Visualising missing data in a scatterplot, and annotating ggplots

Susannah Cowtan

1 July 2019

Get yourself set up

You'll need these packages loaded. You don't have to use `suppressPackageStartupMessages()`, but I do, because none of the package conflicts where one package's "filter", say, masks another package's are going to cause a problem here, so I'd rather not have those messages cluttering up my output.

```
# I am an incorrigible tidyverse programmer.  Sorry not sorry.
suppressPackageStartupMessages(library(dplyr))
library(readr)
suppressPackageStartupMessages(library(tidyr))

# Libraries for graphics
suppressPackageStartupMessages(library(scales))
library(ggplot2)
library(ggrepel)
library(naniar)
```

Define some global variables

I want to have these defined at the top so I only ever need to change them in one place.

```
# My "target species"
species <- "Hermania scabra"

# "target year" - we look at the observations before and including/after this
target_year <- 1992
```

Read in the data

My data are in a directory called `talk_data`, which is in the same directory as the directory holding my scripts. I always access data using relative paths because they are less likely to get broken. You may still need to change the path, depending on where you put the data! Usually I'd define the path to the data directory where I define the global variables, above.

I use `readr::read_csv` rather than read.csv because it is faster and makes fewer assumptions - in particular, it doesn't assume character columns are factors. Then I contradict myself by telling it to make assumptions, using `col_types = cols()` (but it still doesn't make characters factors).

```
shark_df <- read_csv("../talk_data/shark.csv", col_types = cols())
```

Sadly, there are no actual sharks in these data that I know of; it's mostly molluscs.

Set up a theme

You don't need to set up the theme for the workshop, but you have it if you want it for reference. I suggest you **do** define the colours, because those will appear in the code later.

```
# Custom colours (colourblind friendly, supposedly!) -
# Pick widely spaced range, and omit colours nearest black and white
prof_colours <- scales::viridis_pal(option = "viridis")(8)[c(2, 5)]
# Nice and contrasty (for a colour vision person - should really check this!)
highlight_colour <- "violetred3"

# Look of ggplot
theme_prof <- function (base_size = 12, base_family = "") {
  theme_bw(base_size = base_size, base_family = base_family) %+replace%
    theme(
      axis.text = element_text(size = 10),
      axis.title = element_text(size = 12),
      axis.title.x = element_text(margin = margin(t = 14)),
      axis.title.y = element_text(margin = margin(l = 0, r = 14),
        angle = 90),
      legend.text = element_text(size = 10),
      legend.title = element_text(size = 12)
    )
}

# Make this theme the default
theme_set(theme_prof())
```

Data munging: classify observations as pre or post the target year

You don't need to type the `packagename::` parts of this as you have the packages loaded; I include them to show which package different commands come from, in case you're unfamiliar with the tidyverse.

A big part of making good plots is getting the data in the right format beforehand. `ggplot` likes one column per variable. Here my variables are `post` and `pre`, referring to whether the observation was made after the target year or not, and `scientificName` for the labels. It doesn't start out that way though! Can you see what I did?

```
## Pre & post target_year
shark_gg_df <- shark_df %>%
  # Up to and including target year, or after?
  dplyr::mutate(pre_post = ifelse(year > target_year, "post", "pre")) %>%
  # I only need these columns
  dplyr::select(scientificName, pre_post) %>%
  # Group the data by pre or post target_year
  # Any operation after this will be performed on each group separately
  dplyr::group_by(pre_post) %>%
    # Count the number of records for each species (puts in column "n")
    dplyr::count(scientificName) %>%
  dplyr::ungroup() %>%
  # ggplot likes a "wide" format, so pre and post need to be separate columns
  # get columns scientificName, pre (count), post (count)
  tidyr::spread(pre_post, n)
```

Making a terrible approximation that survey effort was equal for all years, calculate the slope for a reference line.

```
# Calculate slope of comparison line by how many years either side of target_year
# (approximation - takes no account of survey effort)
slope <- length(which(unique(shark_df$year) > target_year)) /
  length(which(unique(shark_df$year) <= target_year))
```

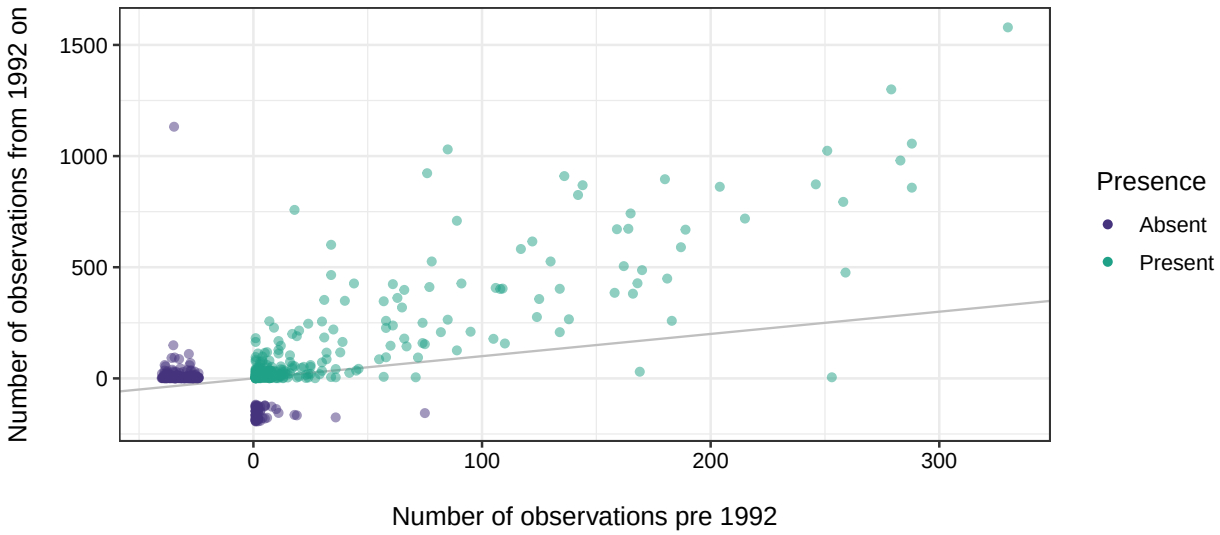
Plotting missing data with `naniar`

Here we're just making a scatterplot, but we're using `geom_miss_point` from the `naniar` package where we would usually use `geom_point`.

We then need two colours defined - here I'm using `scale_colour_manual` because I'm using a vector of colours I chose myself. You can miss out this whole command and get the default colours.

```
shark_missing_gg <- shark_gg_df %>%
  ggplot(aes(x = pre, y = post)) +
    # Comparison line to show equality of observations given the number of years
    # I like to put this under the datapoints, so it's in here first
    geom_abline(slope = slope, intercept = 0, col = "grey75") +
    # Scatterplot of data including missing data
    naniar::geom_miss_point(alpha = 0.5) +
    # Set my own discrete colour scheme
    scale_colour_manual(values = prof_colours,
                        # Change legend header
                        guide = guide_legend("Presence"),
                        # Change key labels from "Not missing" and "Missing"
                        labels = c("Absent", "Present")) +
    # Human-readable axis labels
    labs(x = paste("Number of observations pre", target_year),
         y = paste("Number of observations from", target_year, "on"))

# Do plot
shark_missing_gg
```



Labelling interesting points

Some of these points look interesting. There are some species which were common but have gone extinct, or which were not present before 1992 but which have subsequently colonised the area strongly. There are also some species which are still present in both time periods but in one they are quite common and in the other very rare. Finally, I have a target species, so I sould probably label that!

Identifying species to be labelled

```
# Species present in both periods and interesting
shark_present_df <- shark_gg_df %>%
  # Filter the dataset to get the interesting species
  dplyr::filter(scientificName == species |
        # Successful post-target_year
        (pre > 150 & pre > post) |
        # Strong decline post-target_year from high base
        (post > 500 & (post / pre) > 15))

# Particularly noticeable extinction and colonisation points
shark_absent_df <- shark_gg_df %>%
  # Only want observations where either pre or post are missing
  dplyr::filter(!complete.cases(pre, post)) %>%
  # Thresholds determined by eye
  dplyr::filter(pre > 35 | post > 500)

# They are exactly the same format so I can just stick them together
shark_labels_df <- dplyr::bind_rows(shark_present_df, shark_absent_df)
```

Adding labels - the wrong way

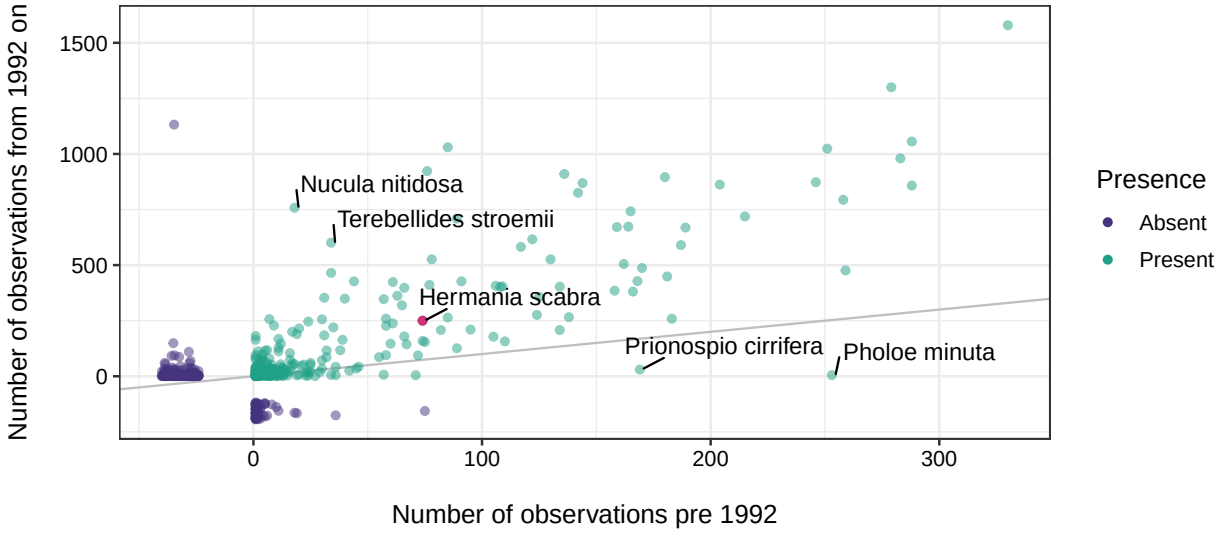
Here, instead of `geom_text` or `geom_label`, I am using `ggrepel`'s `geom_text_repel`. This means that if I label points which are close together, the position of the label is adjusted so they don't overlap. There is also a `geom_label_repel` which draws a neat little box to put the label in, but I don't like it here because it obscures other points. You can try it out though, and see what you think!

Can you see why this code won't label the colonisation / extinction points?

```
shark_bad_labels_gg <- shark_missing_gg +
  # Colour my target species
  # gghighlight doesn't work with geom_miss_point, do it manually
  geom_point(data = dplyr::filter(shark_gg_df, scientificName == species),
    col = highlight_colour) +
  # Label all species of interest with ggrepel
  # ggrepel stops your text labels overlapping
  ggrepel::geom_text_repel(data = shark_labels_df,
    aes(label = scientificName),
    # Avoids ambiguity - moving the labels away from the points means
    # it will make a short line from the label to the exact point
    nudge_x = 15, nudge_y = 15,
    # This means the line doesn't intrude into the point
    point.padding = 0.25)
```

```
# Do plot
shark_bad_labels_gg
```

```
## Warning: Removed 3 rows containing missing values (geom_text_repel).
```



Adding labels to missing data - the kludgey way

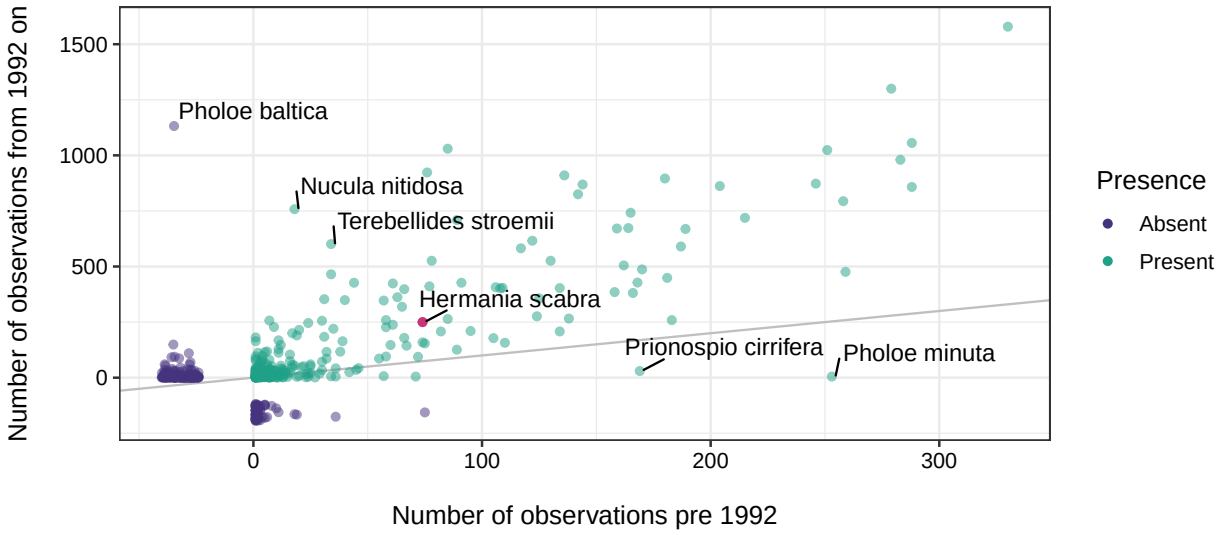
Let's take the last plot and add a label manually for one of the missing datapoints, using `annotate`. If you want to get rid of the messages about missing data, you can replace `data = shark_labels_df` in the `geom_text_repel` command in the above code with `data = shark_present_df`.

These plots are made with RMarkdown chunk options `fig.height = 3.5, fig.width = 8` – different sized plots will show annotations in the wrong place!

```
# There is only one colonising species in our label dataframe
i <- which(is.na(shark_absent_df$pre))

# Let's add that one as a label
shark_bad_labels_gg +
  annotate("text",
           # Annotate tries to put the middle of the label nearest the point,
           # but that doesn't match and won't work this close to the y axis
           # so I am calculating the position using the length of the string
           #
           # The "fudge factors" are done by eye - time consuming!
           x = (nchar(shark_absent_df[i,]$scientificName) * 2) - 28,
           y = shark_absent_df[i,]$post + 70,
           label = shark_absent_df[i,]$scientificName)
```

```
## Warning: Removed 3 rows containing missing values (geom_text_repel).
```



That worked, but it took a lot of fiddling around to get it in the right place, and if you print the plot out with different dimensions it will be wrong again. Not ideal! Mostly we avoid annotate, or use it for just one or two bits of text (or lines, or rectangles, or...) where we want to judge by eye where it should go.

Adding labels to missing data - the smart way

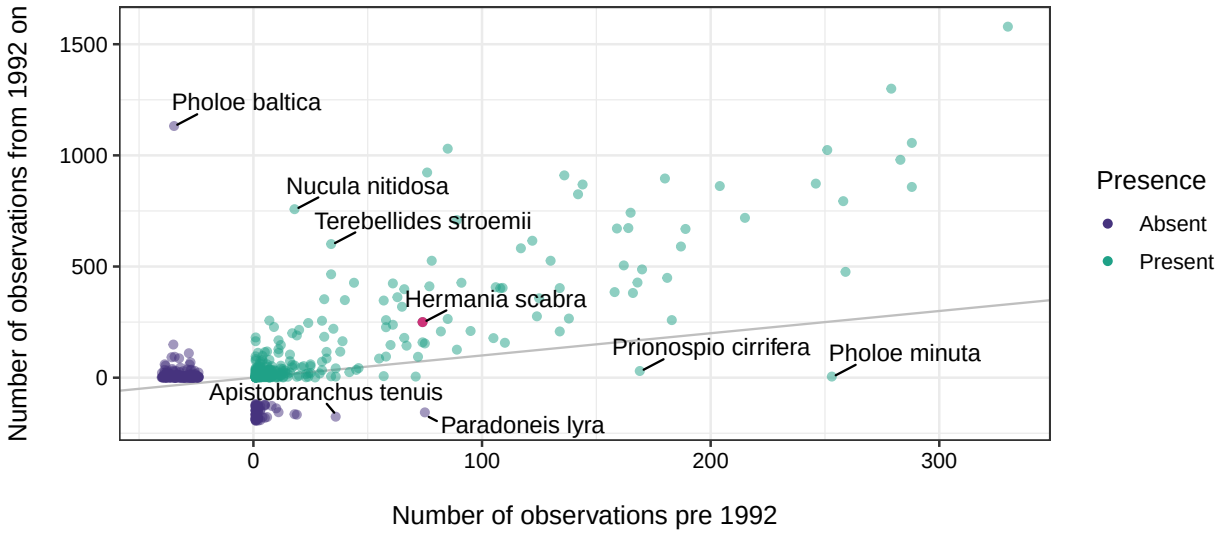
The way `geom_miss_point` works is to jitter the points around a fixed negative value. Here the colonising points are jittered around `x = -35` and the extinction points around `y = -175`. We can use that!

```
shark_labels_df <- shark_absent_df %>%
  # Replace all the NA values in the pre and post columns
  # This will be a bit approximate because of jitter
  tidyr::replace_na(list(pre = -35, post = -175)) %>%
  # Glue all the labels together, because now they will all work with ggrepel
  bind_rows(shark_present_df)
```

So now all the labels are in one dataframe again. Let's plot!

```
shark_label_gg <- shark_missing_gg +
    # Colour my target species
    geom_point(data = dplyr::filter(shark_gg_df, scientificName == species),
      col = highlight_colour) +
    # ggrepel only works for non-missing data; label species of interest
    ggrepel::geom_text_repel(data = shark_labels_df,
      aes(label = scientificName),
      # Avoids ambiguity or overlapping with manual labels
      nudge_x = 15, nudge_y = 15,
      # Point padding stops lines obscuring points
      # min.segment.length is the shortest line it will draw;
      # I made it lower than the default to get lines for nearly all the points
      point.padding = 0.25, min.segment.length = 0.2)

shark_label_gg
```



Further options

- I did not include how to make the histogram of observed species by year from earlier in my talk. If you are new to tidyverse, why not see if you can `select` `unique` combinations of `scientificName` and `year` to do that?
- I left the latitude and longitude columns in shark_df in case you wanted to play with these data using techiques you've learnt from the mapping workshop.