Lab 5 – Customized Scientific Graphs
Part III

Questions? montwe@ualberta.ca or isaacren@ualberta.ca

Here, we continue creating scientific graphs. This lab will help you apply concepts discussed in class and will help completing the assignments or course project.

5.20 Facet function for multiple plots in one figure

Sometimes it makes sense to show multiple plots in one figure. Ggplot2 has a nice function to archive this with little code.

We will try this on the tree ring dataset (download from website):

```
dat1 = read.csv("Tree-rings.csv")
head(dat1)
```

First we summarize the data with `ddply()` from plyr package again, but this time by year and site:

```
install.packages("plyr")
library(plyr)
dat1_sum =ddply(dat1, .(YEAR,SITE),summarise,
                mRW=mean(Ringwidth,na.rm=T), sdRW=sd(Ringwidth,na.rm=T))
head(dat1_sum)
```

Next, we plot the data:

```
install.packages("ggplot2")
library(ggplot2)

Fig1 = ggplot(dat1_sum, aes(x=YEAR,y=mRW, fill=SITE))+
  geom_line()+
  geom_ribbon(aes(ymin = mRW - sdRW, ymax = mRW + sdRW),alpha=0.5)+
  theme_minimal()+
  theme(legend.position = "none")
Fig1
```

As you can see, we now have two curves because we specified fill=SITE. However, the curves do not differ much, and it is hard to distinguish them. Here, we can use the function `facet_grids()` function, which enables plotting in different panels:

```
Fig1 = ggplot(dat1_sum, aes(x=YEAR,y=mRW, fill=SITE))+
  geom_line()+
  geom_ribbon(aes(ymin = mRW - sdRW, ymax = mRW + sdRW),alpha=0.5)+
  theme_minimal()+
  theme(legend.position = "none")+
  facet_grid(SITE ~ .)
Fig1
```

The command “SITE~.” splits the figure into two panels horizontally. Now, if we turn this command around, you should see two plots beside each other vertically.

```
Fig1 = ggplot(dat1_sum, aes(x=YEAR,y=mRW, fill=SITE))+
  geom_line()+
  geom_ribbon(aes(ymin = mRW - sdRW, ymax = mRW + sdRW),alpha=0.5)+
  theme_minimal()+
  theme(legend.position = "none")+
  facet_grid(.~SITE)
Fig1
```

5.21 Multi-plot with the cowplot package

Another way to combine plots in one figure is the package cowplot. This package creates plots with multiple rows/columns and aligns the plots efficiently. Start by installing and loading the package:

```
install.packages("cowplot")
library(cowplot)
```

Before we can use the package, we first have to create a few with the iris-dataset:

```
dat2 = iris
```

To plot multiple plots in one figure using cowplot, each figure must first be generated and saved as an object as per usual – these are then combined in a later step using the `plot_grid()` function.

Let’s start by plotting the iris dataset (dat2) and saving it to an object (Fig1), which will include a legend. Just as a heads up: Per default, ggplot2 creates a legend with a white background. This may cover up your data point if the legend is placed inside the plot. This can be changed with the `theme()` function, see below:

```
Fig1 = ggplot(dat2, aes(x=Petal.Width,y=Petal.Length, col=Species))+
  geom_point()+
  labs(x="Petal width (cm)",y="Petal length (cm)")+
  theme_bw()+
  stat_smooth(method="lm")+
  theme(legend.position = c(0.2,0.8), legend.background=
      element_rect(fill="transparent",colour=NA))
 Fig1
```

When developing a multi-plot figure with panels representing data from the same datasets, repeating legends in each figure is redundant. Therefore, we switch off the legend for the remaining plots, which will lead to less clutter and a higher data-to-ink ratio in the final product.

Please run the following code to plot the next three figures (which will be turned into panels in the multi-plot):

```
Fig2 = ggplot(dat2, aes(x=Sepal.Width,y=Sepal.Length, col=Species))+
  geom_point()+
  labs(x="Sepal width (cm)",y="Sepal length (cm)")+
  theme_bw()+
  stat_smooth(method="lm")+
  theme(legend.position = "none")
Fig2
```

```
Fig3 = ggplot(dat2,aes(x=Species,y=Petal.Width,fill=Species))+
  geom_boxplot()+
  labs(y="Petal width (cm)")+
  theme_bw()+
  theme(legend.position = "none")
Fig3

Fig4 = ggplot(dat2,aes(x=Species,y=Sepal.Width,fill=Species))+
  geom_boxplot()+
  labs(y="Sepal width (cm)")+
  theme_bw()+
  theme(legend.position = "none")
Fig4
```

You probably noticed that we now have to give each plot a unique name (i.e., Fig1, Fig2, Fig3, Fig4). That is because we don't want to overwrite the previous plot – each figure needs to be its own object for the next line of code to work:

```
plot_grid(Fig1, Fig2, Fig3,Fig4, align='hv', nrow=1, labels=c("A", "B",
"C","D"))
```

The first few items in the `plot_grid()` function specify the objects (Fig1 – Fig4) to plot as individual panels. The command align= "hv" means that the individual plots should be aligned horizontally and vertically. The code above specifies the panels should be arranged in 1 row through the command "nrow" (which stands for "number of rows"). The "labels" command assigns letters to each panel. The labels can be turned off with labels="", or automated with labels="AUTO".

For two columns and two rows, we have to include "ncol" and type:

```
plot_grid(Fig1, Fig2, Fig3,Fig4, align='hv', nrow=2,ncol=2,
               labels=c("A", "B", "C","D"))
```

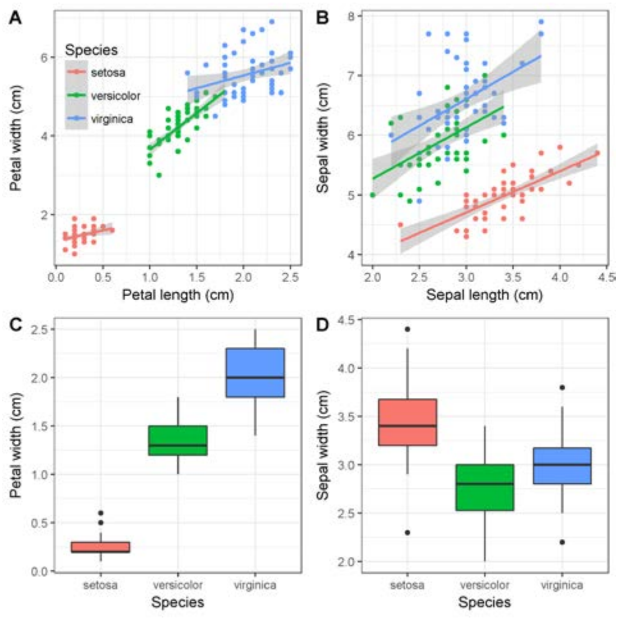
5.22 Exporting graphics for manual formatting

To save the file, first save the plot in an object and then export it with the `ggsave()` function.

```
p1 = plot_grid(Fig1, Fig2, Fig3,Fig4, nrow=2,ncol=2,align='hv',
               labels=c("A", "B", "C","D"))

ggsave(p1, file="Plot1.png", scale=2.0, width=8.1, height=8.1, units="cm",
dpi=300)
```

Here, I defined the width and height of the plot as 8.1 cm, which would be a good size for a 1 column figure in a scientific journal article. We use scale=2 because this scaling gives us good font sizes and line widths. Open up word and insert the picture (the .png file we just saved) and reduce the size to 50%. dpi stands for dots per inch. In general, 150 to 300 dpi is good for printing and the web.



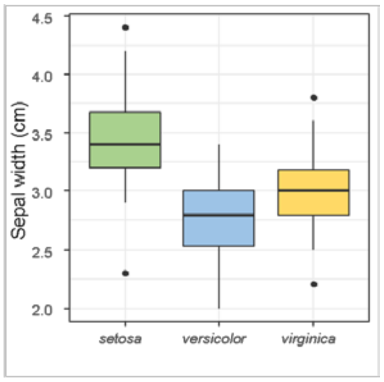
This method of defining the size and scale of your plot is efficient for most applications. If you need a larger font, line and point size (e.g. figures for a website), the scale command can be adjusted with scale=1.8 or 1.6.

If the plot is saved as a vector graphic, we can do some modifications after exporting the image from R. The following code will save Fig 4 as an .emf file. This file type can be read by Powerpoint.

```
ggsave(Fig4,file="Fig4.emf", scale=2.0, width=8.1,height=8.1,units="cm")
```

Open Powerpoint and insert the Fig4.emf file into an empty slide. Then, right-click on the figure and go to Group > Ungroup (click “yes”). Repeat this a few times until you see that all single elements are selectable.

Now you can modify the figure. For example, the species names should be italic, the colors can be changed, etc. When you are done, select all elements, right click, go to group > Regroup. Now you can export this figure again by right clicking on the figure and selecting “Save As Picture” to export the graphic.



While this is a good way to fix minor things, it is always better to make as many changes as possible directly in R. Code in R is easily reproducible while such manual formatting changes are not.

Editing in Powerpoint also has limitations. A better way is to save your plots as .pdf files and edit those in professional vector editing software. The most common program is Adobe Illustrator, but there are also free programs available. Of those, Inkscape is the best alternative. Depending on time, we will cover editing in this program later in the course.

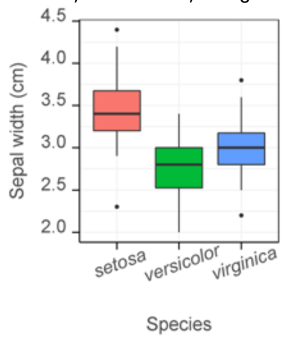
If you have time: Font sizes, line widths and virtually anything about the appearance of the plot can be changed with the `theme()` function:

```
Fig4 = ggplot(dat2,aes(x=Species,y=Sepal.Width,fill=Species))+
  geom_boxplot(size=0.25, outlier.size=0.25)+
  labs(y="Sepal width (cm)")+
  theme_bw()+
  theme(legend.position = "none")+
  theme(axis.ticks= element_line(size = 0.25, colour = "black"),
        panel.grid.major =  element_line(colour = "grey95", size = 0.25),
        panel.grid.minor =  element_line(colour = "grey95", size = 0.25),
        axis.line = element_line(size = 0.25, colour = "black"),
        panel.border =      element_rect(fill = NA, colour="black",size=0.25),
         axis.text.x = element_text(colour="grey30",size=7,angle=20,
         face="italic"),
         axis.text.y = element_text(colour="grey30",size=7,angle=0,hjust=1,
        vjust=0, face="plain"),
        axis.title.x =   element_text(colour="grey30",size=7,angle=0,hjust=.5,vjust=0,
       face="plain"),
        axis.title.y = element_text(colour="grey30",
       size=7,angle=90,hjust=.5,vjust=.5,face="plain"))
Fig4
```

This might seem line a pile of code, but I want to illustrate that you can change anything about your plot manually. If you type `?theme`, the documentation will show you all the available arguments for the function. For example, I wanted to change the angle of the x-axis labels to avoid over-plotting. Therefore, I typed in the argument “axis.text.x” and the function “element_text” in which you can specify – among other things – the angle in degrees. I chose 20 degrees in this case (see bold code). We also have to adjust the size of the geom_boxplot (see above, also in bold).

Now we can save the plot without scaling. Note that I increased the dpi to 600 to get a crisper result.

```
ggsave(Fig4,file="Fig4_v2.png", scale=1.0, width=4,height=4.5,units="cm",dpi=600)
```


Sepal width (cm)
setosa
versicolor
virginica
Species