Wild recipes

Mike Page

2025-10-01

Table of contents

Welcome

The title for this book is inspired by Russ Roberts book, Wild Problems. In this books he describes a class of problem he calls 'wild problems'. These are big life decisions (e.g., should I move country) where an algorithmic approach to solving them, such as using a cost-benefit analysis, often fails. Instead he proposes a new framework to tackle these problems. (I will leave the curious reader to discover more on their own)

Much like the 'wild problems' discussed in Roberts' book, the R programmer also faces a class of coding problem which could be described as wild. That is, a set of problems found outside of a controlled environment such as a classroom or textbook and instead found in an environment which is uncontrolled and wild. Here, the examples found elsewhere often fail, or require a more complex workaround. This could be due to underlying bugs in R or its libraries, poor documentation, or quite simply and most often the case, the complexity of the problem space you are working in doesn't map easily to materials found elsewhere.

This book is a collection of recipes to some of these wild coding problems I've experienced in my work as an R programmer. It is by no means exhausative. Nor is it entriely unique. Indeed, a plethora of resources already exist in this area and this book even rehashes some of these (e.g., Stack Overflow, Posit Community, etc.). This book has primarily been created as a means for myself to document problems I've faced and the wild recipes I've implemented along the way. My hope is these recipes might also help you too.

Book structure

This book is not designed to be read from cover to cover. Instead it is written as a collection of individual recipes. Inspired by Tidy design principles, each recipe can be read in isolation and will cover:

- What is the problem?
- What is an example?
- What is a solution?

Occasionally, recipes will also cover:

- What other solutions exist?

Part I

ggplot2

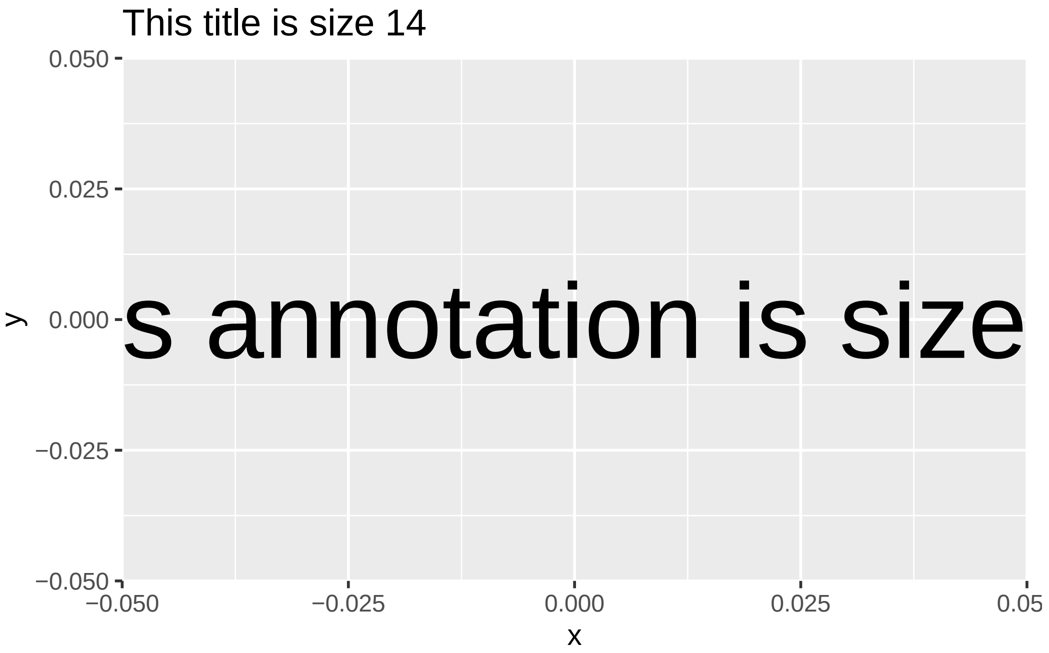
1 Annotations: text size

1.1 What is the problem?

When adding a text annotation to a plot with `ggplot2::annotate()`, the size of the text in the annotation does not match other elements on the plot despite setting the `size` argument to equal values.

1.2 What is an example?

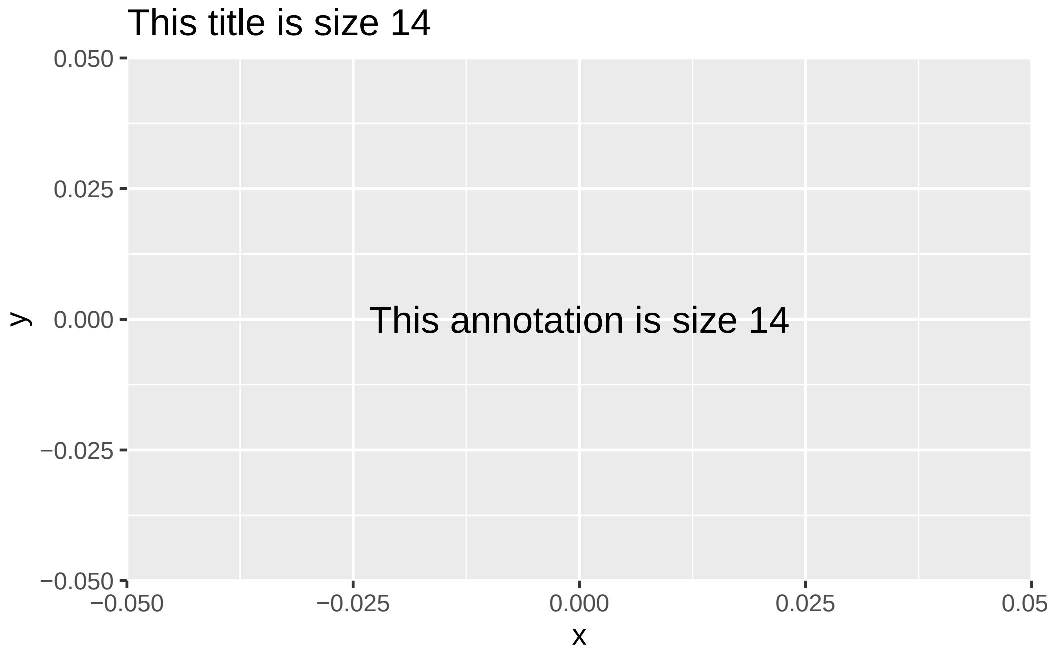
```r
library(ggplot2)

ggplot() +
  ggtitle("This title is size 14") +
  theme(plot.title = element_text(size = unit(14, "pt"))) +
  annotate(
    "text",
    label = "This annotation is size 14",
    x = 0, y = 0,
    size = unit(14, "pt")
  )
```



1.3 What is a solution?

To align the sizes of annotations and other elements on the plot, annotation sizes must be divided by `.pt`:

```r
ggplot() +
  ggtitle("This title is size 14") +
  theme(plot.title = element_text(size = unit(14, "pt"))) +
  annotate(
    "text",
    label = "This annotation is size 14",
    x = 0, y = 0,
    size = unit(14, "pt") / .pt # divide by .pt
  )
```



This works because `annotate()` calculates font size by multiplying the specified size by the global variable `.pt` (equal to `2.845276`). See this Stack Overflow post for more information.

2 Annotations: infinite positions

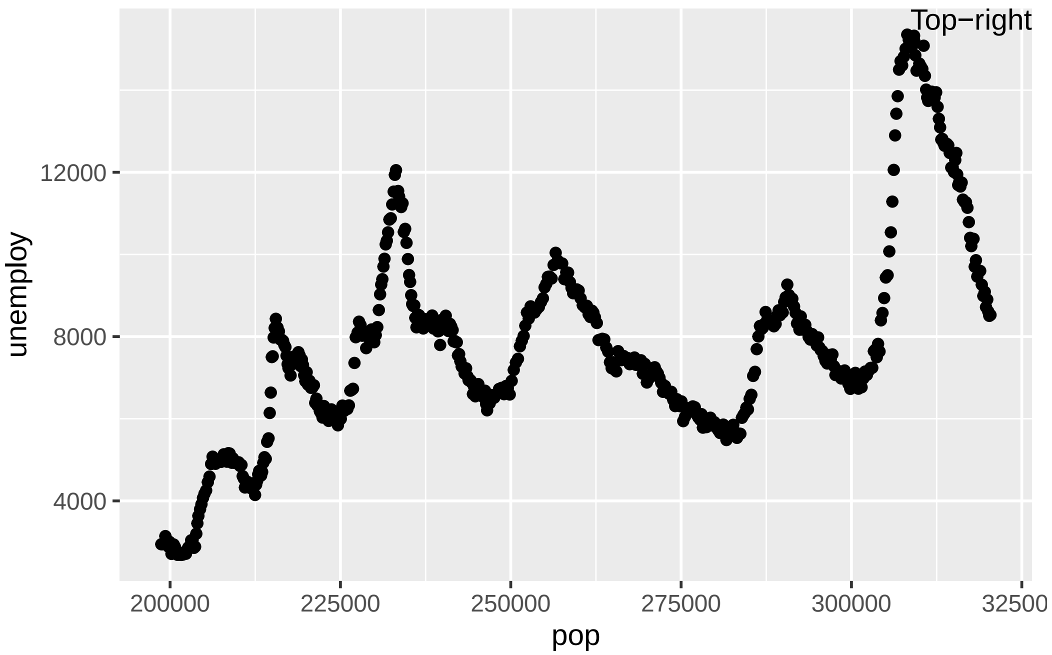
2.1 What is the problem?

To position annotations at the edge of a plot, the values `Inf` and `-Inf` can be passed to the positioning aesthetics (e.g., `x`) of `ggplot2::annotation()`. This technique fails for scales that are of class `Date`.

2.2 What is an example?

It is useful to first see how we can position annotations on scales which aren't dates. For example, using the built-in economics dataset of ggplot2, we can postion an annotation in the top-right corner of the plot like so:

```r
library(ggplot2)

ggplot(economics, aes(x = pop, y = unemploy)) +
  geom_point() +
  annotate(
    "text",
    label = "Top-right",
    vjust = 1, hjust = 1, # Prevent text being chopped
    x = Inf, y = Inf
  )
```



When we try the same approach to a scale that uses dates, we get an error:

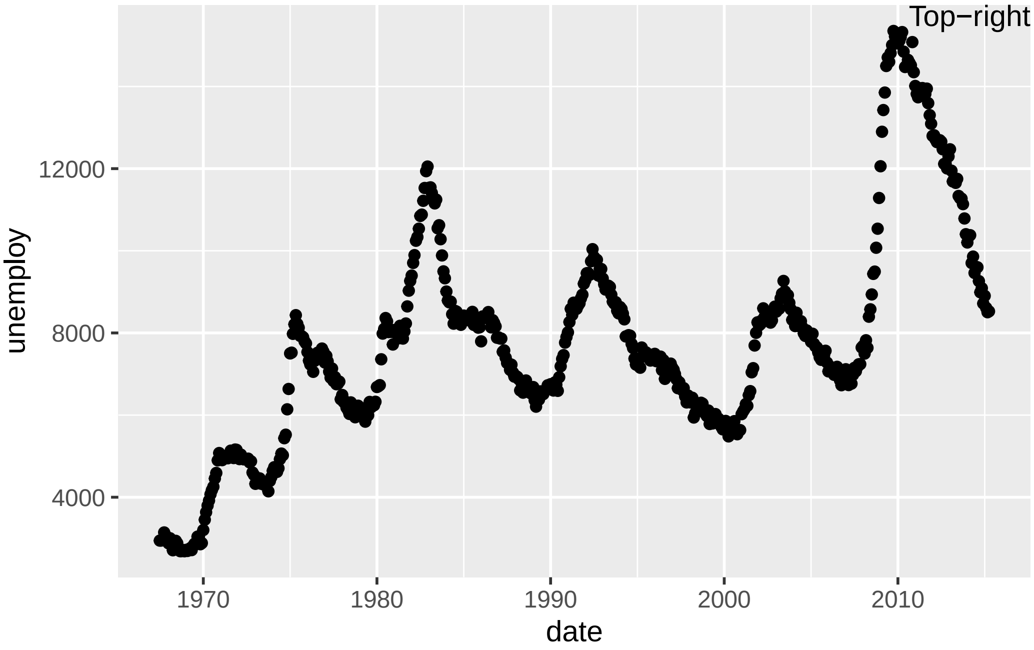
```
ggplot(economics, aes(x = date, y = unemploy)) +
  geom_point() +
  annotate(
    "text",
    label = "Top-right",
    vjust = 1, hjust = 1, # Prevent text being chopped
    x = Inf, y = Inf
  )
```

```
Error in `self$trans$transform()`:
! `transform_date()` works with objects of class <Date> only
```

2.3 What is a soltuion?

To plot an annotation at the edge of a scale of class `Date`, you should change the class of `Inf` to a `Date` class:

```
ggplot(economics, aes(x = date, y = unemploy)) +
  geom_point() +
  annotate(
    "text",
    label = "Top-right",
    vjust = 1, hjust = 1, # Prevent text being chopped
    x = structure(Inf, class = "Date"), y = Inf
  )
```



See this GitHub issue for more information.

3 Inspecting ggplot2 objects

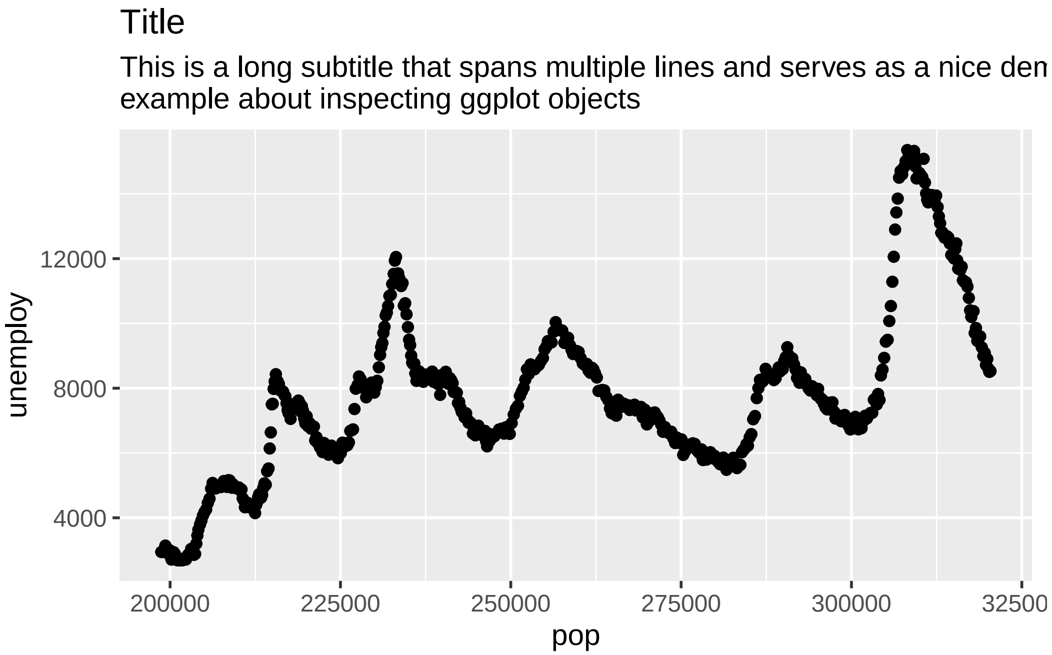
3.1 What is the problem?

After creating a ggplot2 object, it can be useful to inspect the object created, for example, to change the behaviour of functions using that object, or for writing unit tests. Searching through the documentation index reveals no help, and printing the name of the plot to the console just calls the default print method, re-printing the plot.

3.2 What is an example?

Let's create and print a plot with a long subtitle that spans multiple lines:

```r
library(ggplot2)

example <- ggplot(economics, aes(x = pop, y = unemploy)) +
  geom_point() +
  labs(
    title = "Title",
    subtitle = paste0(
      "This is a long subtitle that spans multiple lines and serves as a nice demonstratatio
      "\n",
      "example about inspecting ggplot objects"
    )
  )

example
```



Now, we want to write a wrapper function to `ggsave()` that changes the `height` parameter of the output plot, depending on whether a multiline subtitle has been detected.

3.3 What is a solution?

Internally, a ggplot object is just stored as a list, and can be inspected with `str()` like most objects:

```
typeof(example)
```

```
[1] "list"
```

```
str(example)
```

```
List of 9
 $ data       : spc_tbl_ [574 x 6] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
  ..$ date    : Date[1:574], format: "1967-07-01" "1967-08-01" ...
  ..$ pce     : num [1:574] 507 510 516 512 517 ...
  ..$ pop     : num [1:574] 198712 198911 199113 199311 199498 ...
  ..$ psavert : num [1:574] 12.6 12.6 11.9 12.9 12.8 11.8 11.7 12.3 11.7 12.3 ...
  ..$ uempmed : num [1:574] 4.5 4.7 4.6 4.9 4.7 4.8 5.1 4.5 4.1 4.6 ...
  ..$ unemploy: num [1:574] 2944 2945 2958 3143 3066 ...
 $ layers     :List of 1
  ..$ :Classes 'LayerInstance', 'Layer', 'ggproto', 'gg' <ggproto object: Class LayerInstanc
    aes_params: list
    compute_aesthetics: function
    compute_geom_1: function
    compute_geom_2: function
    compute_position: function
    compute_statistic: function
    computed_geom_params: list
    computed_mapping: uneval
    computed_stat_params: list
    constructor: call
    data: waiver
    draw_geom: function
    finish_statistics: function
    geom: <ggproto object: Class GeomPoint, Geom, gg>
        aesthetics: function
        default_aes: uneval
        draw_group: function
        draw_key: function
        draw_layer: function
        draw_panel: function
        extra_params: na.rm
        handle_na: function
        non_missing_aes: size shape colour
        optional_aes:
        parameters: function
        rename_size: FALSE
        required_aes: x y
        setup_data: function
        setup_params: function
        use_defaults: function
        super:  <ggproto object: Class Geom, gg>
    geom_params: list
    inherit.aes: TRUE
    layer_data: function
    map_statistic: function
    mapping: NULL
    position: <ggproto object: Class PositionIdentity, Position, gg>
        compute_layer: function
        compute_panel: function
```

```
       required_aes: 
       setup_data: function
       setup_params: function
       super:  <ggproto object: Class Position, gg>
   print: function
   setup_layer: function
   show.legend: NA
   stat: <ggproto object: Class StatIdentity, Stat, gg>
       aesthetics: function
       compute_group: function
       compute_layer: function
       compute_panel: function
       default_aes: uneval
       dropped_aes: 
       extra_params: na.rm
       finish_layer: function
       non_missing_aes: 
       optional_aes: 
       parameters: function
       required_aes: 
       retransform: TRUE
       setup_data: function
       setup_params: function
       super:  <ggproto object: Class Stat, gg>
   stat_params: list
   super:  <ggproto object: Class Layer, gg>
$ scales     :Classes 'ScalesList', 'ggproto', 'gg' <ggproto object: Class ScalesList, gg>
   add: function
   clone: function
   find: function
   get_scales: function
   has_scale: function
   input: function
   n: function
   non_position_scales: function
   scales: list
   super:  <ggproto object: Class ScalesList, gg>
$ mapping    :List of 2
 ..$ x: language ~pop
 .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv> 
 ..$ y: language ~unemploy
 .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv> 
 ..- attr(*, "class")= chr "uneval"
```

```
$ theme      : list()
$ coordinates:Classes 'CoordCartesian', 'Coord', 'ggproto', 'gg' <ggproto object: Class Coor
   aspect: function
   backtransform_range: function
   clip: on
   default: TRUE
   distance: function
   expand: TRUE
   is_free: function
   is_linear: function
   labels: function
   limits: list
   modify_scales: function
   range: function
   render_axis_h: function
   render_axis_v: function
   render_bg: function
   render_fg: function
   setup_data: function
   setup_layout: function
   setup_panel_guides: function
   setup_panel_params: function
   setup_params: function
   train_panel_guides: function
   transform: function
   super:  <ggproto object: Class CoordCartesian, Coord, gg>
$ facet      :Classes 'FacetNull', 'Facet', 'ggproto', 'gg' <ggproto object: Class FacetNull
   compute_layout: function
   draw_back: function
   draw_front: function
   draw_labels: function
   draw_panels: function
   finish_data: function
   init_scales: function
   map_data: function
   params: list
   setup_data: function
   setup_params: function
   shrink: TRUE
   train_scales: function
   vars: function
   super:  <ggproto object: Class FacetNull, Facet, gg>
$ plot_env   :<environment: R_GlobalEnv>
```

```
 $ labels     :List of 4
  ..$ title   : chr "Title"
  ..$ subtitle: chr "This is a long subtitle that spans multiple lines and serves as a nice
  ..$ x       : chr "pop"
  ..$ y       : chr "unemploy"
 - attr(*, "class")= chr [1:2] "gg" "ggplot"
```

Here, we can see and access all the elements that make up the plot (e.g., scales, data, etc.). The subtitle can be accessed like so:

```
subtitle <- example$labels$subtitle
```

We could then write a wrapper function to detect whether our subtitle is multiline and change the `height` of `ggsave()` accordingly:

```
ggsave_multline <- function(...) {
  height <- ifelse(grepl("\\n", subtitle), 11, 10)
  ggsave(height = height, ...)
}
```

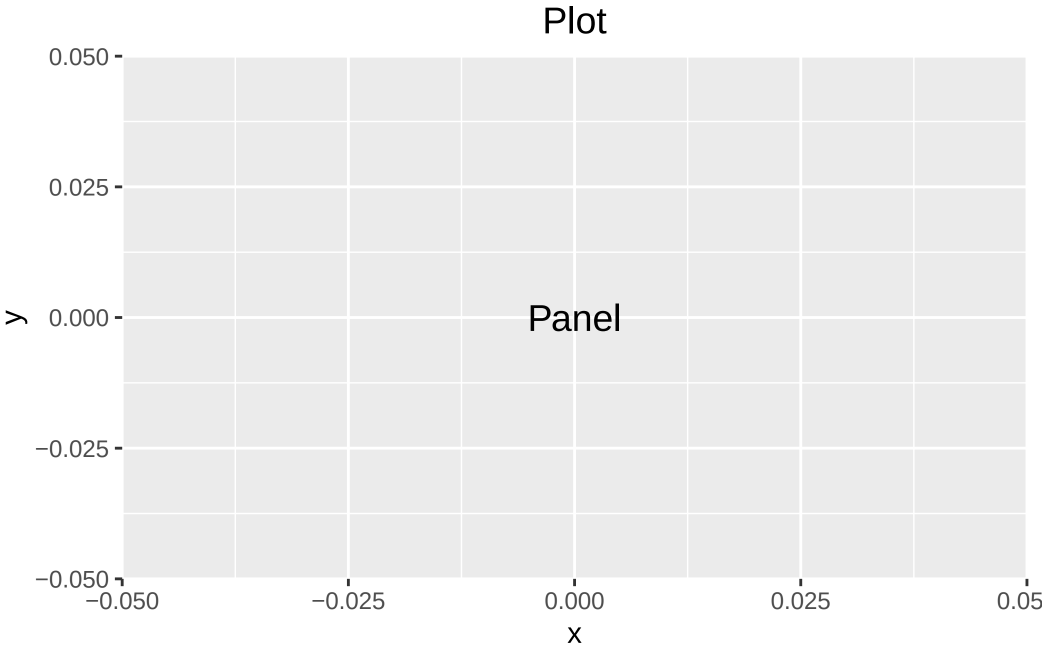
3.4 What other solutions exist?

ggplot2 provides a set of functions to render plot objects, which can also be used to inspect the underlying data and panel object. These functions do not appear in the documentation index however, and so are not easily discoverable. For a deeper diver on these functions and the internals of ggplot objects, see this chapter in the book “ggplot2: Elegant Graphics for Data Analysis (3e)”.

4 Panel sizes

4.1 What is the problem?

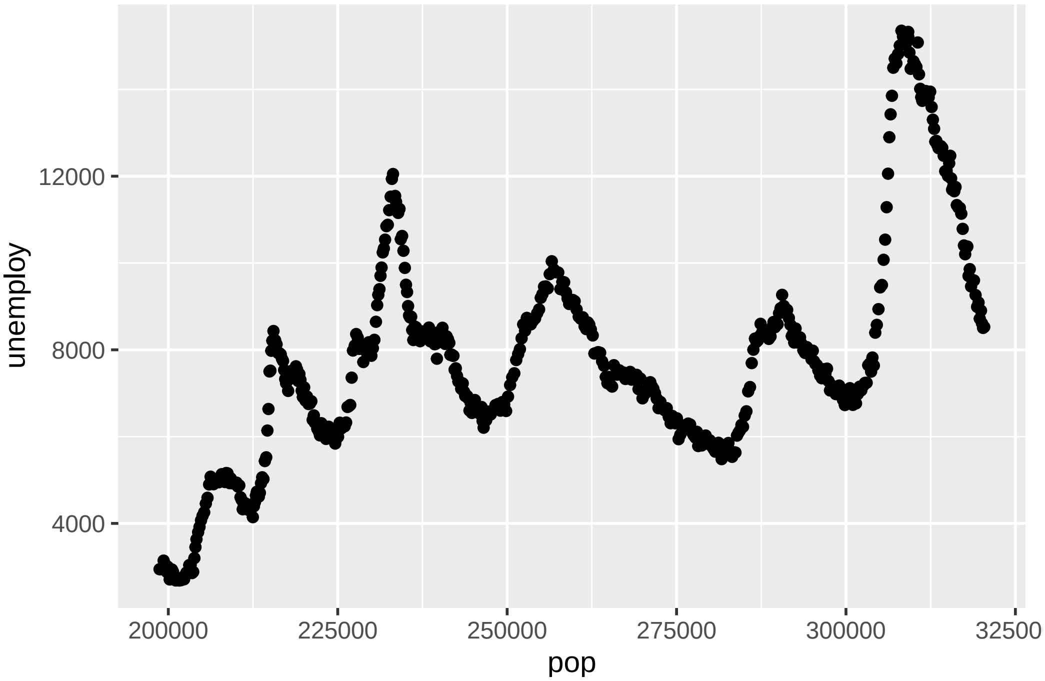
There is no default method to set the panel size of a plot in ggplot2, only a method to set the plot size using the `width` and `height` paramaeters of `ggplot2::ggsave()`. The panel refers to the inner plotting window that contains the data, and the plot refers to the whole plotting window that contains both the panel and all other elements (e.g., legeneds, labels, etc.):



4.2 What is a solution?

The `ggh4x::force_panelsizes()` function can be used to coerce a single panel to a set size:

```r
library(ggplot2)

ggplot(economics, aes(x = pop, y = unemploy)) +
  geom_point() +
  ggh4x::force_panelsizes(
    rows = unit(8, "cm"),
    cols = unit(12, "cm")
  )
```



Part II

Utilities

5 Consecutive true values

5.1 What is the problem?

For a column of logical values in a data frame, count of the number of consecutive `TRUE` values at the given point in the column. `NA` values should return `NA` and reset the count.

5.2 What is an example?

For the data frame:

```
df <- tibble::tibble(x = c(TRUE, TRUE, NA, TRUE, FALSE, TRUE, TRUE, TRUE))
```

The following data frame should be returned:

```
# A tibble: 8 x 2
  x     consecutive_trues
  <lgl>             <dbl>
1 TRUE                  2
2 TRUE                  1
3 NA                   NA
4 TRUE                  1
5 FALSE                 0
6 TRUE                  3
7 TRUE                  2
8 TRUE                  1
```

5.3 What is a solution?

We can leverage the `dplyr::consecutive_id()` function to create unique group id's, and then create sequences along these id's:

```r
library(tibble)
library(dplyr)

df |>
  mutate(id = consecutive_id(x)) |>
  add_count(id) |>
  mutate(consecutive_trues = seq(n(), 1), .by = id) |>
  mutate(
    consecutive_trues = case_when(
      is.na(x) ~ NA_integer_,
      x ~ consecutive_trues,
      .default = 0
    )
  ) |>
  select(-id, -n)
```

```
# A tibble: 8 x 2
  x     consecutive_trues
  <lgl>             <dbl>
1 TRUE                  2
2 TRUE                  1
3 NA                   NA
4 TRUE                  1
5 FALSE                 0
6 TRUE                  3
7 TRUE                  2
8 TRUE                  1
```

To reverse the order of the count of `TRUE` values, that is, to count the number of trailing `TRUE` values, the order of the sequence can simply be switched.

5.4 What are alternate solutions?

For a generalised solution that works with any vector, outside the context of a data frame:

```r
consecutive_trues <- function(x) {
  result <- integer(length(x))
  current_total <- 0

  for (i in seq_along(x)) {
    if (is.na(x[i])) {
      result[i] <- NA_integer_
      current_total <- 0
    } else if (x[i]) {
      j <- i
      while (isTRUE(x[j])) {
        current_total <- current_total + 1
        j <- j + 1
      }
      result[i] <- current_total
      current_total <- 0
    } else {
      current_total <- 0
      result[i] <- current_total
    }
  }

  return(result)
}
```