Week 1 Module 7: Exporting Graphs (Bonus)

Let's clear the environment:

```
rm( list = ls() )
```

Module Overview and Learning Objectives

In this module, we'll learn how to export graphs into standard file formats.

Section 1: Introduction

OK, now you've created your data visualization masterpiece using base R graphics.

But how do you then display this to your audience?

Do they also have to install R and RStudio, and load in the R notebook?

That seems impracticable.

Of course, you can always knit to a PDF and distribute that, but you might want to present your work in something other than an R notebook.

In this module, we're going to explore how to export an R graph to an external graphics file, which can then be used in applications such as Microsoft Word, Microsoft Powerpoint, Adobe Illustrator, Adobe InDesign, or Affinity Designer.

Section 2: Graphics File Formats

There are many different file formats for graphics objects.

R can export to 4 different file formats:

- PNG
- Jpeg
- Bitmap
- Tiff

For most purposes, the best format for exporting R graphs is PNG, but if you want to you can use the others as well.

The procedure is more or less the same for all four supported formats, although sometimes there are some special options that are unique to that format.

The first step in exporting a graph is to create a graphics device, and the function to create a PNG file is named `png()`.

Once you've created a graphics device, then you can use all the methods that we've studied to draw your graph.

Note that when you open a graphics device, all graphics will be drawn on that, so you *won't* see anything in the R notebook or in the console.

When you're all done, you should close the graphics device by calling the `dev.off()` function, which takes no input arguments.

In RStudio, the `dev.off()` function will be called automatically when a code chunk finishes, so technically you don't *have* to do this, but I think it's best practice to explicitly make the call in your code.

Section 3: Worked Example

Let's see a simple example of how this works.

First, let's write some code to create a simple graph:

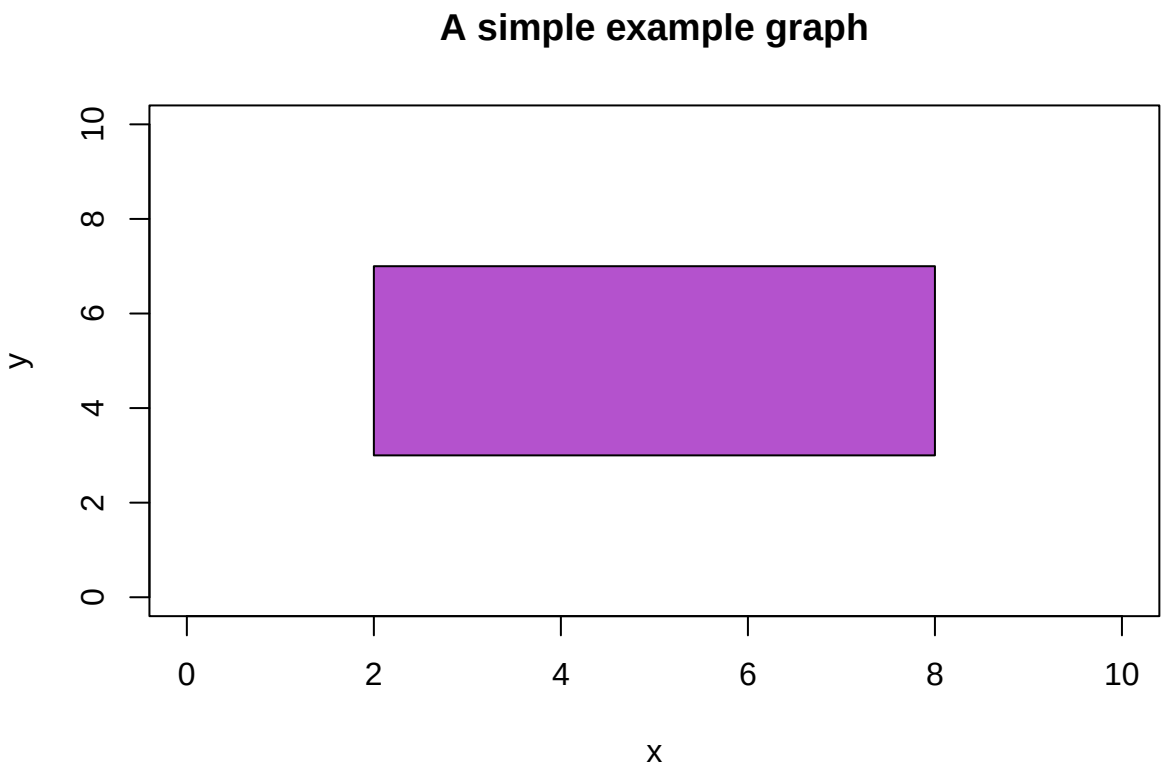
```
# Example 1: Creating a simple graph

# First, let's create an empty plotting region
# with no data

plot(
    x = NULL,
    xlim = c(0, 10),
    ylim = c(0, 10),
    main = "A simple example graph",
    xlab = "x",
    ylab = "y"
)

# Now let's draw a rectangle in the middle of
# this plotting region

polygon(
    x = c(2, 2, 8, 8),
    y = c(3, 7, 7, 3),
    col = "mediumorchid3"
)
```



Now we'll create the same graph, but this time we'll export it to a PNG graphics file instead of displaying in RStudio.

```
# Example 2: Exporting the simple graph

# First, let's open the graphics device

png()


# Now we can use all the code from the
# previous code chunk

plot(
    x = NULL,
    xlim = c(0, 10),
    ylim = c(0, 10),
    main = "A simple example graph",
    xlab = "x",
    ylab = "y"
)

# Now let's draw a rectangle in the middle of
# this plotting region

polygon(
```

```
    x = c(2, 2, 8, 8),
    y = c(3, 7, 7, 3),
    col = "mediumorchid3"
)

# Once we've finished with our graph,
# we should close the device.

dev.off()
```

```
## pdf
##   2
```

If we now look in the folder for this notebook, we'll see a PNG graphics file called "Rplot001.png", which will contain our simple graphic.

Notice that when you run this code, no graph is displayed in the R notebook.

That's because all our graphing commands have now been written to the graphics device.

As a workflow suggestion, I recommend developing the graph by first working in RStudio to get all the details of the image correct, and only then adding in code to export to a graphics file.

Section 4: Naming the file

If we don't supply a filename for the exported graphics, R will automatically generate one for us.

As a general rule, computer-generated filenames are not particularly informative, and in our example this is definitely the case, since the filename is just "Rplot001.png".

Instead, we can explicitly specify the output filename, by using the `filename` option:

```
# Example 3: Specifying the filename

# First, let's open the graphics device

png(
    filename = "Simple Graph.png"
)


# Now we can use all the code from the
# previous code chunk

plot(
    x = NULL,
    xlim = c(0, 10),
    ylim = c(0, 10),
    main = "A simple example graph",
    xlab = "x",
    ylab = "y"
)

# Now let's draw a rectangle in the middle of
```

```r
# this plotting region

polygon(
    x = c(2, 2, 8, 8),
    y = c(3, 7, 7, 3),
    col = "mediumorchid3"
)

# Once we've finished with our graph,
# we should close the device.

dev.off()
```

```
## pdf
##   2
```

After you run this code, you should look in the folder for this R notebook, and now you should see a file called "Simple Graph.png", containing our graph.

By the way, if you're going to explicitly specify the filename, you should be sure to include the extension ".png".

R will write out the file correctly, regardless of whether or not you use the ".png" extension.

But if you don't include the ".png" extension, some operating systems (e.g. Windows) or applications won't recognize that this is a PNG graphics file.

Even if this isn't an issue for you with your operating system or application, I think it's still a best practice to include the extension, so that your code will produce the correct result on all platforms.

Section 5: Aspect Ratio

If you look at the output file, you should notice that the actual image is a square.

When R creates a PNG graphics device, it sets the width and height of the output file.

By default, R creates a file with a width of 480 pixels and a height of 480 pixels.

Since the width and height are the same number of pixels, the graph is a square.

Often, it's desirable to have a graph with a different shape.

The ratio of the width to the height is called the *aspect ratio*, and there are two common aspect ratios in use:

- A 16:9 ratio.
- A 4:3 ratio (which is the same as 16:12).

Personally, I always use a 16:9 aspect ratio.

This is the standard for wide-screen TVs, most movies, YouTube and Vimeo videos, and Powerpoint slides.

It's also often desirable to increase the resolution, which is the number of pixels.

The standard R default value of 480 pixels for height and width is very low, and typically you would want to use a higher resolution.

For a 16:9 aspect ratio, I recommend a width of 1280 pixels and a height of 720 pixels, and in fact in the video world these are the dimensions of HD video, which is the standard for Amazon and Netflix streaming.

For a 4:3 aspect ratio, I recommend a width of 800 pixels and a height of 600 pixels, or a width of 1280 pixels and a height of 960 pixels.

The more pixels, the better the resolution, but the size of the file will be corresponingly larger.

After a certain point, your display will not be able to handle the increased resolution, and you won't be able to tell the difference.

Let's create another PNG file, this time with a high-resolution 16:9 HD aspect ratio:

```
# Example 4: Modifying the aspect ratio and resolution

# First, let's open the graphics device

png(
    filename = "HD Graph.png",
    width = 1280,
    height = 720
)


# Now we can use all the code from the
# previous code chunk

plot(
    x = NULL,
    xlim = c(0, 10),
    ylim = c(0, 10),
    main = "A simple example graph",
    xlab = "x",
    ylab = "y"
)

# Now let's draw a rectangle in the middle of
# this plotting region

polygon(
    x = c(2, 2, 8, 8),
    y = c(3, 7, 7, 3),
    col = "mediumorchid3"
)

# Once we've finished with our graph,
# we should close the device.

dev.off()
```

```
## pdf
##   2
```

Now when we look in the file folder we'll see a new graphics file with the desired dimensions.

Section 6: Adjusting the Point Size

There's a problem with the graph that we just drew.

If you open this graph in a graphics display, you'll see that while the overall ratios are all correct, the text seems to be very small.

What's happening here is that when you increase the dimensions of the graph, R doesn't adjust the size of the text to compensate for this.

Thus, although the dimensions of the graph are much larger, the size of the text remains fixed, and so the text becomes smaller relative to the overall graph.

To compensate for this, you can use the `pointsize` option, which will adjust the size of the text.

The appropriate value for the `pointsize` option will depend on the specific values of the height and width, and you might have to do some experimenting to get something that looks good.

However, I can tell you that when I use a 1280 x 720 HD resolution, I find that a `pointsize` value of 24 seems to work really well.

Let's try this:

```
# Example 5: Modifying the pointsize option

# First, let's open the graphics device

png(
    filename = "HD Graph Improved.png",
    width = 1280,
    height = 720,
    pointsize = 24
)


# Now we can use all the code from the
# previous code chunk

plot(
    x = NULL,
    xlim = c(0, 10),
    ylim = c(0, 10),
    main = "A simple example graph",
    xlab = "x",
    ylab = "y"
)

# Now let's draw a rectangle in the middle of
# this plotting region

polygon(
    x = c(2, 2, 8, 8),
    y = c(3, 7, 7, 3),
    col = "mediumorchid3"
)

# Once we've finished with our graph,
```

```
# we should close the device.

dev.off()
```

```
## pdf
##   2
```

The graph won't necessarily look *exactly* like what you see when you write out to the console, so you have to be willing to experiment and try some different values for the optional parameters in order to get it to look reasonably good.

Section 7: Importing to a Graphics Program

Once you've created a graphics file, you can then use this just like any other.

In particular, you can then import this file into a graphics program, and then perform further editing.

Let's try this out with Microsoft Powerpoint.

I'm using the Microsoft Office Home and Business 2016 suite, which includes Word, Excel, and Powerpoint.

First, I'll create a new presentation, and then I'll create a new blank slide.

By default, Powerpoint presentations use a 16:9 aspect ratio, although you can change this if you want.

I then go to the Insert tab, select Pictures, and choose to insert a picture from this device (i.e. my local hard drive).

I then navigate to the file folder containing my graphics file and click on it.

The graphics file will load, and if we've used a resolution of 1280 x 720 it will perfectly fit the Powerpoint slide frame.

You can then resize the graph if you want it to be smaller.

You can also layer graphics objects such as text boxes or shapes on top of the PNG graph.

This is a great way to combine the power of R with the rich set of tools in programs such as Powerpoint, Adobe Illustrator, or Affinity Designer.

Section 8: Exporting to a PDF

You can also export your graphics image to a PDF file by using the `pdf()` function.

Because the PDF format is somewhat different than standard graphics formats such as PNG or JPEG, the `pdf()` function has many different options.

You can still set the height and width, but these are now specified using inches, not pixels.

There is also a `pointsize` option, and this is similar to what we saw with the `png()` function.

In my experience, the PDF exporting process is a little weird, and objects such as legends are positioned slightly differently than what you were expecting.

I don't really understand why this is, and the only workaround that I've found is to create a dedicated code chunk just for the graphic, export it, and then look at the output image and make adjustments.

The result will be a graph that looks weird if you render it within RStudio, but it looks OK as a PDF.

In my projects, I don't really use the PDF format for graphics objects all that much, and instead I typically find it most useful to work with PNG graphics.

In particular, Powerpoint cannot import PDFs as graphics images, and many video editors cannot work with them either.

However, it all depends on the specifics of your project, and you might that the PDF format is very useful for you, so I don't want to rule this out completely.

```
# Example 6: Exporting to a PDF file

# First, let's open the graphics device

pdf(
    file = "PDF graph.pdf",
    width = 6.4,
    height = 3.6
)


# Now we can use all the code from the
# previous code chunk

plot(
    x = NULL,
    xlim = c(0, 10),
    ylim = c(0, 10),
    main = "A simple example graph",
    xlab = "x",
    ylab = "y"
)

# Now let's draw a rectangle in the middle of
# this plotting region

polygon(
    x = c(2, 2, 8, 8),
    y = c(3, 7, 7, 3),
    col = "mediumorchid3"
)

# Once we've finished with our graph,
# we should close the device.

dev.off()
```

```
## pdf
##   2
```