Lab 5 – Customized Scientific Graphs & Direct Data Entry Part IV

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

This lab introduces you to a few additional tricks for creating data frames in R, for faceting and modifying your graphs using ggplot2, which help you complete the assignments or course project.

5.23 Directly data entry in R: vectors, data frames, matrices

Entering data in Excel then saving a "clean" version of the data table as a comma separated value file (.csv) is likely the most practical way of importing data. However, it can also be useful to develop your own data-tables directly in R or to manage data.

We have already covered the `c()` command in earlier labs, which creates a vector – a simple string of observations (numbers or factor). Try out these lines of code, where you can see that the strings of observations that you entered are now saved as objects in the working memory of R (remember you can change the object name).

```
Vector1 = c("Low", "Medium ", "High", "VeryHigh", "Extreme")
Vector1
Vector2 = c(1, 10, 20, 30, 40)
Vector2
```

As you can see, Vector1 and Vector2 have five observations each. Although they are clearly made up for illustration purposes, you could create a data table from these two variables because the vectors are the same length. To do so, use the `data.frame()` command:

```
Dat1 = data.frame(Vector1,Vector2)
head(Dat1)
```

Now let's make a new table of temperature highs and lows in each month over one year: the first variable will be the month, and the other two variables will be high and low temperatures for each month. First we create a vector for each variable, then combine them with the `data.frame()` function. Please run the following code:

```
MONTH = c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep",
"Oct", "Nov", "Dec")
HIGH_TEMP_2015 = c(-5, -2, 0, 5, 12, 18, 25, 22, 18, 5, 1, -3)
LOW_TEMP_2015 = c(-16, -10, -5, 2, 5, 8, 15, 12, 5, -1, -6, -11)

climate = data.frame(MONTH, HIGH_TEMP_2015  , LOW_TEMP_2015 )
head(climate)
```

You now have a data.frame, which is a table in R that allows different variables to have different data types. Let's say that you would now like to enter data for another year, too. Please now enter (or copy/paste) and run the following:

```
HIGH_TEMP_2016 = c(-7, -3, 4, 8, 18, 20, 27, 26, 14, 2, -8, -10)
LOW_TEMP_2016 = c(-19, -14, -9, -1, 7, 12, 18, 14, 7, -3, -10, -12)
HIGH_TEMP_2016
LOW_TEMP_2016
```

How would you add these new vectors to your "climate" data frame? There are a few options to do this. One option would be to just add the new vectors as a variable to your data frame because you know if will fit in the exact same order (Jan -> Dec) and is of the same length (12), you know that it will fit into the new data table:

```
Opt1 =data.frame(MONTH,HIGH_TEMP_2015 ,LOW_TEMP_2015,HIGH_TEMP_2016,
LOW_TEMP_2016)
head(Opt1)
```

Another efficient option involves `cbind()`, which stands for column-bind. This only works if you know that it is meaningful to stick another column onto your data.frame on the right-hand side. In other words, the vector must occur in the same order that you want it to occur as a column in the data frame. Try this:

```
Opt2 = cbind(climate,HIGH_TEMP_2016  , LOW_TEMP_2016)
head(Opt2)
```

Another option would be to merge the data together by a common variable, as previously covered. This is an artificial example (here it is less efficient than data.frame or cbind commands), but is included to help manage data tables since sometimes the merge function can re-order variables.

```
newclimate=data.frame(MONTH, HIGH_TEMP_2016  , LOW_TEMP_2016)
head(newclimate)

Opt3 = merge(climate,newclimate,by="MONTH")
head(Opt3) # Note new alphabetical order for MONTH
```

To order the data table, try this:

```
Opt3$MONTH<-factor(Opt3$MONTH, levels=c("Jan", "Feb", "Mar", "Apr", "May",
"Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"))
```

Or, because "MONTH" in this case is already a vector saved previously and is therefore an object in the global environment, the above line of code can be shortened to:

```
Opt3=Opt3[match(MONTH, Opt3$MONTH),]
head(Opt3)
```

A matrix is a table with only numbers (no factors or characters). These can be useful for running some analyses on your tables like correlations. To create a matrix out of the climate data, first subset to remove the month, convert the data frame to a numeric matrix, and then save it to a new object:

```
SubClim = Opt3[,2:5]   # Equivalent to: SubClim = Opt3 [,-c(1)]
MyMatrix = as.matrix(SubClim)
head(MyMatrix)
```

One disadvantage of creating a matrix is that the variable giving the rows meaning (i.e., we lost "MONTH", which is important for interpretation). However, if you have a unique ID, you can add that back to your matrix. Since each observation under "MONTH" is unique, we can do that here:

```
head(MyMatrix) # No row names
rownames(MyMatrix)=Opt3$MONTH
```

```
head(MyMatrix)  # Now you see row names
```

Finally, if you want to convert back to data frame from a matrix, you can easily do that with as.data.frame(), and then add your unique ID back as a column:

```
MyDF = as.data.frame(MyMatrix)
MyDF$MONTH2=row.names(MyDF)
head(MyDF)
```

Now we can continue to modify the data tables, as per usual, making them long, for example with the tidyr package:

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

DF = gather(MyDF, VARIABLE, TEMPERATURE, 1:4)
head(DF) # Now in long format
```

Another way to modify your data is with the stringr package (another tidyverse contribution from Hadley Wickham). Let's use the str_split_fixed function in stringr to create a new column called YEAR. This function allows you to split text in a certain column by a special symbol, tell it how many pieces to split it into, and then select which part of the section you would like to retain:

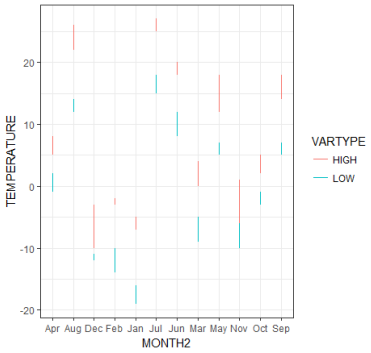
```
install.packages("stringr")
library(stringr)

DF$YEAR = str_split_fixed(DF$VARIABLE, "_", 3)[,3]
head(DF)
DF$VARTYPE = str_split_fixed(DF$VARIABLE, "_", 3)[,1]
head(DF)
```

Good work. You just created a really nice data set directly in R, which we will now use to learn about some finer points for ggplot2 usage.

5.24 ggplot2: re-ordering axes for plotting

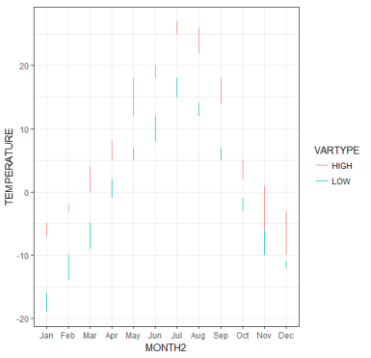
Let's explore this data using ggplot2. First, install and load the ggplot2 package and try to run a simple line graph:

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

Fig1 = ggplot(DF, aes(x=MONTH2,y=TEMPERATURE,color=VARTYPE))+
  geom_line()+
  theme_bw()
Fig1
```



The output is not great. For starters, the axes labels are not in a meaningful order, making interpretation difficult. To plot this in the order we want, we first have to convert the month to factor and re-order them with the following:

```
DF$MONTH2<-as.factor(DF$MONTH2)
summary(DF$MONTH2) # Check order. (Numbers show observations/level.)
DF$MONTH2<-factor(DF$MONTH2, levels=c("Jan", "Feb", "Mar", "Apr", "May",
   "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"))
summary(DF$MONTH2) # Check order.

Fig2 = ggplot(DF, aes(x=MONTH2,y=TEMPERATURE,color=VARTYPE))+
  geom_line()+
  theme_bw()
Fig2
```



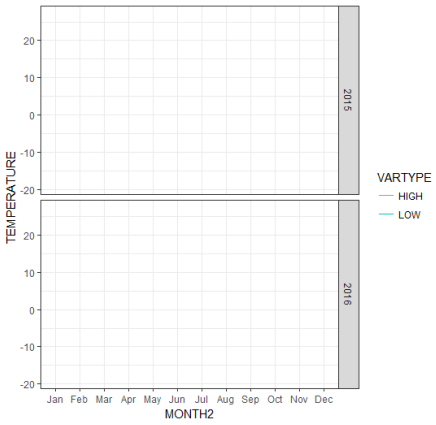
An alternative method that gives you the same results is to specify the order in the `scale_x_discrete()` function:

```
Fig2 = ggplot(DF, aes(x=MONTH2,y=TEMPERATURE,color=VARTYPE))+
  geom_line()+
  theme_bw()+
  scale_x_discrete(limits=c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"))
Fig2
```

An improvement, but ggplot2 thinks that the geom_line() command should be plotted vertically (i.e., the values for 2015 and 2016 are both plotted in January). However, we want two separate lines representing both years to be plotted horizontally across the months. Could faceting help?

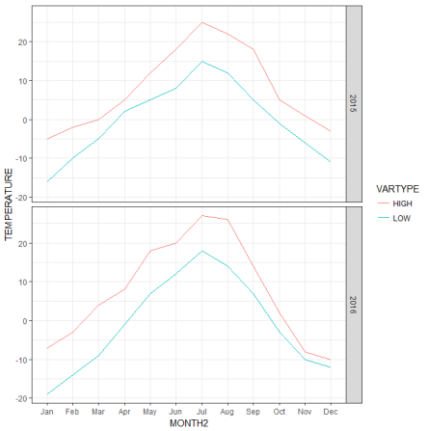
5.24 ggplot2: Grouping the data for faceting line graphs

```
Fig3 = ggplot(DF, aes(x=MONTH2,y=TEMPERATURE, color=VARTYPE))+
  geom_line()+
  facet_grid(YEAR~.)+
  theme_bw()
Fig3
```



Unfortunately we lost our data. In fact, we have to tell ggplot2 to connect the line by "group":

```
Fig4 = ggplot(DF, aes(x=MONTH2,y=TEMPERATURE,color=VARTYPE,
    group=VARTYPE))+
  geom_line()+
  facet_grid(YEAR~.)+
  theme_bw()
Fig4
```



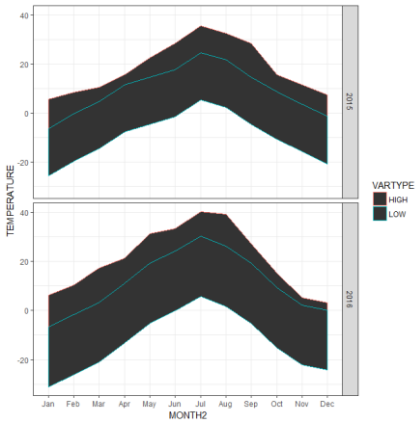
Much better. But what if we want to add some ribbons showing an average measure of spread? We can do that easily with ddply from the plyr package and merge the output back onto the data frame. Try this code:

```
head(DF)
dat=ddply(DF,.(VARIABLE), summarise, STDEV = sd(TEMPERATURE))
head(dat)
```

```
DF2 = merge(DF, dat, by="VARIABLE",keep.all=T)
head(DF2)
```

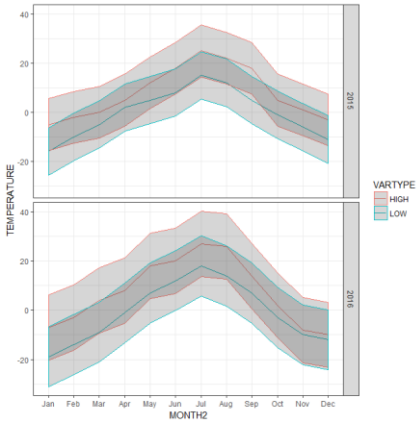
To plot this, we would like to merge this data onto our newer data frame:

```
Fig5 = ggplot(DF2, aes(x=MONTH2,y=TEMPERATURE,color=VARTYPE,
group=VARTYPE))+
  geom_line()+
  geom_ribbon(aes(ymax=TEMPERATURE+STDEV,ymin=TEMPERATURE-STDEV))+
  facet_grid(YEAR~.)+
  theme_bw()
Fig5
```



Let's make that somewhat transparent, which we learned in Lab 5 Part II:

```
Fig6 = ggplot(DF2, aes(x = MONTH2,y = TEMPERATURE, color = VARTYPE,  group
   = VARTYPE))+
  geom_line()+
  geom_ribbon(aes(ymax = TEMPERATURE + STDEV, ymin = TEMPERATURE - STDEV),
   alpha=.2)+
  facet_grid(YEAR~.)+
  theme_bw()
Fig6
```



Better. But wouldn't it be great if the colours of the ribbon matched the line?

5.25 ggplot2: setting universal parameters *versus* geom parameters

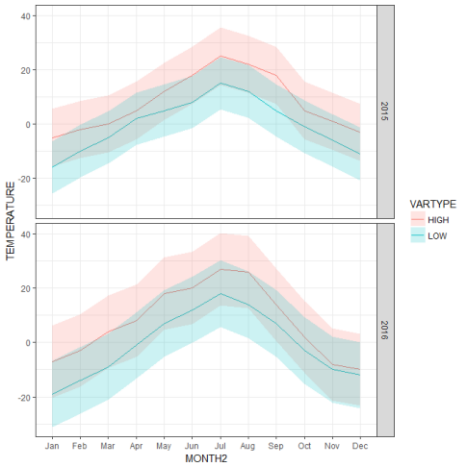
In the code above, the user specifies the most important graphical parameters directly in the main ggplot() command – in the first line. These include specifying the data table as well as the x axis, y axis and other aesthetics in the aes() command. By setting the commands in the main ggplot() command, ggplot2 will try to apply the same data source and aesthetics (e.g. "color" and "fill") to all subsequent forms of data visualization (geoms).

But what if we want to specify different aesthetics in the geoms? We can do this by removing those specifications from the ggplot() command and adding the appropriate version of those specifications within the geom command itself. For example, the following code removes "color" from the main ggplot() command and adds "color=VARTYPE" to the geom_line function (remember "color" is only for lines or outlines) and adds "fill=VARTYPE" to the geom_ribbon function:

```
Fig7= ggplot(DF2, aes(x=MONTH2,y=TEMPERATURE,  group=VARTYPE))+
  geom_line(aes(color=VARTYPE))+
  geom_ribbon(aes(ymax=TEMPERATURE+STDEV,ymin=TEMPERATURE-
STDEV,fill=VARTYPE), alpha=.2)+
  facet_grid(YEAR~.)+
  theme_bw()
Fig7
```

We can even specify the data source within the geom itself – useful if you ever have to plot data saved in different objects in R (but note: they should share the same axes). The following code removes all specifications from the main ggplot() command, but now the relevant code needs to be repeated within each geom:

```
Fig8= ggplot()+
  geom_line(data=DF2, aes(x=MONTH2,y=TEMPERATURE,
group=VARTYPE,color=VARTYPE))+
  geom_ribbon(data=DF2,aes(x=MONTH2,
ymax=TEMPERATURE+STDEV,ymin=TEMPERATURE-STDEV,fill=VARTYPE,
group=VARTYPE), alpha=.2)+
  facet_grid(YEAR~.)+
  theme_bw()
Fig8
```



Not bad. To modify the colors, you may want to look at pages 3 and 4 from Lab 5 Part II. To modify axis labels and to add a title, please see page 7 in Lab 5 Part I (*hint:* see page 4 if you want the title on two lines). To save it using code, please refer to page 9 of Lab 5 Part I or page 1 of Part II.

You are now a master at graphics using ggplot2 in R!