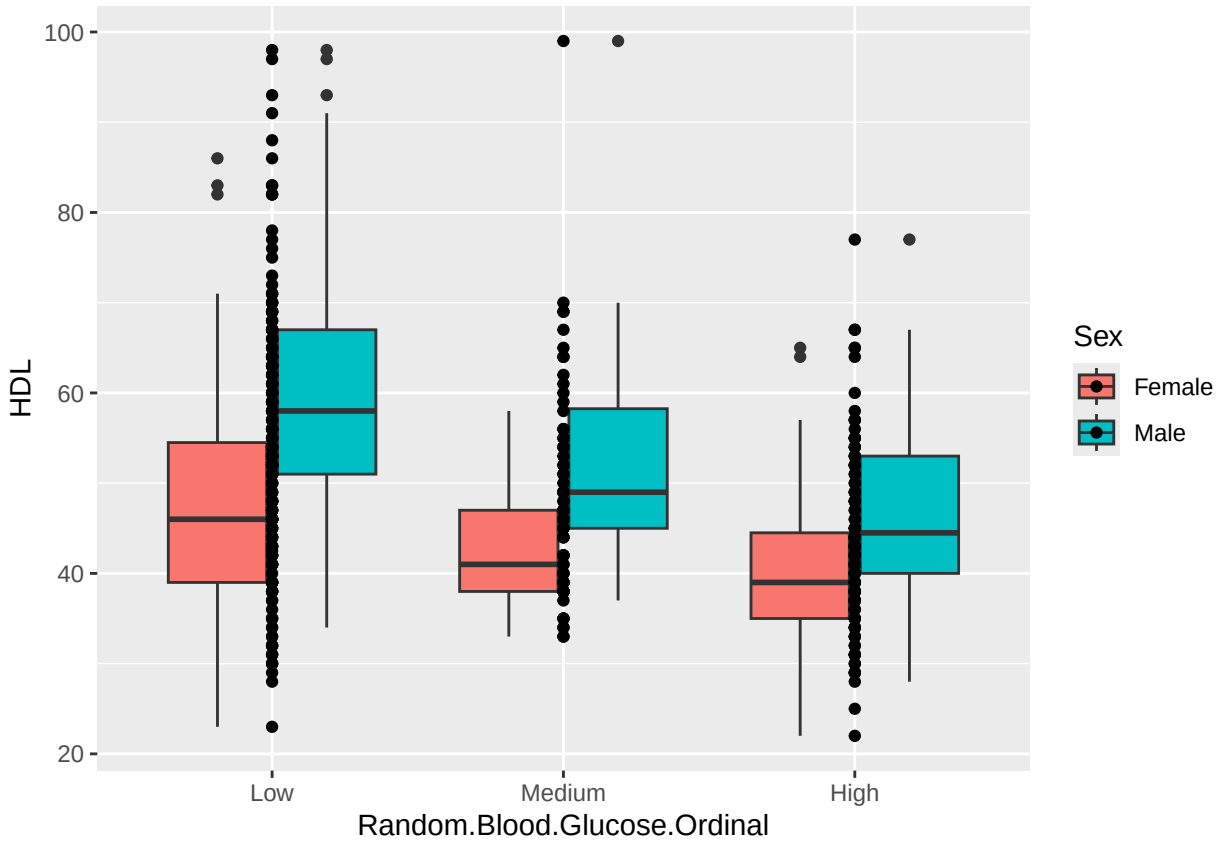
```
ggplot(my_data, aes(x=Random.Blood.Glucose.Ordinal, y=HDL, fill=Sex)) +
  geom_boxplot()
```



Here we pass one additional argument to the aesthetic mapping fuction `aes`. This argument is called `fill` which is interpreted by boxplot to be the color used to fill the boxs.

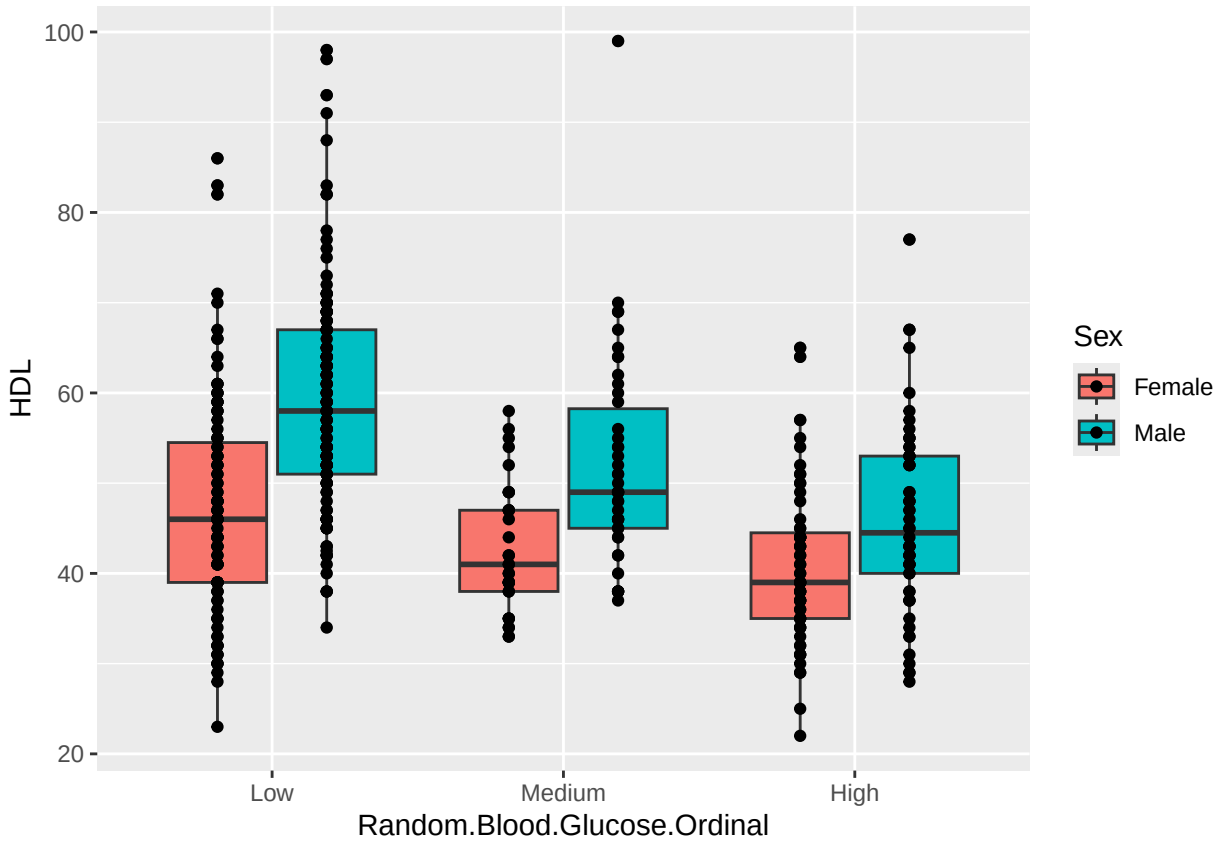
Note I have excluded the points from this plot. Let's try adding them in.

```
ggplot(my_data, aes(x=Random.Blood.Glucose.Ordinal, y=HDL, fill=Sex)) +
  geom_boxplot() +
  geom_point()
```



The points have been added back in but they do not align with the boxplots! The fix for this starts to get a bit complicated and I had to look on stackoverflow for help. But here it is.

```
ggplot(my_data, aes(x=Random.Blood.Glucose.Ordinal, y=HDL, fill=Sex)) +
  geom_boxplot() +
  geom_point(mapping=aes(group=Sex), position=position_dodge(width=0.75))
```



This is complicated but let's break it down a bit. The key issue is that `geom_boxplot` will "dodge" the boxes when there are two groups which it knows about from `fill`. However, `geom_point` does not have the same behavior so we have to tell it about the groups and to dodge. To tell it about the groups we pass the `aes(group=Sex)` argument to `geom_point` so it knows we want our points grouped. Next we need to tell it how to place the points in the group. This is where we pass the `position` argument the `position_dodge(width=0.75)` object. This tells ggplot to move the points away from the center by the amount 0.75 specified using the `width` argument.

> This is the most complicated example we will see. In practice if you hit an issue like this make use of the online resources like stackoverflow. Most problems you have will be answered there, but getting the right search words is key. For reference I searched "ggplot point and boxplot position by fill" in Google.

Plot style

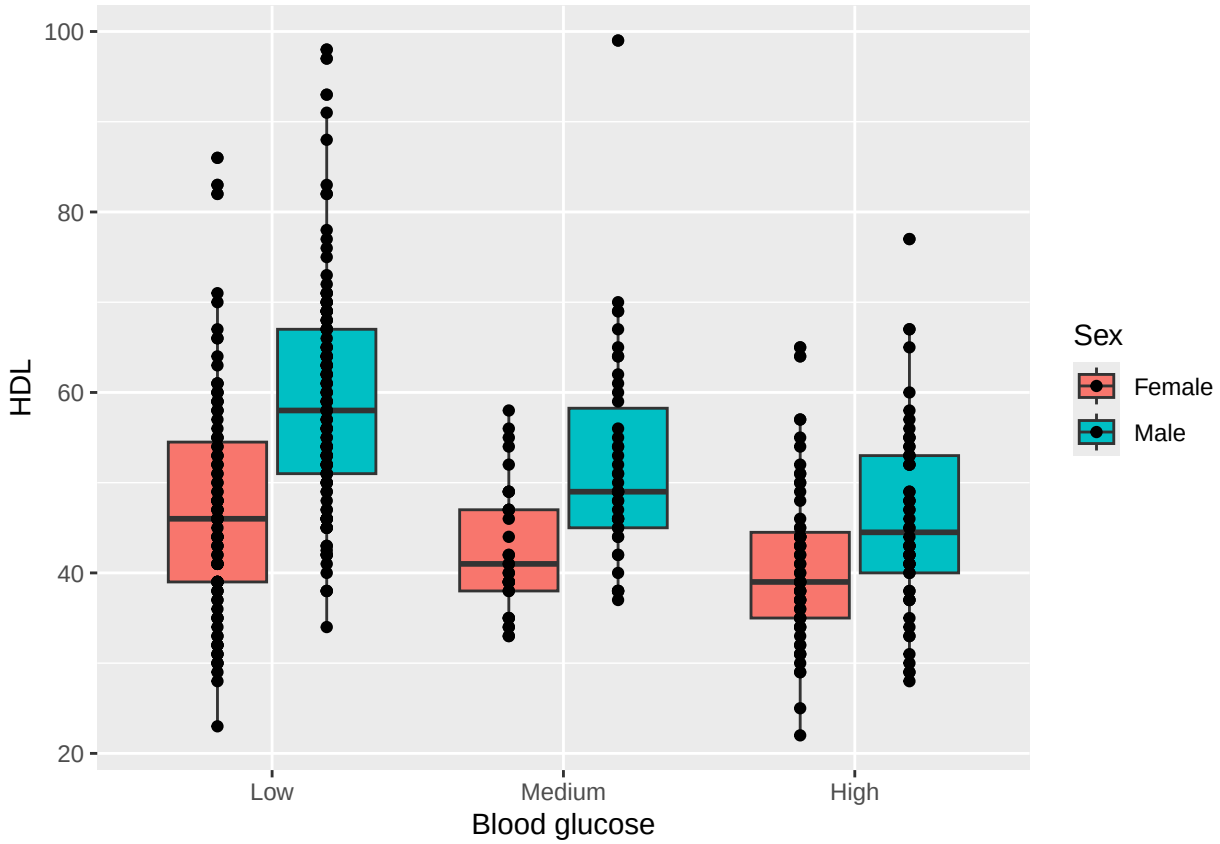
We can now produce some fairly complicated plots. But there are some issues we need to address to get to publication quality plots.

The first issue is the x-axis label is just the name of our column "Random.Blood.Glucose.Ordinal". This has several issues. - It would not typically be the style of a journal to have the works separated by a period. - Having ordinal in the name is not great.

By default R will make the axis labels match the corresponding column labels. So we could fix this by changing the name column in our dataset. But this is a bit of an ugly fix, especially when you need spaces between works. A better solutions is to the `labs` function.

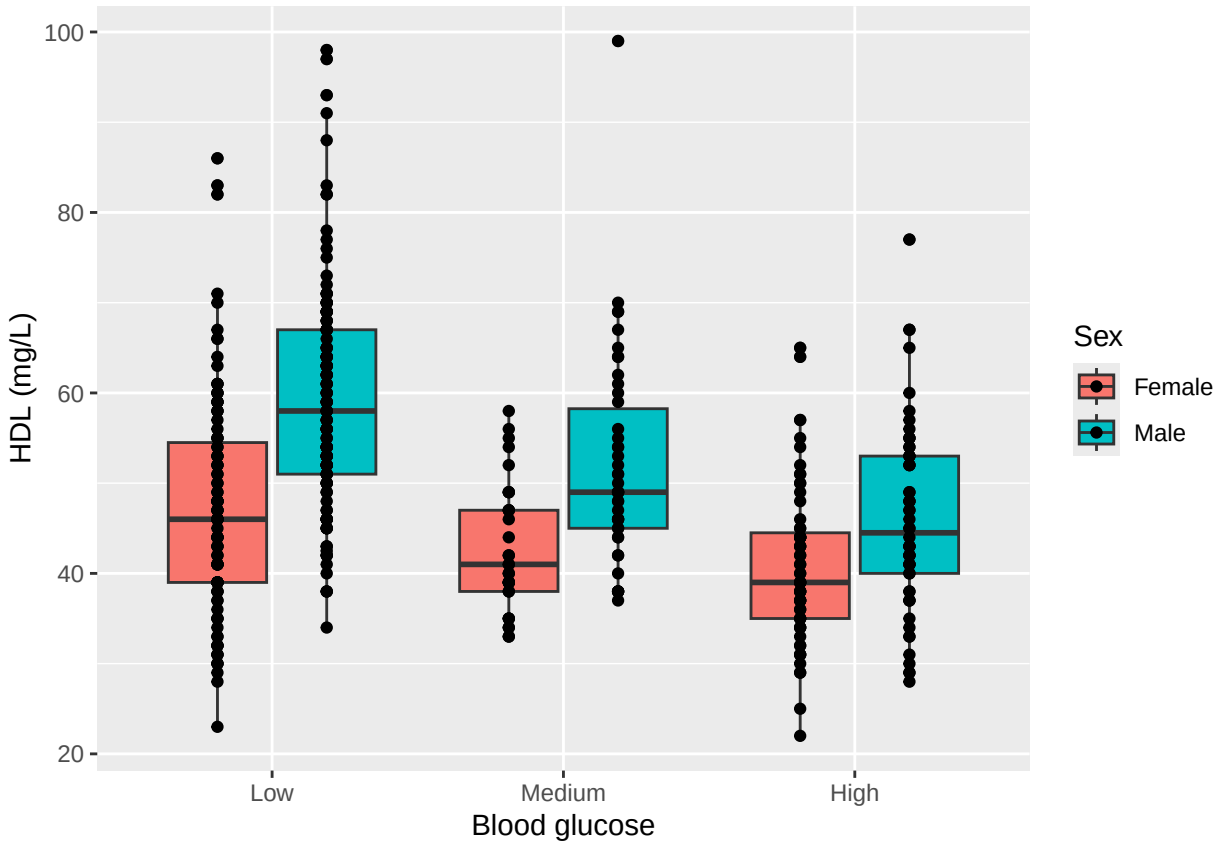
```
ggplot(my_data, aes(x=Random.Blood.Glucose.Ordinal, y=HDL, fill=Sex)) +
  geom_boxplot() +
  geom_point(mapping=aes(group=Sex), position=position_dodge(width=0.75)) +
```

```
  labs(x="Blood glucose")
```



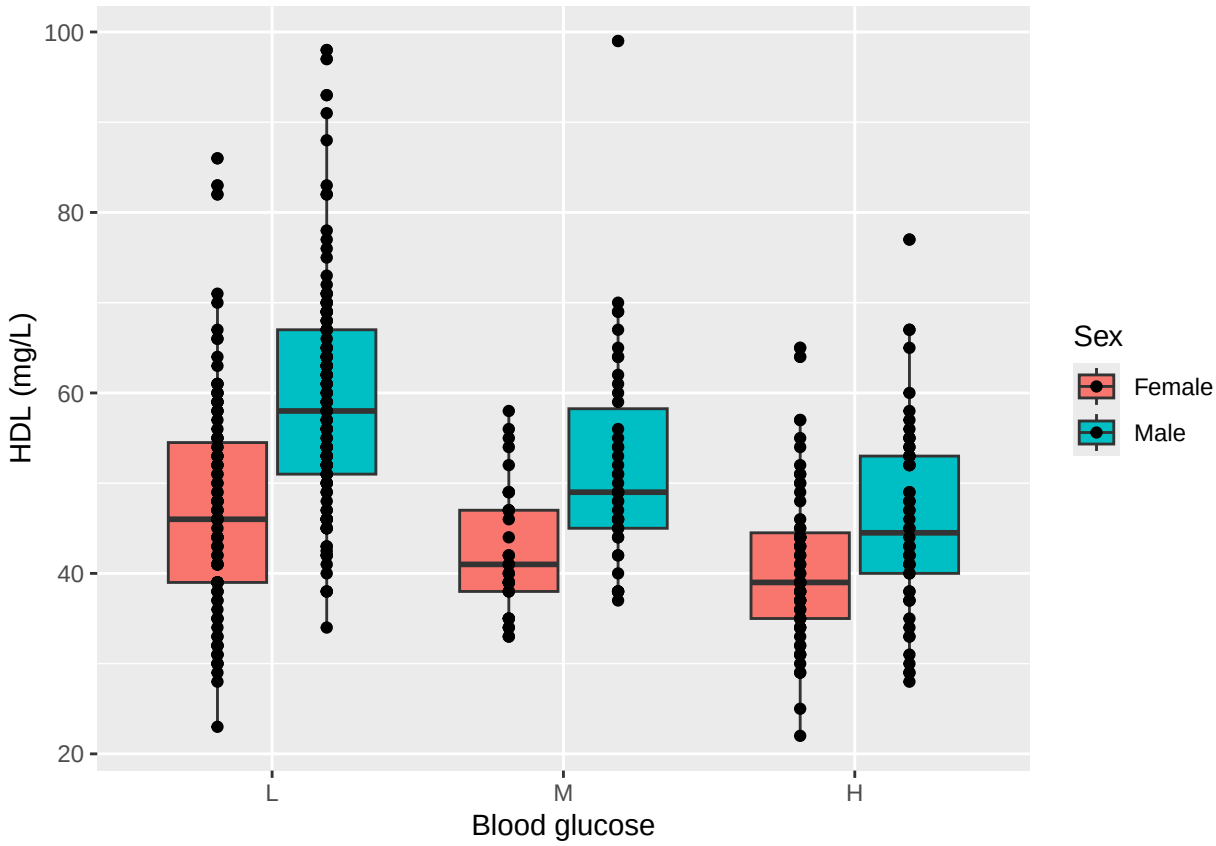
Now we should really have units for HDL on the y-axis. I am going to make up that the correct units are milligrams per litre i.e. mg/L.

```
ggplot(my_data, aes(x=Random.Blood.Glucose.Ordinal, y=HDL, fill=Sex)) +
  geom_boxplot() +
  geom_point(mapping=aes(group=Sex), position=position_dodge(width=0.75)) +
  labs(x="Blood glucose", y="HDL (mg/L)")
```



We may also want to shorten the labels on the x-axis to L, M, H instead of the full names. This can be done with the `x_labels_discrete` function

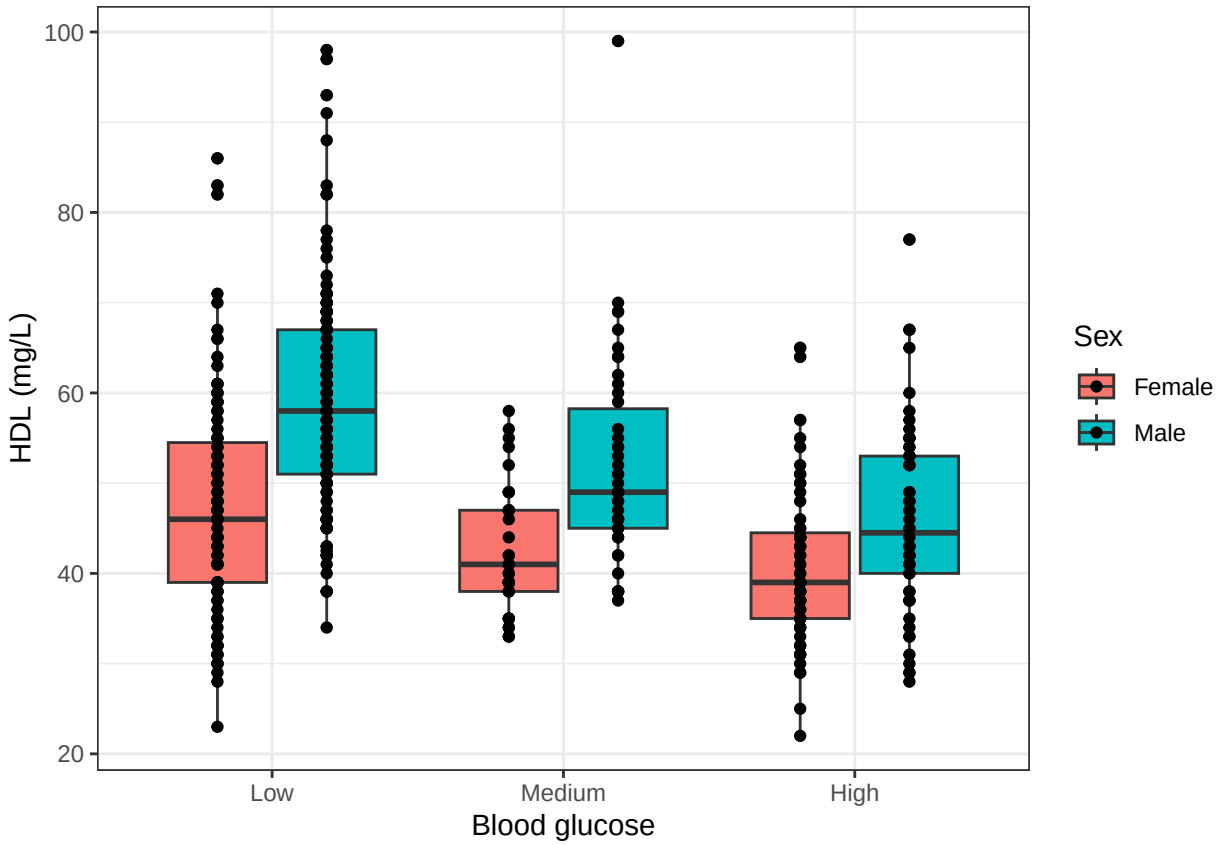
```
ggplot(my_data, aes(x=Random.Blood.Glucose.Ordinal, y=HDL, fill=Sex)) +
  geom_boxplot() +
  geom_point(mapping=aes(group=Sex), position=position_dodge(width=0.75)) +
  labs(x="Blood glucose", y="HDL (mg/L)") +
  scale_x_discrete(labels=c("Low" = "L", "Medium" = "M", "High" = "H"))
```



Themes

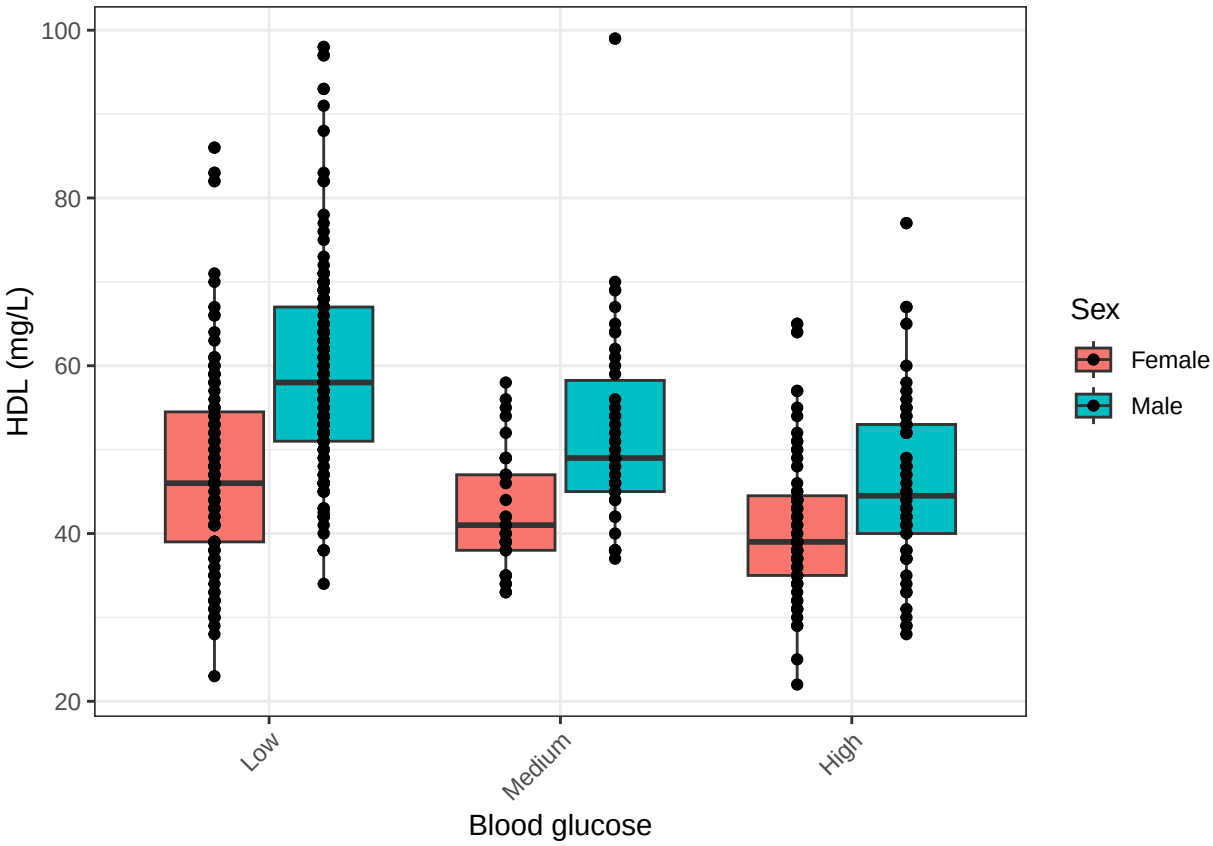
Now we can also change the plot styling using themes in ggplot. There are several built in themes. Here I will use the `bw` theme.

```
ggplot(my_data, aes(x=Random.Blood.Glucose.Ordinal, y=HDL, fill=Sex)) +
  geom_boxplot() +
  geom_point(mapping=aes(group=Sex), position=position_dodge(width=0.75)) +
  labs(x="Blood glucose", y="HDL (mg/L)") +
  theme_bw()
```



Plot style can be customized using the `theme` function. For example, suppose we want to rotate the x-labels to 45 degrees.

```
ggplot(my_data, aes(x=Random.Blood.Glucose.Ordinal, y=HDL, fill=Sex)) +
  geom_boxplot() +
  geom_point(mapping=aes(group=Sex), position=position_dodge(width=0.75)) +
  labs(x="Blood glucose", y="HDL (mg/L)") +
  theme_bw() +
  theme(axis.text.x=element_text(angle=45, hjust=1))
```



Conclusion

Hopefully this tutorial gives you enough concepts to get started plotting with ggplot and R. There is lots of things we did not cover. A good starting point for figuring out how to do things is the ggplot cheatsheet. There are also many examples online. Stackoverflow can be particularly helpful if you have specific tasks you are trying to accomplish.