Data Visualization Workshop – Biology 850

So far, we've learned how to import our data in RStudio, format and manipulate them, and now it's time we talk about communicating the results of our analyses - data visualisation! Next week we will start getting into how to do some basic analyses in R.

Note: I've adapted this workshop from The University of Edinburgh's Ecology Coding Club here; as well as information on ggplot2 here; here and here

Today, we will be using ggplot2 - another package that makes up the tidyverse. There are lots of different ways to visualize and plot your data, including base R, but ggplot2 is easy to use, versatile, and we will focus on using this package. **ggplot2** is a plotting package that makes it simple to create complex plots from data in a data frame. We will use an example set of data to try making some various graphics.

ggplot2 uses something called the "grammar of graphics" (the gg of ggplot) which separates the basic components of a graphic into distinct parts (e.g. like the parts of a sentence). You add these parts together to get a figure. ggplot graphics are built step by step by adding new elements. Adding layers in this fashion allows for extensive flexibility and customization of plots.

Lesson Goals:

- Become familar with some basic plots in ggplot2
- Be able to manipulate the default themes
- Output plots to files
- Know where to get some help

Note : all the files you need to complete this tutorial can be obtained on a GitHub repository. you can fork the repository to your own Github account and then add it as a new RStudio project by copying the URL link.

1. On GitHub, go to the repository on my GitHub, and click the Fork button (topright) to create a fork in your own account
   https://github.com/JonSweetman/BIOL850_dataviz
2. In RStudio: Choose File → New Project → Version Control → Git
   Then enter the URLfor your repository, a Project directory name, and choose a subdirectory where you want to put the files.
   Click the "Create Project" button

Prerequisites: We will be using ggplot2, part of the tidyverse package, as well as a package called **gridExtra, which helps** arrange multiple grid-based plots on a page. You should have tidyverse already installed from the previous workshop, but you will need to install gridExtra

```
install.packages("gridExtra ")
```

1. Use a script!

A script is a `.R` file that contains your code: you could directly type code into the R console, but that way you have no record of it and you won't be able to reuse it later. To make a new `.R` file, open RStudio and go to `File/New file/R script`

Open RStudio, select `File/New File/R script` and start writing your script with the help of this tutorial.

```
# Purpose of the script
# Your name, date and email


# Libraries - if you haven't installed them before, run the code install.packages("package_name")
library(tidyverse)
library(gridExtra)
```

We will use data from the Living Planet Index, which you have already downloaded from the repository. (Note this is a slightly different way of reading in data)

```
# Import data from the Living Planet Index - population trends of vertebrate species from 1970 to 2014
LPI <- read.csv(file.choose())  # find where you saved LPIdata_CC.csv
```

This is a large dataset. We will first explore our dataset using the `str()` function which shows what type each variable is - what is the dataset made of?

```
# str displays the structure of our data
str(LPI)
```

The data are in wide format - the different years are column names, when really they should be rows in the same column. We will reshape the data using the `gather()` function from the `tidyr` package.

```
# Reshape data into long form
# By adding 9:53, we select columns from 9 to 53, the ones for the different years of monitoring
LPI2 <- gather(LPI, "year", "abundance", 9:53)
View(LPI2)
```

There is an 'X' in front of all the years because when we imported the data, all column names become characters. R puts an 'X' in front of the years to turn the numbers into characters. Now

that the years are rows, not columns, we need them to be proper numbers, so we will transform them using `parse_number()` from the `readr` package.

```
# parse_number drops any non-numeric characters before or after the first number
LPI2$year <- parse_number(LPI2$year)

# When manipulating data it's always good check if the variables have stayed how we want them
# Use the str() function
str(LPI2)

# Abundance is also a character variable, when it should be numeric, let's fix that
LPI2$abundance <- as.numeric(LPI2$abundance)
```

This is a very large dataset, so for the first few graphs we will focus on how the population of one species has changed. Pick a species of your choice, make sure you spell it the same way as it is entered in the dataframe, in this example we are using the “Griffon vulture”, but you can use whatever species you want. To see what species are available use the following to get a list:

```
# unique returns a vector, data frame or an array with duplicate entries removed
unique(LPI2$Common.Name)
```

Then filter out just the records for that species using the following code, substituting `Common.Name` for your chosen species. As it is NDSU, I’ll use the American bison as my example.

```
# filter is part of the dplyr package we looked at in the data manipulation workshop
bison <- filter(LPI2, Common.Name == "American bison / Wood bison / Plains bison")
head(bison)

# There are a lot of NAs in this dataframe, so we will get rid of the empty rows using na.omit()
bison <- na.omit(bison)
```

Visualizations! Steps for creating a Plot using ggplot2

We will go through a few examples of creating plots using ggplot. The great thing about this, is that the approach and steps are exactly the same for any kind of plot. So once you learn how, you can create

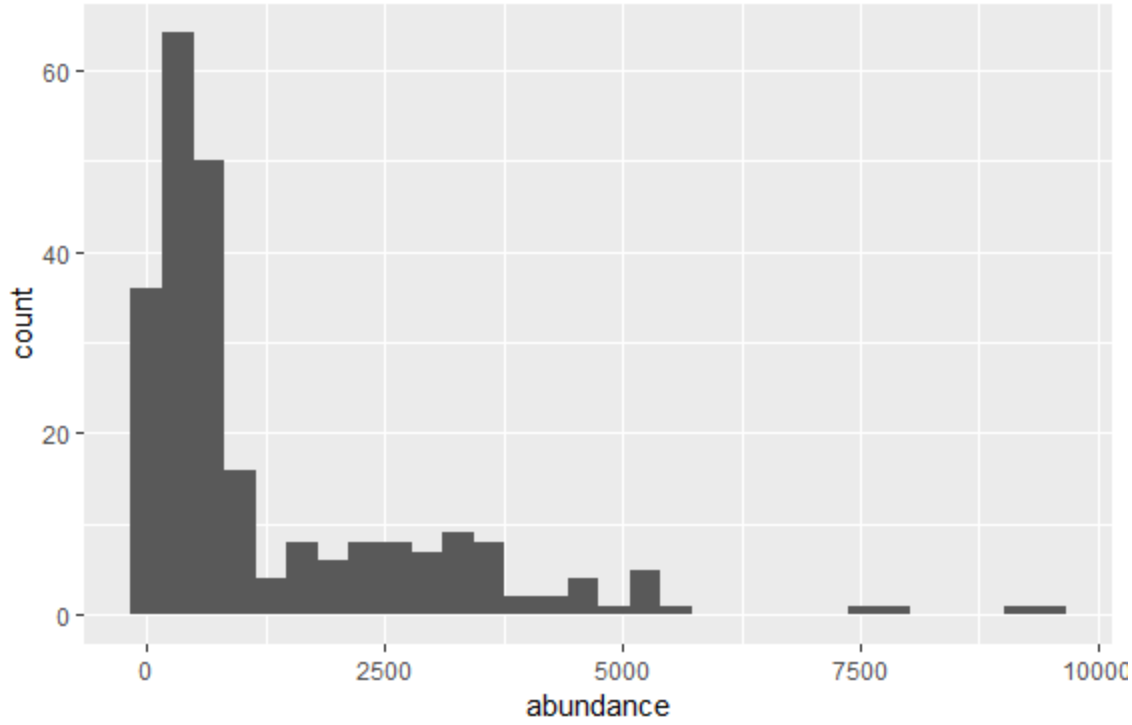
any plot/figure you need. See R for Data Science and the RStudio cheatsheet for ggplot, and this cheatsheet by Zev Ross for more details

For inspiration on the types of plots

Creating a histogram to visualize data distribution

```
bison_hist <- ggplot(bison, aes(x=abundance)) +
  geom_histogram()
# putting your entire ggplot code in () creates the graph and shows it in the plot viewer. Without brackets, you hav
e to call the object, so that it gets displayed, e.g.
 bison_hist
# Here is how it looks with the brackets
(bison_hist <- ggplot(bison, aes(x=abundance)) +
  geom_histogram())
```

Your output will look something like this:



These are the default ggplot settings. There is lots of unnecessary grey space behind the histogram, the axes labels are small, and the bars blend together. This can be improved

```
(bison_hist <- ggplot(bison, aes(x=abundance)) +
```

```
# Changing the binwidth and colours
  geom_histogram(binwidth=250, colour="#8B5A00", fill="#CD8500") +
# Adding a line for mean abundance
  geom_vline(aes(xintercept=mean(abundance)),
# Changing the look of the line
                colour="red", linetype="dashed", size=1) +
# Changing the theme to get rid of the grey background


    theme_bw() +
# Changing the text of the y axis label
ylab("Count\n") +
 # \n adds a blank line
xlab("\nBison abundance")  +
# Changing font size of axis labels
theme(axis.text.x=element_text(size=12),
                axis.text.y=element_text(size=12),
# Changing font size of axis titles
                axis.title.x=element_text(size=14, face="plain"),
# face="plain" changes font type, could also be italic, etc
                axis.title.y=element_text(size=14, face="plain"),
# Removing the grey grid lines
                panel.grid.major.x=element_blank(),
                panel.grid.minor.x=element_blank(),
                panel.grid.minor.y=element_blank(),
                panel.grid.major.y=element_blank(),
# Putting a 1 cm margin around the plot
        plot.margin = unit(c(1,1,1,1), units = , "cm")))


# We can see from the histogram that the data are very skewed –
# a typical distribution of count abunda nce data
```

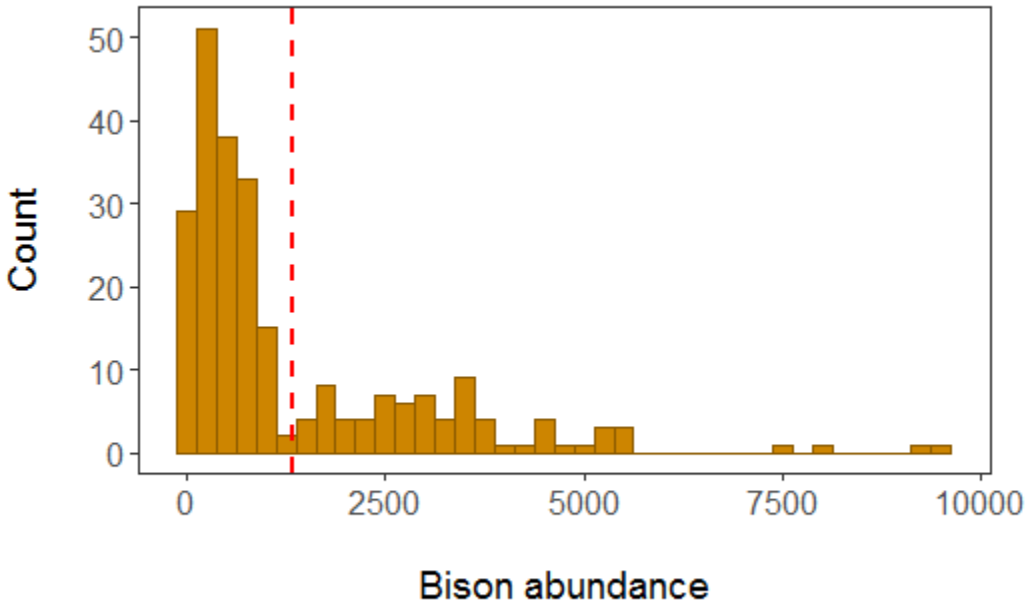


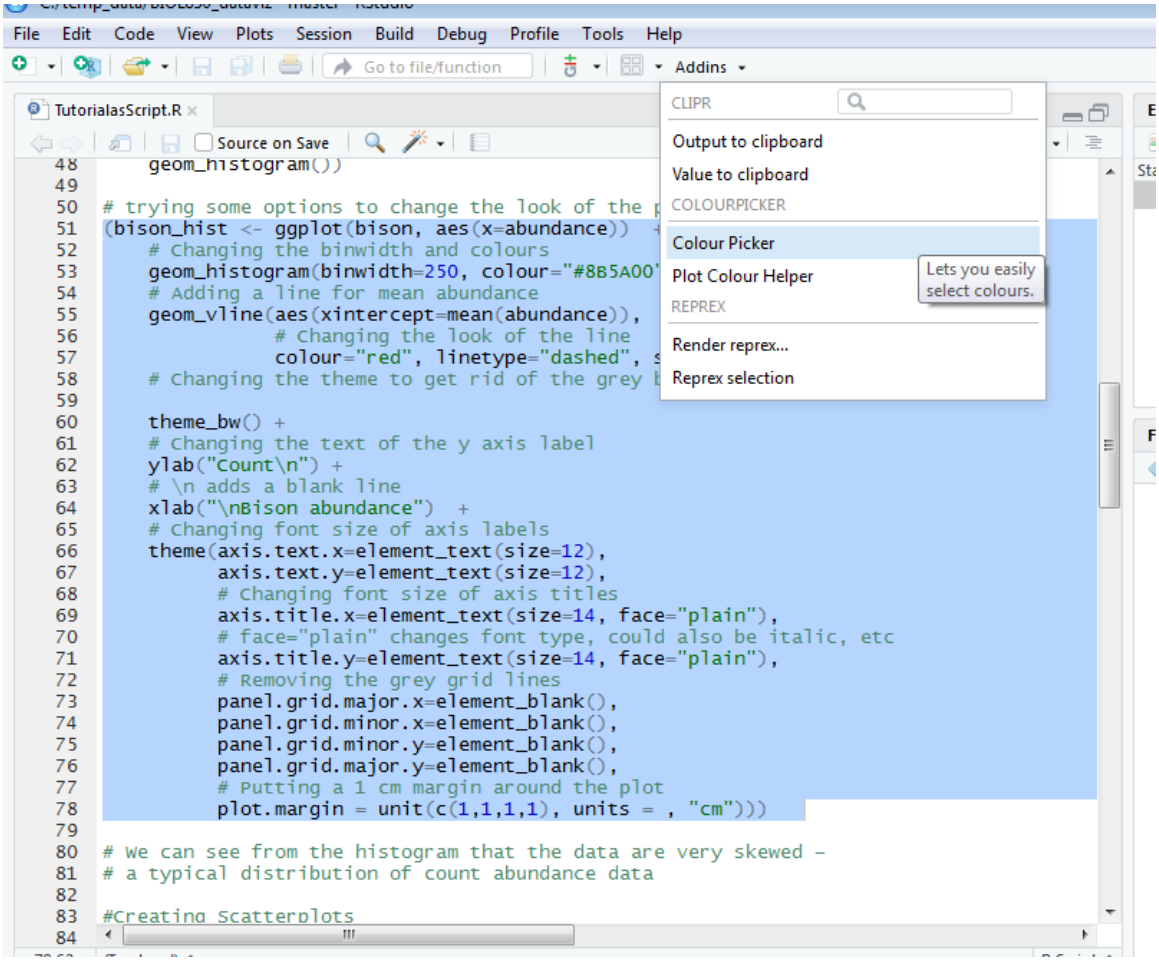
Figure 1. Histogram of American Bison abundance in populations included in the LPI dataset. Red line shows mean abundance.

Pressing enter after each "layer" of your plot (i.e. indenting it) prevents the code from being one gigantic line and makes it much easier to read.

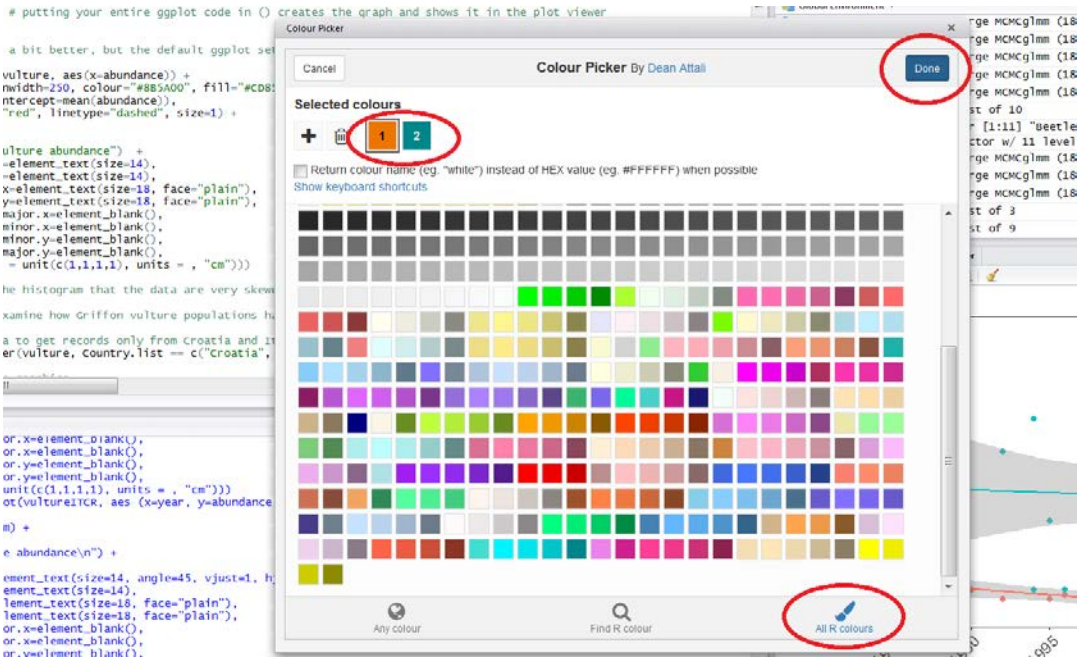
In the code above you can see a colour code `colour="#8B5A00"` - each colour has a code, called a "hex code", a combination of letters and numbers. You can get the codes for different colours online, from Paint, Photoshop or similar programs, or even from RStudio, which is very convenient! There is an RStudio Colourpicker addin - to install it, run the following code:

```
install.packages("colourpicker")
```

To find out what is the code for a colour you like, click on Addins/Colour picker.



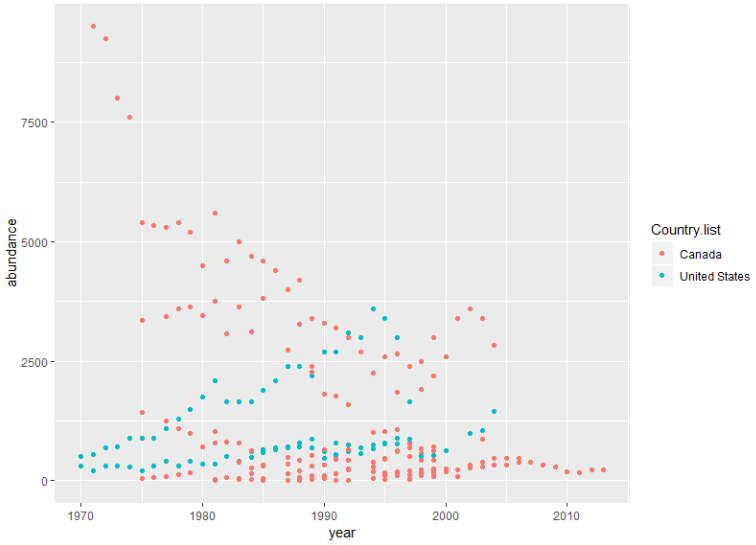
When you click on All R colours you will see lots of different colours you can choose from - a good colour scheme makes your graph stand out, but of course, don't go crazy with the colours. When you click on 1, and then on a certain colour, you fill up 1 with that colour, same goes for 2, 3 - you can add more colours with the +, or delete them by clicking the bin. Once you've made your pick, click Done. You will see a line of code c("#8B5A00", "#CD8500") appear - in this case, we just need the colour code, so we can copy that, and delete the rest. Try changing the colour of the histogram you made just now.



Creating Scatterplots

Scatterplot to examine how American Bison populations have changed between 1970 and 2014 in the US and Canada

```
# note that the code is similar to the histogram, except a different geom
(bison_scatter <- ggplot(bison, aes (x=year, y=abundance, colour=Country.list)) +
    geom_point())
```



The figure above still needs some work to make it look nice. You might have noticed that sometimes we have the `colour =` argument surrounded by `aes()` and sometimes we don't. If you are designating colours based on a certain variable in your data, like here `colour = Country.list`, then that goes in the `aes()` argument. If you just want to give the lines, dots or bars a certain colour, then you can use e.g. `colour = "blue"` and that does not need to be surrounded by `aes()`.

```
(bison_scatter <- ggplot(bison, aes (x=year, y=abundance, colour=Country.list)) +
   geom_point(size=2) +                    # Changing point size
   geom_smooth(method=lm, aes(fill=Country.list)) + # Adding a linear model fit and colour-coding by country
   theme_bw() +
   scale_fill_manual(values = c("#EE7600", "#00868B")) +    # Adding custom colours
   scale_colour_manual(values = c("#EE7600", "#00868B")) +   # Adding custom colours
   ylab("American Bison abundance\n") +
   xlab("\nYear")  +
   theme(axis.text.x=element_text(size=12, angle=45, vjust=1, hjust=1),      # making the years at a bit of an angl
e
         axis.text.y=element_text(size=12),
         axis.title.x=element_text(size=14, face="plain"),
         axis.title.y=element_text(size=14, face="plain"),
         panel.grid.major.x=element_blank(),                          # Removing the background grid lines
         panel.grid.minor.x=element_blank(),
         panel.grid.minor.y=element_blank(),
         panel.grid.major.y=element_blank(),
         plot.margin = unit(c(1,1,1,1), units = , "cm")) +               # Adding a 1cm margin around the plot
   theme(legend.text = element_text(size=12, face="italic"),              # Setting the font for the legend text
         legend.title = element_blank(),                                 # Removing the legend title
         legend.position=c(0.9, 0.9)))              # Setting the position for the legend - 0 is left/bottom, 1 is top/right
```

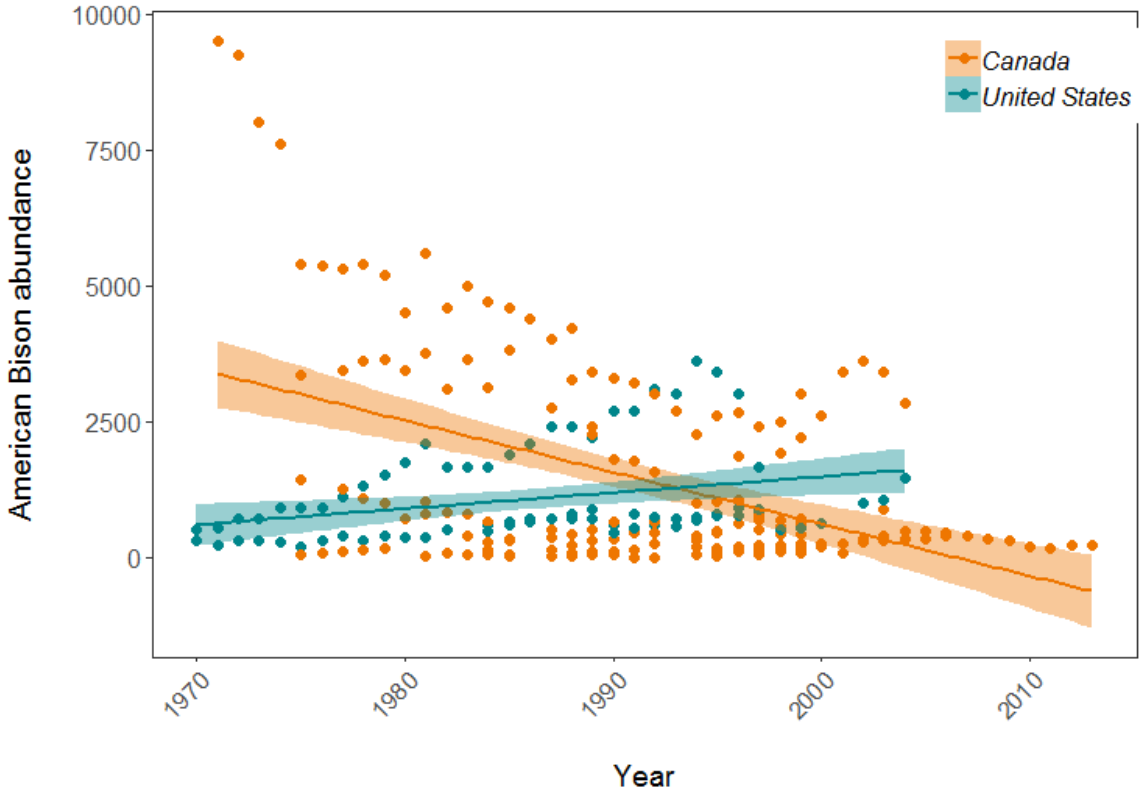


Figure 2. Population trends of American Bison in Canada and the United States. Data points represent raw data with a linear model fit and 95% confidence intervals. Abundance is measured in number of breeding individuals.

Boxplot to examine whether bison abundance differs between Canada and the US

```
(bison_boxplot <- ggplot (bison, aes(Country.list, abundance)) + geom_boxplot())


# Beautifying


(bison_boxplot <- ggplot (bison, aes(Country.list, abundance)) + geom_boxplot(aes(fill=Country.list)) +
   theme_bw() +
   scale_fill_manual(values = c("#EE7600", "#00868B")) +            # Adding custom colours
   scale_colour_manual(values = c("#EE7600", "#00868B")) +          # Adding custom colours
   ylab("American Bison abundance\n") +
```

```
xlab("\nCountry") +
theme(axis.text.x=element_text(size=12),
      axis.text.y=element_text(size=12),
      axis.title.x=element_text(size=14, face="plain"),
      axis.title.y=element_text(size=14, face="plain"),
      panel.grid.major.x=element_blank(),                    # Removing the background grid lines
      panel.grid.minor.x=element_blank(),
      panel.grid.minor.y=element_blank(),
      panel.grid.major.y=element_blank(),
      plot.margin = unit(c(1,1,1,1), units = , "cm"),        # Adding a margin
      legend.position="none"))                               # Removing the legend - not needed with only two factors
```

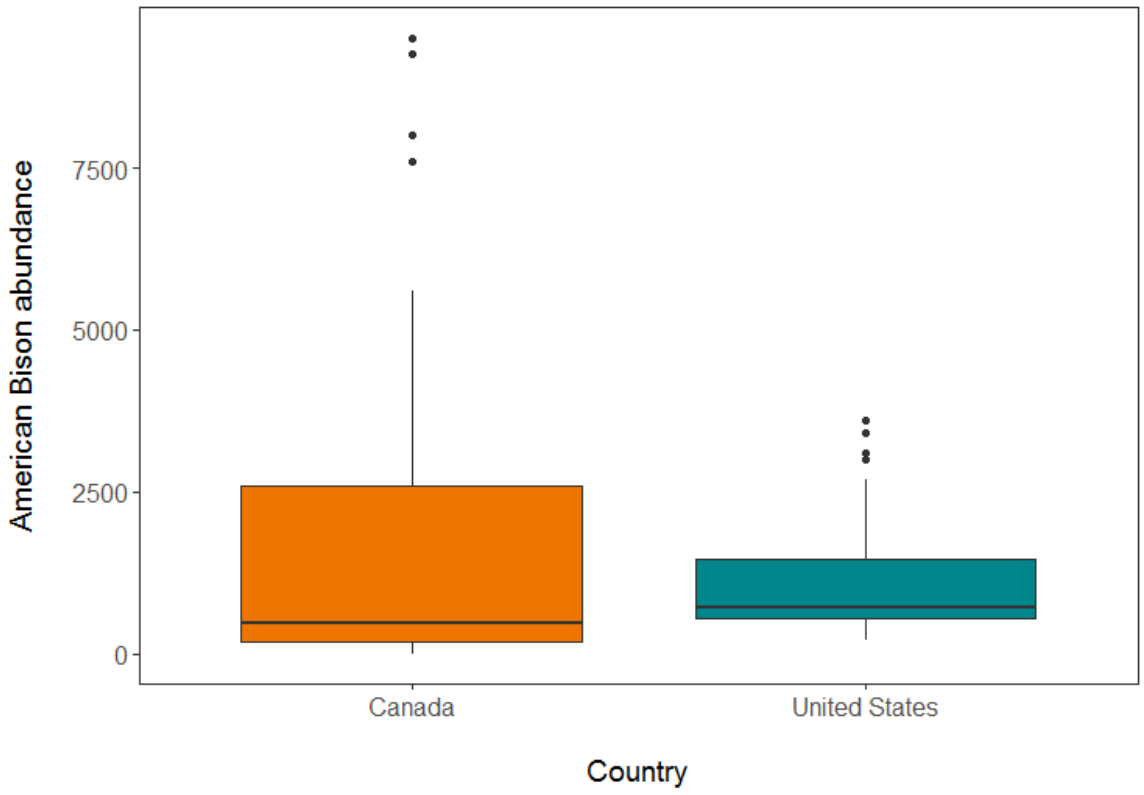


Figure 3. American Bison abundance in Canada and the United States.

Barplot to examine the species richness of a few European countries

```
# Calculating species richness using pipes ``%>%` from the `dplyr` package
richness <- LPI2 %>% filter (Country.list == c("United Kingdom", "Germany", "France", "Netherlands", "Italy")) %>%
        group_by(Country.list) %>%
        mutate (., richness=(length(unique(Common.Name))))


(richness_barplot <- ggplot(richness, aes(x=Country.list, y=richness)) +
   geom_bar(position=position_dodge(), stat="identity", colour="black", fill="#00868B") +
   theme_bw() +
   ylab("Species richness\n") +
   xlab("Country")  +
   theme(axis.text.x=element_text(size=12, angle=45, vjust=1, hjust=1),  # x axis labels angled, so that text doesn't overlap
        axis.text.y=element_text(size=12),
        axis.title.x=element_text(size=14, face="plain"),
        axis.title.y=element_text(size=14, face="plain"),
        panel.grid.major.x=element_blank(),
        panel.grid.minor.x=element_blank(),
        panel.grid.minor.y=element_blank(),
        panel.grid.major.y=element_blank(),
        plot.margin = unit(c(1,1,1,1), units = , "cm")))
```

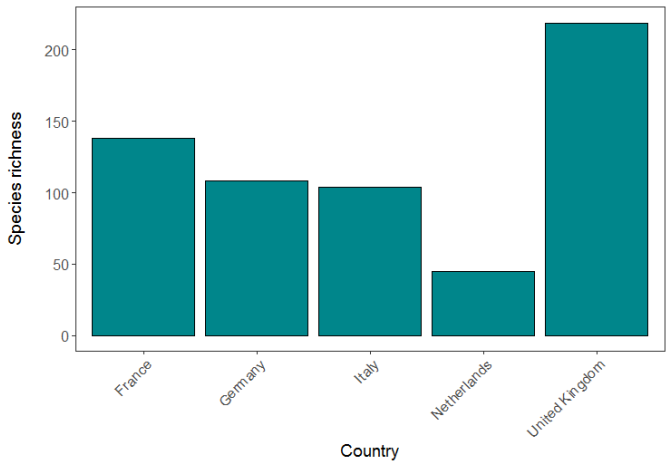


Figure 4. Species richness in five European countries. Based on LPI data.

You might be picking up on the fact that we repeat a lot of the same code - same font size, same margins, etc. Less repetition makes for tidier code and it's important to have consistent formatting across graphs for the same project. If you have settled on a look, you can make your own functions to make your own ggplot2 theme, that you can reuse it in all your ggplots. We won't cover this in the tutorial today, but potentially in a future class.

Arranging plots in a panel using grid.arrange() from the package gridExtra

```
grid.arrange(bison_hist, bison_scatter, bison_boxplot, ncol=1)


# This doesn't look right - the graphs are too stretched, the legend and text are all messed up, the white margins a
re too big

# Fixing the problems - adding ylab() again overrides the previous settings


panel <- grid.arrange(bison_hist + ggtitle("(a)") + ylab("Count") + xlab("Abundance") +   # adding labels to the diffe
rent plots
                      theme(plot.margin = unit(c(0.2, 0.2, 0.2, 0.2), units = , "cm")),
                    bison_boxplot + ggtitle("(b)") + ylab("Abundance") + xlab("Country") +
                      theme(plot.margin = unit(c(0.2, 0.2, 0.2, 0.2), units = , "cm")),
                    bison_scatter + ggtitle("(c)") + ylab("Abundance") + xlab("Year") +
                      theme(plot.margin = unit(c(0.2, 0.2, 0.2, 0.2), units = , "cm")) +
                      theme(legend.text = element_text(size=12, face="italic"),
                            legend.title = element_blank(),
                            legend.position=c(0.85, 0.85)), # changing the legend position so that it fits within the panel
                    ncol=1) # ncol determines how many columns you have
```

Saving your Figures:

To get around the too stretched/too squished panel problems, we will save the file and give it exact dimensions using ``ggsave`.

The ggsave() function saves your figure

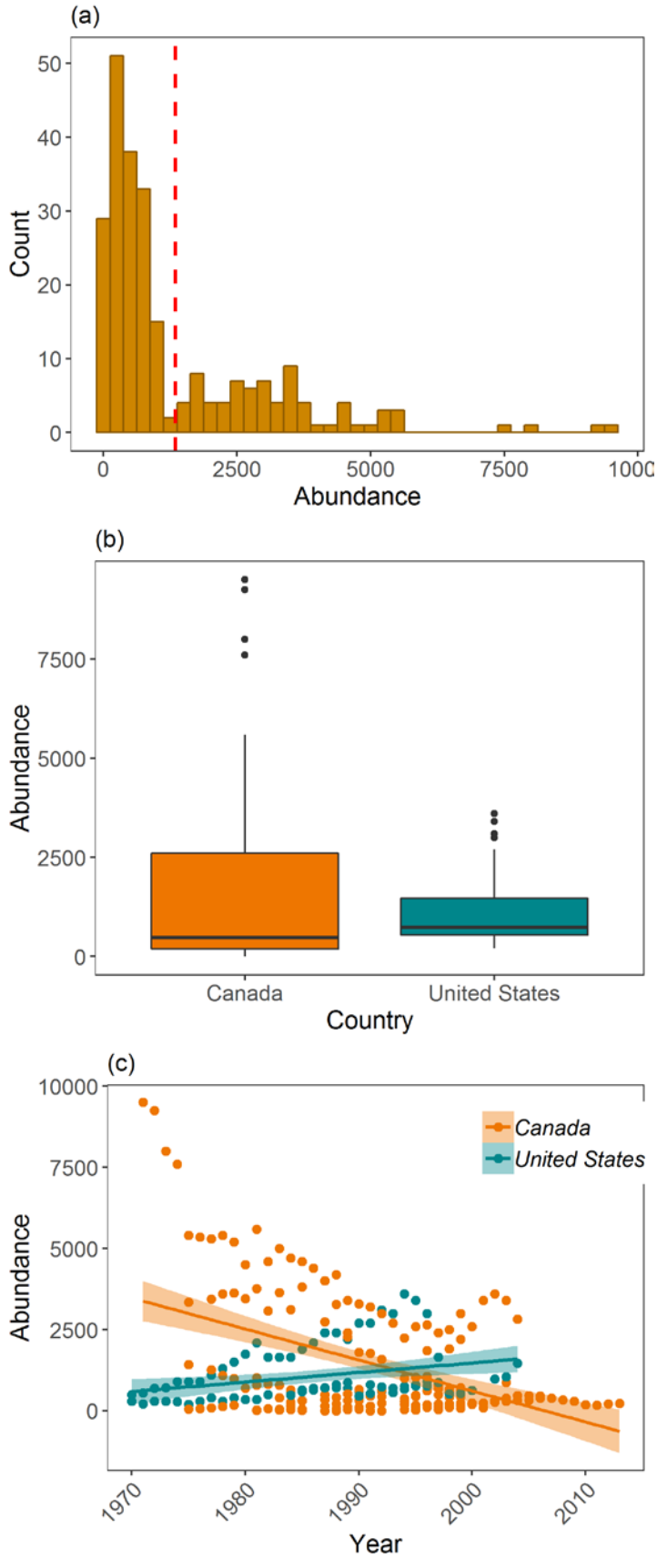
Before we do this, it is good practice to save your figures in a subdirectory of your working directory, named output or figures. You can manually create the subfolder in your working directory, and refer to it's relative path, like I have below:

```
#make sure you manually create the subdirectory folder first before saving!

ggsave(panel, file="figures/bison_panel2.png", width=5, height=12) # the file is saved in your working directory, find it with getwd()

ggsave(panel, file="figures/bison_panel2.pdf", width=5, height=12) # save as a PDF
```

Note on File formats for figures:

If you're printing, use PDF - If you plan to print your graphic, you want to use a vector-based format. This means that the graphic is represented in a scale-independent format, and it can be recreated in any size small or large without resulting in jagged lines or pixellated text. When you print it on a printer, lines will appear smooth and text will be clear, even if the graphic has been enlarged or reduced and regardless of the DPI (dots-per-inch) rating of the printer. PDF is a good choice, since most people have the ability to view PDF files.

For the internet/web display, a good choice is a PNG. GIF was the most popular format for many years, but it has several limitations (not least, graphs using many colors -- like image plots -- might not look correct in GIF format). These days, the best choice is the PNG format, generated by the png() driver. Most browsers these days can display PNG graphics without trouble.


(a)
Count
Abundance
0
10
20
30
40
50
0
2500
5000
7500
(b)
Abundance
Country
Canada
United States
0
2500
5000
7500
(c)
Abundance
Year
Canada
United States
0
2500
5000
7500
10000
1970
1980
1990
2000
2010