Lab 5 – Customized Scientific Graphs
Part II

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.

5.12 Adding functions to scatter plots with stat_smooth

We will need two packages for this lab, and it is good practice to load packages at the top of your script.

```
library(ggplot2)
library(plyr)
library(tidyr)
```

We start with the same data set as in last week's lab. You can download it from the website.

```
dat1=read.csv("Scatter.csv")
```

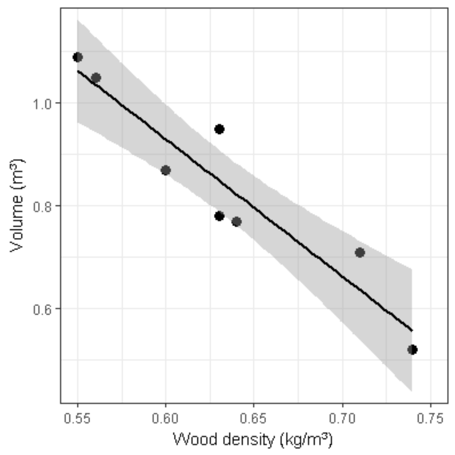
Do you know what each of the following lines of code does?

```
Fig1 = ggplot(dat1, aes(x=DENSITY,y=VOL))+
  geom_point(size=3)+
  labs(x="Wood density (kg/m³)",y="Volume (m³)")+
  theme_bw()+
  coord_cartesian(ylim=c(0.45,1.15),xlim=c(.55,0.75))
Fig1
```

We will now add a regression line with confidence intervals to the scatter plot. Regression will be covered in an upcoming lecture, but for now, we will focus on the technical implementation. Please add the line in bold to your script:

```
Fig1 = ggplot(dat1, aes(x=DENSITY,y=VOL))+
  geom_point(size=3)+
  labs(x="Wood density (kg/m³)",y="Volume (m³)")+
  theme_bw()+
  coord_cartesian(ylim=c(0.45,1.15),xlim=c(.55,0.75))+
  stat_smooth(method="lm", col="black")
Fig1
```

There are many methods available for the `stat_smooth()` function, and you can even create your own with the `formula =` [insert your command]. For now, we have used a linear model with the `lm()` function.



The black line is the regression line, and in grey we see the confidence interval of the regression. Often, the regression is shown in combination with an R^2-value. This number indicates how well the regression line explains the underlying data points. High R^2 values indicate good explanation, low values poor explanation.

Before we can add an R^2 value to the plot, we have to calculate it first. This can be done with the `lm()` function of R. The "lm" stands for linear model. We save the model in a new object, called m1:

```
m1 = lm(dat1$DENSITY~dat1$VOL)
```

The summary function gives access to a bunch of statistics. For now, look for the R squared value:

```
summary(m1)
```

Using the $-sign, we can get direct access to the R^2 value:

```
r2 = summary(lm(dat1$DENSITY~dat1$VOL))$r.squared
r2
```

As you can see, the number has many digits. The round() function takes care of this. Indicate the desired number of decimals under digits=…

```
r2 = round(r2, digits=2)
r2
```

Great. Now we can add this number to our plot with the annotate-function. Note, the ASCII code for creating a superscript two (e.g. R^2) is ALT+253 (almost any special character can be typed through ASCII code – you can google how to enter é, for example).

```
Fig1 = ggplot(dat1, aes(x=DENSITY,y=VOL))+
  geom_point(size=3)+
  labs(x="Wood density (kg/m³)",y="Volume (m³)")+
  theme_bw()+
  coord_cartesian(ylim=c(0.45,1.15),xlim=c(.55,0.75))+
  stat_smooth(method="lm",col="black")+
  annotate("text", x = 0.7, y = 1.1, label = paste("R²=",r2,  sep=""))
Fig1
```

Note that we use the `paste()` function here. This function let's you piece together strings of text, separated with the `sep=` command. Instead of "", try "_". You could also use the annotate function to add the equation of the regression to your plot.

5.13 Multivariate bubble plots

Adding additional dimensions to scatterplot can produce "high density" figures that look nice and allow the reader to learn a lot about a dataset. Multi-variate plots are usually great ways to create a high data-to-ink ratio. Here, we will use colour to define ecosystem, shapes to denote species and the size of the points will correspond to age – that is a lot of information shown in one graph. We define additional variables (each it's own column) using the `aes()` function, in which you specify the size, col and shape. If you want to make sure that the dots don't get plotted too small or too big, the `scale_continuous()` function can be added with a defined range.

```
Fig1 = ggplot(dat1, aes(x=DENSITY, y=VOL, size=AGE, col=ECOSYS,shape=SPEC))+
  geom_point()+
  labs(x="Wood density (kg/m³)",y="Volume (m³)")+
  theme_bw()+
  theme(legend.position = "bottom")+
  coord_cartesian(ylim=c(0.45,1.15),xlim=c(.55,0.75)) +
  scale_size_continuous(range = c(2,6))
Fig1
```

The legend is, as always, created automatically.

5.14 Color palettes

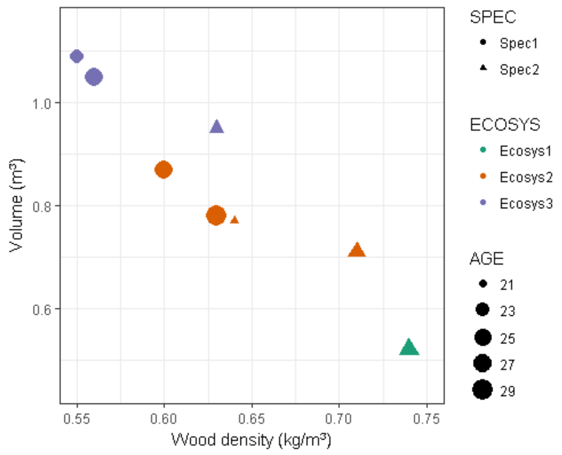
In the previous lab, we already worked on customizing colors. A neat way to do this quickly are the `scale_color_brewer()` and `scale_fill_brewer()` functions. Of the two choices, the most appropriate method depends on what kind of aesthetic you want to plot. Remember from the last lab that col defines the color of points without an outline. For points with an outline, col defines the outline color while fill defines the inside color. For color of the default point symbol (16), we need to use the `scale_color_brewer()` function.

```
Fig1 = ggplot(dat1, aes(x=DENSITY,y=VOL,size=AGE,col=ECOSYS,shape=SPEC))+
  geom_point()+
  labs(x="Wood density (kg/m³)",y="Volume (m³)")+
  theme_bw()+
  theme(legend.position = "bottom")+
  coord_cartesian(ylim=c(0.45,1.15),xlim=c(.55,0.75)) +
  scale_size_continuous(range = c(2,6))+
  scale_color_brewer(palette="Dark2")
Fig1
```

There are many different palettes available. The color_brewer functions correspond to those listed on the Color Brewer website: http://colorbrewer2.org. (Check out this website to see what the following means.) The color scheme depends on the structure by the desired scale:

Diverging	Qualitative	Sequential
BrBG, PiYG, PRGn, PuOr, RdBu, RdGy, RdYlBu, RdYlGn, Spectral	Accent, Dark2, Paired, Pastel1, Pastel2, Set1, Set2, Set3	Blues, BuGn, BuPu, GnBu, Greens, Greys, Oranges, OrRd, PuBu, PuBuGn, PuRd, Purples, RdPu, Reds, YlGn, YlGnBu, YlOrBr, YlOrRd

Try a few. Not all are appropriate for all applications. You want good contrast and ideally a color blind friendly color palette.



5.15 Dotplots and Violin plots to show data

Dotplots and violin plots can be alternatives to boxplots and barplots. We will use the iris-dataset to go through some examples.

```
dat2= iris
```

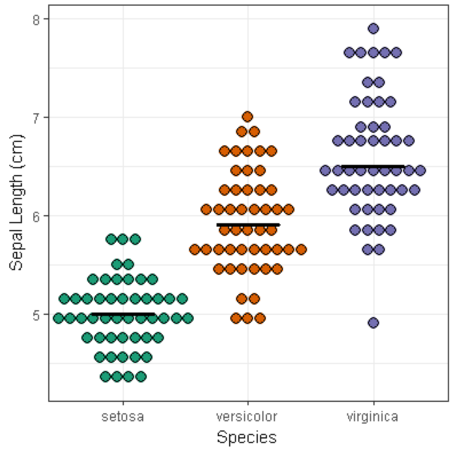
Here, we use the `geom_dotplot` with binaxis ="y" and stackdir= "center". Binaxis="y" means that the data points are plotted on the y-axis. Stackdir="center" means that the datapoints are centered in the middle.

```
Fig1 = ggplot(dat2, aes(x=Species,y=Sepal.Length,fill=Species))+
  geom_dotplot(binaxis = "y", stackdir = "center")+
  labs(x="Species",y="Sepal Length (cm)")+
  theme_bw()+
  theme(legend.position = "none")+
  scale_fill_brewer(palette="Dark2")
Fig1
```

As you can see, we have a plot that somewhat resembles a histogram. We can see the distribution, and every bit of ink is meaningful. However, we don't know where the median or mean is. To fix this, we can add the median with the `stat_summary()` function.

```
Fig1 = ggplot(dat2, aes(x=Species,y=Sepal.Length,fill=Species))+
  geom_dotplot(binaxis = "y", stackdir = "center")+
  labs(x="Species",y="Sepal Length (cm)")+
  theme_bw()+
  theme(legend.position = "none")+
  scale_fill_brewer(palette="Dark2")+
  stat_summary(fun.y = "median", fun.ymin = median, fun.ymax = median,
               geom = "crossbar", width=0.5)
Fig1
```

Because there is no geom that represents a horizontal line, we have to use the "crossbar" geom, but set ymin and ymax to the median, and therefore get a solid line.

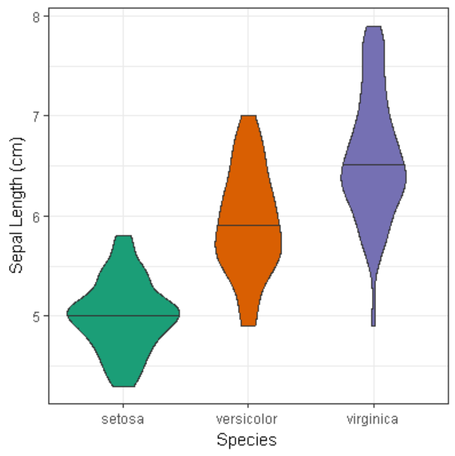


Next, we are going to produce a violin plot, by adding the `geom_violin()` function.

```
Fig1 = ggplot(dat2, aes(x=Species,y=Sepal.Length,fill=Species))+
  geom_violin()+
  labs(x="Species",y="Sepal Length (cm)")+
  theme_bw()+
  theme(legend.position = "none")+
  scale_fill_brewer(palette="Dark2")
Fig1
```

Again, we are faced with the lack of a median or mean. Fortunately, we can tell the `geom_violin` function to draw quantiles. Here we specify the 0.5 quartile, which is the median:

```
Fig1 = ggplot(dat2, aes(x=Species,y=Sepal.Length,fill=Species))+
  geom_violin(draw_quantiles = c( 0.5))+
  labs(x="Species",y="Sepal Length (cm)")+
  theme_bw()+
  theme(legend.position = "none")+
  scale_fill_brewer(palette="Dark2")
Fig1
```



5.16 Dotplots to show multiple means per category

Dotplots can also be useful to summarize a complicated dataset. Let's say we want to show means and standard deviations of the entire iris dataset in one figure. First, we have to convert the wide-data table to long. Remember from an earlier lab what to put into the gather() function: the first item is the dataset we want to convert (here: dat2); the second item is a new column name that we specify – this new column will contain the old column names (here we have called it "FlowerPart"); the third item is a new column name that we specify – this new column will contain the corresponding values (here we have called it "measurement"); and the final item specifies the columns to gather (here we want to convert all columns from Sepal.Length to Petal.Width into long format, so we denote this as "Sepal.Length:PetalWidth").

```
dat2_long = gather(dat2, FlowerPart, measurement, Sepal.Length:Petal.Width)
head(dat2_long)
```

Next, we have to calculate means and standard deviations with `ddply`:

```
dat2_sum =ddply(dat2_long, .(Species,FlowerPart),summarise,
                mMeasurement=mean(measurement,na.rm=T),
                sdMeasurement=sd(measurement,na.rm=T))
head(dat2_sum)
```

Now we can plot this data set:

```
Fig1 <-ggplot(dat2_sum,aes(x=FlowerPart,y=mMeasurement, fill=Species)) +
  geom_dotplot(binaxis="y", stackdir="center") +
  theme_bw() +
  labs(y="Measurement (cm)",x="Flower part")
Fig1
```

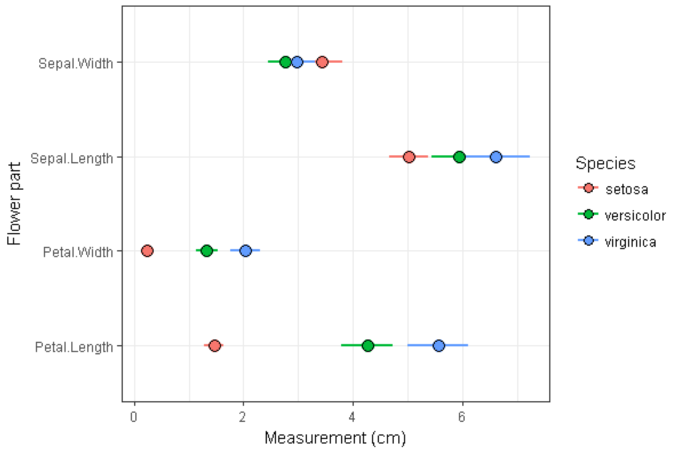
In this case it might be easier to interpret the plot by flipping the coordinate system with `coord_flip()`:

```
Fig1 <-ggplot(dat2_sum,aes(x=FlowerPart,y=mMeasurement, fill=Species)) +
  geom_dotplot(binaxis="y", stackdir="center") +
  theme_bw() +
  coord_flip() +
  labs(y="Measurement (cm)",x="Flower part")
```

```
Fig1
```

Great. Now we only need error bars. These can be added with the `geom_errorbar()` function. As you can see, we subtract the standard deviation from the mean to get a ymin. Similarly, we add the standard deviation to the mean to get a ymax. This therefore gives us the entire range to plot for the error bars. The width= command allows us to set the caps of the error bars to 0 (makes them disappear). You can change this to 0.1 to make them appear. size= 1 makes the lines a bit thicker.

```
Fig1 <-ggplot(dat2_sum,aes(x=FlowerPart,y=mMeasurement, fill=Species)) +
  geom_errorbar(aes(ymin=mMeasurement-sdMeasurement,
        ymax=mMeasurement+sdMeasurement, col=Species), width=.0, size=1)+
  geom_dotplot(binaxis="y", stackdir="center", dotsize=1) +
  coord_flip() +
  theme_bw() +
  labs(y="Measurement (cm)",x="Flower part")
Fig1
```



5.17 Barplots and dotplots for means

As previously discussed, barplots for means are not the preferred way to show group differences in ggplot2. However, it is still possible if this is a standard in your field. We need the `stat_summary()` function:

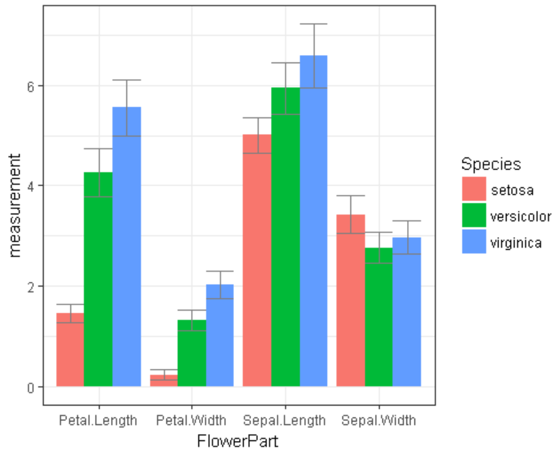
```
Fig1 = ggplot(dat2_long, aes(x=FlowerPart, y=measurement, fill=Species)) +
  stat_summary(fun.y=mean, geom="bar", position=position_dodge())
Fig1
```

What we specify in the code above is that we want the mean of the y-variable and that we want to plot the “geom” as a bar. The “position_dodge” command indicates that we want the bars grouped by FlowerPart. The result looks good, but we are still missing the error bars showing the standard deviation. To get these, we have to add a second `stat_summary function()` to the plot:

```
Fig1 = ggplot(dat2_long, aes(x=FlowerPart, y=measurement,fill=Species)) +
  stat_summary(fun.y=mean, geom="bar",position=position_dodge()) +
  stat_summary(fun.ymin = function(x) mean(x) - sd(x),
               fun.ymax = function(x) mean(x) + sd(x), geom = "errorbar",
               colour = "grey50", position=position_dodge())
Fig1
```

Here, we tell R to apply two functions to get the ymin and ymax. In the first we substract the standard deviation from the mean, and in the second we add the standard error to the mean. Important is to define the position again, otherwise R won't know where to plot the error bars.

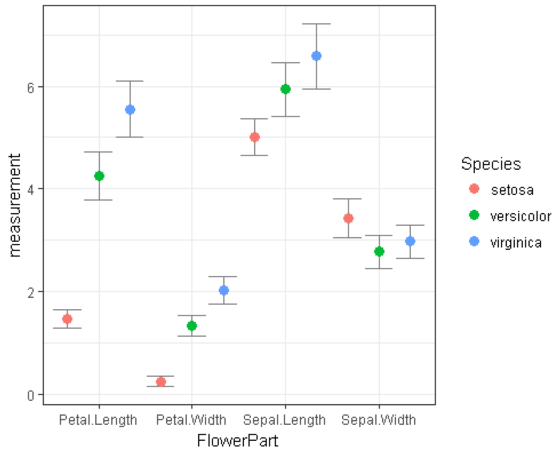
Your result should look like this:



By changing just a few lines of code, we can also create a dot plot showing mean and standard errors. This will create a higher data-to-ink ratio.

As you can see in the code below, we have to specify group in the aesthetics to make this work. Also, we have to move the errorbar `stat_summary` up. That's because ggplot2 runs each line/command at a time and plots them as it goes – if commands plot something on the graph, they occur as layers that build up on each other. (If you have worked with GIS, the concept of layers will be familiar to you). If we don't move the line defining error bars up (i.e., thereby plotting them last), they would otherwise be plotted on top of the points, making the points harder to see.

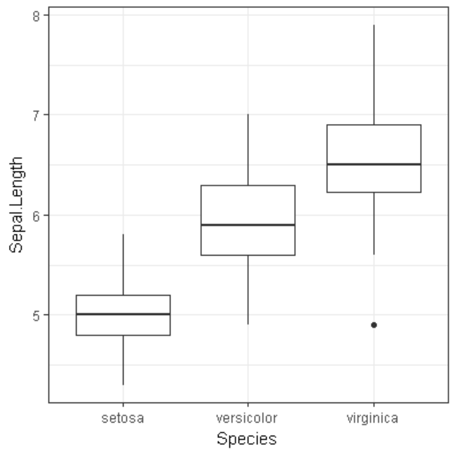
```
Fig1 = ggplot(dat2_long, aes(x=FlowerPart,y=measurement, group=Species,
                   col=Species))+
       stat_summary(fun.ymin = function(x) mean(x) - sd(x),
             fun.ymax = function(x) mean(x) + sd(x), geom = "errorbar",
             colour = "grey50",  position=position_dodge(width=1))+
        stat_summary(fun.y=mean, geom="point",
             position=position_dodge(width=1), size=3) +
      theme_bw()
Fig1
```



5.18 Modifying elements on the x-axis

It is sometimes necessary to order the groups on the x-axis, or to change the labels. The ggplot function relies on the ordering of factors in your data table. As an example, we will go back to the original iris dataset. We plot Sepal.Length by Species.

```
Fig1 = ggplot(dat2, aes(x=Species, y=Sepal.Length)) +
  geom_boxplot()+
  theme_bw()
Fig1
```



To see how the factors are ordered, type:

```
levels(dat2$Species)
```

Change the order of factors with this code and check the new order:

```
dat2$Species = factor(dat2$Species, levels(dat2$Species)[c(2,3,1)])
levels(dat2$Species)
```

If you plot this data again, the order shown on the x-axis will have changed as well. This also applies to the order in the legend (specify fill=Species to see)

```
Fig1 = ggplot(dat2, aes(x=Species, y=Sepal.Length)) +
  geom_boxplot()+
  theme_bw()
Fig1
```

5.19 Line plots to show response over time

Line plots are useful when you have data with a temporal dimension. For example, tree rings can be dated to an exact year, and we can go back in time to see how well they grew in a year based on the width of the ring. Download the “Tree-ring.csv” dataset from the course website.

```
dat3 = read.csv("Tree-rings.csv")

head(dat3)
```

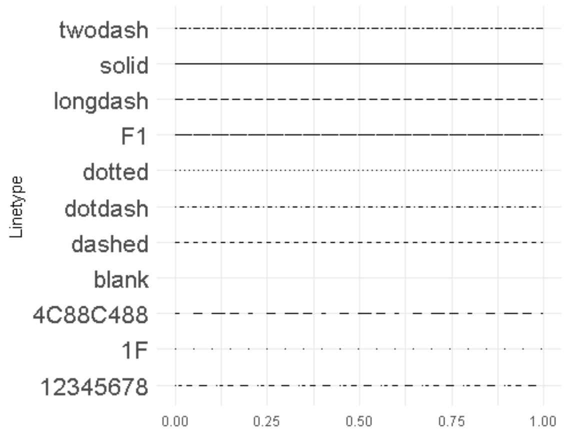
As you can see, we have bunch of trees for two sites additionally many years. Before we create a line graph, we have to calculate means and standard errors. For now, we will summarise only by Year:

```
dat3_sum =ddply(dat3, .(YEAR),summarise,
                      mRW=mean(Ringwidth,na.rm=T), sdRW=sd(Ringwidth,na.rm=T))
```

Now we can create the first line plot by adding geom_line.

```
Fig1 = ggplot(dat3_sum, aes(x=YEAR,y=mRW))+
  geom_line()+
  theme_minimal()+
  theme(legend.position = "none")
Fig1
```

There are many different linetypes available in R. These are especially useful if you want to distinguish different series in your plot.



```
Fig1 = ggplot(dat3_sum, aes(x=YEAR,y=mRW))+
  geom_line(linetype="dotdash")+
  theme_minimal()+
  theme(legend.position = "none")
Fig1
```

Try a few other linetypes.

In addition to the mean, we might want to see how ring width varies among trees. This is where the `geom_ribbon()` comes in handy. It takes ymin and ymax values as input and automatically draws a ribbon around your mean line.

```
Fig1 = ggplot(dat3_sum, aes(x=YEAR,y=mRW))+
  geom_line()+
  geom_ribbon(aes(ymin = mRW - sdRW, ymax = mRW + sdRW))+
  theme_minimal()+
  theme(legend.position = "none")
Fig1
```

This looks a bit bulky and you can’t see the line anymore. It helps if you adjust the transparency with the alpha= command.

```
Fig1 = ggplot(dat3_sum, aes(x=YEAR,y=mRW))+
  geom_line()+
  geom_ribbon(aes(ymin = mRW - sdRW, ymax = mRW + sdRW),alpha=0.5)+
  theme_minimal()+
  theme(legend.position = "none")
Fig1
```

To save this file as a raster image (png), you'll recall the code to do so from the last lab:

```
ggsave(Fig1,file="Fig1.png", width=4,height=4)
```

To save this file as a vector-style graphic (pdf), you'll recall the code to do so from the last lab (remember to specify useDingbats=F when saving a pdf otherwise the symbols will be corrupted):

```
ggsave(Fig1,file="Fig1.pdf", width=4,height=4, useDingbats=F)
```

Open both image files in your working directory and zoom in (say, 800%) to an area with some ink. Can you tell the difference between the two images?

In the following lab, we will discuss the differences in image formats (e.g. when to use a png versus a pdf). We will also cover how to make modifications of vector figures with graphics-software.

We will also cover facetting, which is great for showing multiple variables in one graph that do not share the same y-axis.

Then you will be ready to create your own publication-ready graphics!