Tutorial

Advances in Methods and Practices in Psychological Science
July-September 2023, Vol. 6, No. 3, pp. 1–30
© The Author(s) 2023
Article reuse guidelines: sagepub.com/journals-permissions
DOI: 10.1177/25152459231160103
www.psychologicalscience.org/AMPPS

Dynamic Data Visualizations to Enhance Insight and Communication Across the Life Cycle of a Scientific Project

Kristina Wiebels and David Moreau
School of Psychology & Centre for Brain Research, The University of Auckland, Auckland, New Zealand

Abstract
In scientific communication, figures are typically rendered as static displays. This often prevents active exploration of the underlying data, for example, to gauge the influence of particular data points or of particular analytic choices. Yet modern data-visualization tools, from animated plots to interactive notebooks and reactive web applications, allow psychologists to share and present their findings in dynamic and transparent ways. In this tutorial, we present a number of recent developments to build interactivity and animations into scientific communication and publications using examples and illustrations in the R language (basic knowledge of R is assumed). In particular, we discuss when and how to build dynamic figures, with step-by-step reproducible code that can easily be extended to the reader's own projects. We illustrate how interactivity and animations can facilitate insight and communication across a project life cycle—from initial exchanges and discussions in a team to peer review and final publication—and provide a number of recommendations to use dynamic visualizations effectively. We close with a reflection on how the scientific-publishing model is currently evolving and consider the challenges and opportunities this shift might bring for data visualization.

Keywords
research workflow, transparency, data analysis, data exploration, scientific communication, knowledge dissemination

Received 9/16/21; Revision accepted 2/8/23

Effective data visualization is one of the cornerstones of clear scientific communication (Friendly, 2008). Numerous guidelines have been written about what makes for good data visualization, from choosing the right type of graph (Doumont & Vandenbroeck, 2002) to using color wisely (Borland & Taylor, 2007) and adapting to one's intended audience (Rougier et al., 2014). Other recommendations have emphasized transparency in displaying statistical results, for example, with a shift from the once-ubiquitous bar plots to more comprehensive graphs that include various distribution properties (Allen et al., 2012; Newman & Scholl, 2012; Weissgerber et al., 2015).

Recent increases in data complexity and software capabilities have paved the way for more sophisticated ways of presenting findings. This is especially visible in popular scientific communication, in which a specific mode of presentation—dynamic data visualizations—has flourished. With a blend of animated and interactive features, dynamic data visualizations can be found in the classroom (Fawcett, 2018; Moreau, 2015), the news media (e.g., *The New York Times*), information campaigns by nonprofit organizations (e.g., Our World in Data), and TED talks. Many readers[1] would have seen Hans Rosling's most popular talk, *The Best Stats You've Ever Seen* (Rosling, 2006), in which he introduced multifaceted data in an easily digestible way using pointed animations to direct the audience's attention and stress important information. The dynamic style of Rosling's presentation has since gained traction, and many writers and speakers from academia to government and industry have embraced the trend. Clear, powerful visualizations have become especially important given the growing need to accurately inform populations about high-stake

Corresponding Author:
David Moreau, School of Psychology & Centre for Brain Research, The University of Auckland, Auckland, New Zealand
Email: d.moreau@auckland.ac.nz

problems that require collective action, such as global warming or pandemics.[2]

Despite these recent advances, dynamic data visualizations remain underused in the communication of findings among scientists, for a number of reasons. First, scientists have historically relied on print-only journals to disseminate their findings, and many of the current practices are based on obsolete standards—in the traditional publishing system, figures had to be static to be rendered in print. Second, dynamic visualizations can be challenging to describe accurately given that captions need to capture a vast number of frames or relationships between numerous potential variables, especially in rich data sets. In contrast, figures and graphs in peer-reviewed journals often present a snapshot of the findings, with the goal to portray the most important or most impressive findings. Finally, a number of recommendations have been made to improve plots and figures in scientific publications (Kelleher & Wagener, 2011), and these have often emphasized simplicity over seemingly more advanced, but perhaps less accessible, renderings (e.g., three-dimensional plots). Given these potential limitations, we first present the rationale for using dynamic visualizations in scientific projects.

The Case for Dynamic Visualizations

Effective data visualizations often rely on clarity and simplicity (Few, 2004; Kelleher & Wagener, 2011; Midway, 2020), yet scientific data have become increasingly complex over the last few decades (Cordero et al., 2016; Kamath, 2001). One key feature of effective dynamic data visualizations is their ability to pack rich information into relatively simple displays (Blok, 2005) via two components that can be found in most recent, eye-catching content: interactivity and animation.

Interactive content enables active exploration of data features, for example, by selecting a subset of observations, focusing on specific variables, or displaying particular values or statistics on data-point mouse-overs. Interactivity may help with collaborative data exploration (Isenberg et al., 2011); for example, a team member might have questions about the impact of particular analytic choices, such as the influence of an outlier on a model or statistic. Because new visualizations need to be created to explore each question, this type of conversation might typically result in back-and-forth communication over days or weeks and because of delays inherent to this process, in fewer research questions being explored altogether. Interactive visualizations provide an easy way to address queries in an immediate manner without the need for additional visualizations and thus can greatly streamline this process. In many cases, interactivity can also provide the means to transparently disclose the impact of analytical choices in statistical analysis (Ospina et al., 2014) and could serve as a valuable tool to teach statistical concepts to trainees (Xie, 2013).

In contrast to interactive plots, in which the user is actively exploring variables and relationships to better understand the data at hand and their inherent features, animations are built to be consumed passively. Animated content is content that is dynamic either across time—for example, showing the relationship between variables across hours, days, or years—or across iterations of another variable (e.g., participants, experimental conditions, algorithms). This type of visualization can be particularly useful when presenting variable change (Weiss et al., 2002), illustrating computational algorithms and their outcomes (Kerren & Stasko, 2002), or displaying the results of simulations (Moreau, 2015). The key component is that the variable that is being iterated over does not need to be displayed as another dimension with an additional axis (e.g., three-dimensional plot) or with another plot altogether for each value (i.e., faceting). Rather, the relationship is implicitly and effortlessly inferred from the natural flow of the animation, with the user being introduced to additional content in a passive manner (Rolfes et al., 2020).

In this tutorial, we show how to convert static plots into dynamic ones in the R language (R Core Team, 2020).[3] With various options and implementations, we first discuss how to build interactivity into scientific plots—a feature especially interesting at the exploration stage of a project, for example, to discover relationships among variables. We then focus on animations, or how to transition from static to live figures, a property particularly useful for the presentation of findings. Finally, we propose to combine interactive and animated features via Shiny apps that can be personalized depending on individual needs and preferences. Blending interactive and animated features facilitates the dissemination of findings in the scientific community in a transparent and user-friendly way.

Disclosures

All materials (data, scripts) of this tutorial can be found at osf.io/fwy8j. The OSF repository includes an RMarkdown file with all code that is used in this tutorial, the corresponding html file including all dynamic figures, code for two Shiny apps (one full, one simplified version), and two data sets. We designed this tutorial to be accessible to novices, but we do assume basic knowledge of R and *ggplot2* (Wickham, 2016). For researchers who are not familiar with R and its *ggplot2* visualization capabilities, see Nordmann et al. (2021). Familiarity with *shiny* (Chang et al., 2022) is helpful for the final section of this tutorial but not necessary.

R Packages Enabling Dynamic Content

In the last few years, several R packages have been created that can be used to make plots either interactive, animated, or both. In this tutorial, we use the following packages: *ggiraph* (Gohel & Skintzos, 2022), *gganimate* (Pedersen & Robinson, 2020), *plotly* (Sievert, 2020), and *shiny* (Chang et al., 2022). Note that although for each example used in this tutorial we picked one package to add dynamic content, in most cases, at least one of the other packages can be used to get a similar result. We provide a reproducible environment with the *renv* package (Ushey, 2021), which allows restoring the state of this project from the renv.lock file provided at osf.io/fwy8j. For a more extensive discussion of reproducible computational environments using R, see Wiebels and Moreau (2021).

ggiraph and *gganimate* are packages built on top of *ggplot2*. *ggiraph* enables interactive content by adding tool tips, hover effects, and JavaScript actions to static plots via adapted geoms, such as `geom_point_interactive()` instead of `geom_point()`, together with the aesthetics `tooltip`, `data_id`, and `onclick`. *gganimate* focuses on animations instead of interactivity. In contrast to *ggiraph*, the geom layers remain unchanged; instead, *gganimate* introduces a variety of new grammar classes, such as `transition_states()` and `transition_time()`, that are added to static plots to specify how a plot should change with time. Animated plots can be rendered in Markdown or saved in several file formats, including gif images and a variety of video formats. For more information, see Gohel (2023) and Pederson and Robinson (2022).

Plotly is a computing company that provides visualization tools and products for a variety of programming languages, including R, Python, and Julia. The R package *plotly* can be used to create interactive and animated content and does so via its JavaScript graphing library, plotly.js. The plots can be created using either standalone code or the `ggplotly()` wrapper function, which takes a ggplot object, extracts all features, and redraws it with plotly.js. For more information on *plotly*, see *Plotly R Open Source Graphing Library* (n.d.).[4]

Finally, *shiny* allows building interactive apps that can be deployed locally (*Sharing Apps to Run Locally*, 2014) or on the web (*Deploying Shiny Apps to the Web*, 2017), be embedded in RMarkdown documents, or used to build dashboards. *shiny* provides user interface functions that convert R code into the HTML, CSS, and JavaScript functions necessary for the web content and a style of programming called "reactive programming," which keeps track of dependencies and automatically updates the code when any input changes. *shiny* can be used by itself to make content interactive, or it can be combined with other packages that enable dynamic visualization. For more information, see https://shiny.rstudio.com.

Preparations

To follow this tutorial, you will need to have R installed on your computer. If you do not have R installed and want to install it locally, follow the instructions at r-project.org. We recommend using RStudio (RStudio Team, 2020), an integrated development environment for the R language, which can be downloaded from rstudio.com.

Before starting the tutorial, download the data files (`imagination_study.csv` and `intervention_study.csv`) from osf.io/fwy8j to a chosen location on your computer. These files contain data from a published study on future imagination and a simulated intervention-study data set, respectively. Open RStudio, go to the folder of the downloaded files, and create a new R Script. Alternatively, you can download the RMarkdown file from the OSF repository (`DynamicVisualizations.Rmd`) and simply follow and execute the code that is contained within.

Before creating any plots, you need to install and load the packages needed for this tutorial. A note for Mac users: XQuartz needs to be available on your computer for the *ggiraph* install to succeed (you can download the software from https://www.xquartz.org/). Install the packages that you do not have on your computer yet:

```
# Install packages       (code
  snippet 1)

install.packages("knitr") # needed
  for knitting RMarkdown files
install.packages("tidyverse")
install.packages("ggridges")
install.packages("ggiraph")
install.packages("gganimate")
install.packages("plotly")
install.packages("shiny")
install.packages("transformr") #
  needed for gganimate
install.packages("gifski) # needed
  for rendering gganimate plots
```

We use some *tidyverse* functions to manipulate the data sets and *ggplot2* (which is part of the *tidyverse*) and *ggridges* to build the static plots. All other packages are used to create dynamic content. In case you encounter any issues with the installation of these packages, we provide a reproducible environment that contains all packages and their versions. To make use of this environment, download the `renv.lock` file from osf.io/fwy8j, install *renv* (`install.packages("renv")`), and then use the command `renv::restore()`.

Once all packages have successfully installed, you can load them:

```
# Load packages        (code snippet 2)

library(tidyverse)
library(ggridges)
library(ggiraph)
library(gganimate)
library(plotly)
library(shiny)
library(transformr)
library(gifski)
```

We also set up a custom theme for the plots so that we do not have to add these specifications to every single plot:

```
# Set up custom theme        (code
  snippet 3)

custom_theme <-
  list(theme_classic(),
     scale_color_manual(
       values = c("#eeaa7b",
         "#66b9bf", "#94618e")),
     scale_fill_manual(
       values = c("#eeaa7b",
         "#66b9bf", "#94618e")))
```

This code chunk specifies that we want to use the classic *ggplot2* theme (for an overview of available themes, see Wickham et al., n.d.) and three colors we use to differentiate between the conditions/groups in our data sets.

Finally, we need to load the data:

```
# Load data       (code snippet 4)

imagination_data <- read_csv
  ("imagination_study.csv")
intervention_data <- read_csv
  ("intervention_study.csv")
```

Example 1: future-imagination data set

The future-imagination data set is a subset of a published study on phenomenological differences between imagining future events relative to remembering past events (Wiebels et al., 2020; details can be found at osf.io/xqm5n/). Twenty participants remembered personal past events and imagined possible future events. The time it took to bring these events to mind was measured using button-press response times, and participants recorded in how much detail they remembered/imagined these events. Each past and future event was brought to mind three times during the experiment to test how response times and detail ratings changed across time points.

Let us have a look at the data:

```
# Inspect data       (code snippet 5)

imagination_data
```

`Time_point` in this data set is a factor, but we can see that it was read in as a numerical variable, so we need to change its class before we start:

```
# Convert Time_point into
  factor      (code snippet 6)

imagination_data$Time_point <-
  as.factor(imagination_data$Time_
    point)
```

Using this data set, the plots we create throughout this tutorial address the following research questions:

Research Question 1: Does it take longer to imagine future events compared with remembering past events?

Research Question 2: Do future events become faster to imagine with repetition?

Research Question 3: Can we predict how long it takes people to imagine future events based on how fast they remember past events?

Example 2: intervention data set

The second data set is a simulated study comprising data from 40 participants—20 in each of two groups (intervention and control groups)—with measurements taken once a week for the duration of 20 weeks. The intervention took place from Week 5 to Week 16, so the data set includes a 4-week baseline and a 4-week postintervention phase. There are two outcome variables: performance on a task that is targeted by the intervention and alertness level on the days of testing.

Let us look at the structure of this data set:

```
# Inspect data       (code snippet 7)

intervention_data
```

Using this data set, we create plots that address the following research questions:

Research Question 4: How does performance on the task change over time?

Research Question 5: Does the intervention elicit differences in performance between the groups?

Research Question 6: Can people's performance on the task be predicted from their alertness level?

All questions for this tutorial have been designed to illustrate the potential and the advantages of dynamic visualizations. In the remainder of this section, we provide static plots that could be used to explore these six questions. In the following sections, we then demonstrate how interactive or animated features can be added to create dynamic visualizations.

Static Plots

Example 1: future-imagination data set

Research Question 1: Does it take longer to imagine future events compared with remembering past events?

To address the first question, we construct a violin plot showing response times for remembering past and imagining future events. We also display box plots and individual data points within the violins.

```
# Static violin plot      (code
    snippet 8)

# For this plot, we are only using
  data from time point 1
# and we reverse the factor levels of
  Condition
# so that the past condition is on
  the left hand side
ggplot(imagination_data %>%
  filter(Time_point == 1),
     aes(x = fct_rev(Condition), y =
       RT)) +
  # Violins
  geom_violin(trim = FALSE, alpha =
    0.6, aes(fill = Condition)) +
  # Boxes
  geom_boxplot(width = 0.1) +
  # Individual data points
  geom_point(alpha = .4) +
  xlab("Condition") +
  ylab("Response time (ms)") +
  custom_theme +
  theme(legend.position = "none")
```

We can see in the resulting Figure 1 that novel future events take longer to bring to mind than past events.

Research Question 2: Do future events become faster to imagine with repetition?

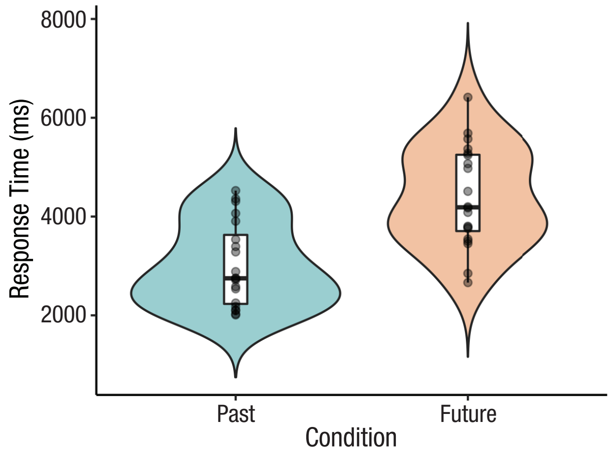


Fig. 1. Violin plot displaying response times for remembering past events and imagining future events.

To address the second question, we construct a violin plot with boxes and individual data points again, this time only for the future events, but for each time point separately. We also connect the individual data points with lines to highlight each person's change in response time across time points.

```
# Static violin plot with lines
  (code snippet 9)

# For this plot, we are only using
  data from the future condition
ggplot(imagination_data %>%
  filter(Condition == "Future"),
     aes(x = Time_point, y = RT)) +
  # Violins
  geom_violin(trim = FALSE, alpha =
    0.6, fill = "#94618e") +
  # Boxes
  geom_boxplot(width = 0.1) +
  # Individual data points
  geom_point(alpha = .4) +
  # Individual lines
  geom_line(aes(group = ID), alpha =
    .2, linetype = "dashed") +
  xlab("Time point") +
  ylab("Response time (ms)") +
  custom_theme
```

Figure 2 shows that future events are brought to mind faster when they are imagined a second/third time. This effect is very consistent across people, as indicated by the dashed lines.

Research Question 3: Can we predict how long it takes people to imagine future events based on how fast they remember past events?

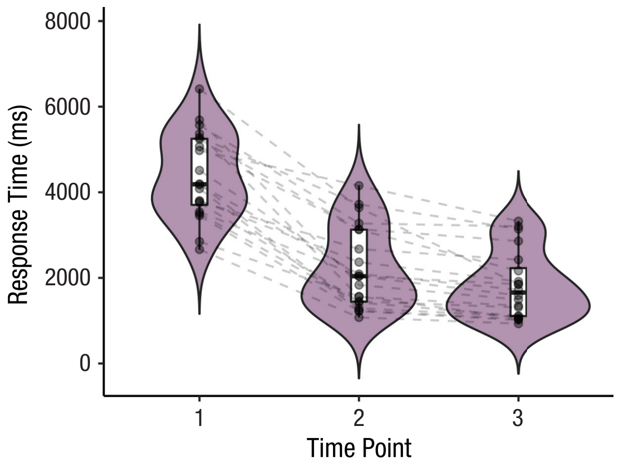


Fig. 2. Violin plot displaying response times for imagining future events across time points.

For the last question on this data set, we construct a scatter plot with a trend line for each time point. The sizes of the points correspond to the mean detail rating made by each person for these events. Before we create this plot, we create a wider version of this data set to have separate columns for the response times in the future and past conditions, respectively. We also compute mean detail ratings.

```
# Create wide version and means of
  dataset    (code snippet 10)

imagination_data_wide <- imagination_
  data %>%
    pivot_wider(names_from = Condition,
      values_from = c(RT, Detail))

imagination_data_wide$Detail_Mean <-
  rowMeans(imagination_data_wide
    [, 5:6])
```

Using this data set, let us create the scatter plot:

```
# Static scatter plot      (code
  snippet 11)

ggplot(imagination_data_wide,
       aes(x = RT_Past, y = RT_Future,
         color = Time_point)) +
  # Individual points
  geom_point(alpha = .4, aes(size =
    Detail_Mean)) +
  # Trendlines
  geom_smooth(method = "lm") +
  # Distributions
  geom_rug(alpha = .4) +
  xlab("Response time past (ms)") +
  ylab("Response time future (ms)") +
  custom_theme +
  guides(color = guide_legend(title =
    "Time point"),
      size = guide_legend(title =
        "Detail rating"))
```

Figure 3 shows that past and future response times are positively correlated for all three time points.

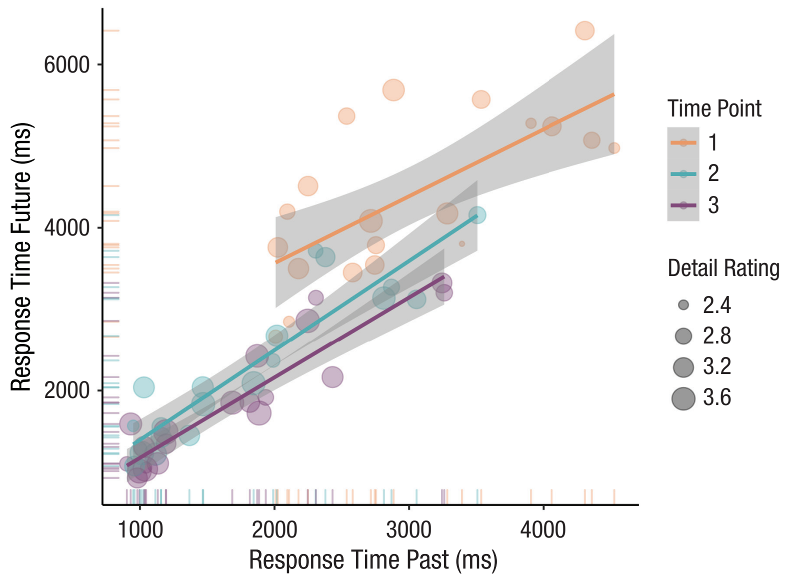


Fig. 3. Scatter plot displaying the correlation between past-event and future-event response times for each time point.

Example 2: intervention data set

Research Question 4: How does performance on the task change over time?

To address the first question for this data set, we construct a line graph. Two bold lines in Figure 4 indicate group-average performance to see whether task performance might differ between groups. In addition, we display each person's individual performance.

```
# Static line graph       (code
  snippet 12)

ggplot(intervention_data, aes(x =
  Week, y = Task, color = Group)) +

  # Gray rectangle to highlight
    intervention phase
  annotate(
    "rect",
    xmin = 4.5,
    xmax = 16.5,
    ymin = 0,
    ymax = Inf,
    alpha = 0.1,
    fill = "gray45"
  ) +
  # Label for rectangle
  annotate(
    geom = "text",
    x = 10.5,
    y = max(intervention_data$Task) + 1,
    label = "Intervention",
    size = 3.5,
    color = "gray35"
  ) +
  # Individual lines
  geom_line(aes(group = ID), size =
    .5, alpha = .15) +
  # Group average lines
  stat_summary(geom = "line", fun =
    "mean") +
  # Group average points
  stat_summary(geom = "point", fun =
    "mean") +
  ylab("Task performance") +
  custom_theme +
  theme(legend.position = "top")
```

Figure 4 shows that performance on the task remained relatively stable across the 20 weeks for the control group, whereas performance gradually improved from shortly after the onset until the end of the intervention for the intervention group.

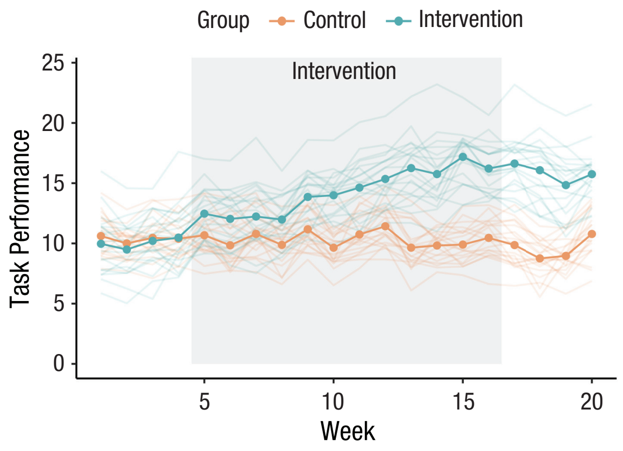


Fig. 4. Line graph displaying group-average and individual task performance over time.

These data and the divergence in task performance between groups across time can also be nicely visualized with a ridgeline plot (Fig. 5):

```
# Static ridgeline plot     (code
  snippet 13)

# We flip the y-axis, so that Week 1
  is at the top
ggplot(intervention_data,
    aes(
      x = Task,
      y = fct_rev(as.factor(Week)),
      color = Group,
      fill = Group
    )) +
  geom_density_ridges(alpha = .4) +
  xlab("Task Performance") +
  ylab("Week") +
  custom_theme
```

If the focus is on individual performance and its change across time, we can also construct a heat map:

```
# Static heatmap        (code
  snippet 14)

ggplot(intervention_data, aes(
  x = Week,
  y = ID,
  group = Group,
  fill = Task
)) +
  # Tiles
  geom_tile(color = "white", size =
    0.35) +
```

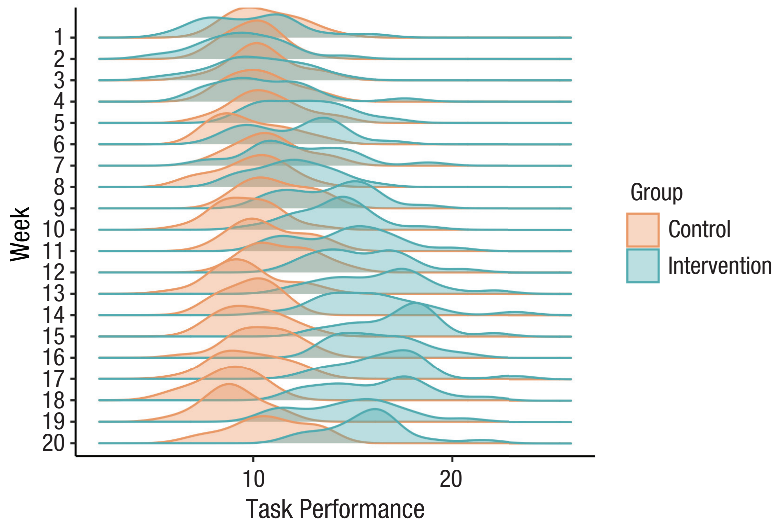


Fig. 5. Ridgeline plot displaying task performance for each group over time.

```
# Lines indicating the start and
  end of the intervention
geom_vline(xintercept = c(4.5,
  16.5), col = "black") +
# Label for start line
geom_label(
  x = 4.5,
  y = 42,
  label = "Start of intervention",
  size = 3.5,
  fill = "white",
  label.size = NA
) +
# Label for end line
geom_label(
  x = 16.5,
  y = 42,
  label = "End of intervention",
  size = 3.5,
  fill = "white",
  label.size = NA
) +
scale_fill_distiller(palette =
  "YlGnBu") +
theme_minimal() +
scale_x_continuous(expand =
  c(0, 0)) +
coord_cartesian(clip = "off") +
theme(legend.position = "top") +
ylab("Participant ID") +
guides(fill = guide_colorbar
  (title = "Task performance")) +
theme(panel.grid = element_blank())
```

The resulting plot (see Fig. 6) visualizes each person's performance on the task across the 50 weeks, expressed by the color of the tiles.

Research Question 5: Does the intervention elicit differences in performance between the groups?

Another potential research question relates to observed group differences at specific time points of the intervention to examine the efficacy of the intervention. This aspect of the data is nicely visualized with a box plot. In our case, we are interested in group differences in Week 1 (start of the baseline), Week 5 (start of the intervention), Week 16 (end of the intervention), and Week 20 (end of the postintervention phase). We are also adding individual data points. Let us construct the box plot:

```
# Static box plot       (code snippet 15)

# For this plot, we are only using
  data from the critical time points
# and the boxplots need the variable
  Week to be a factor
ggplot(intervention_data %>% filter
  (Week %in% c(1, 4, 16, 20)),
     aes(y = Task, x = as.factor
       (Week), color = Group)) +
  # Points indicating individual
    performance
  geom_point(alpha = .4) +
  # Boxes with gray points for
    outliers
```

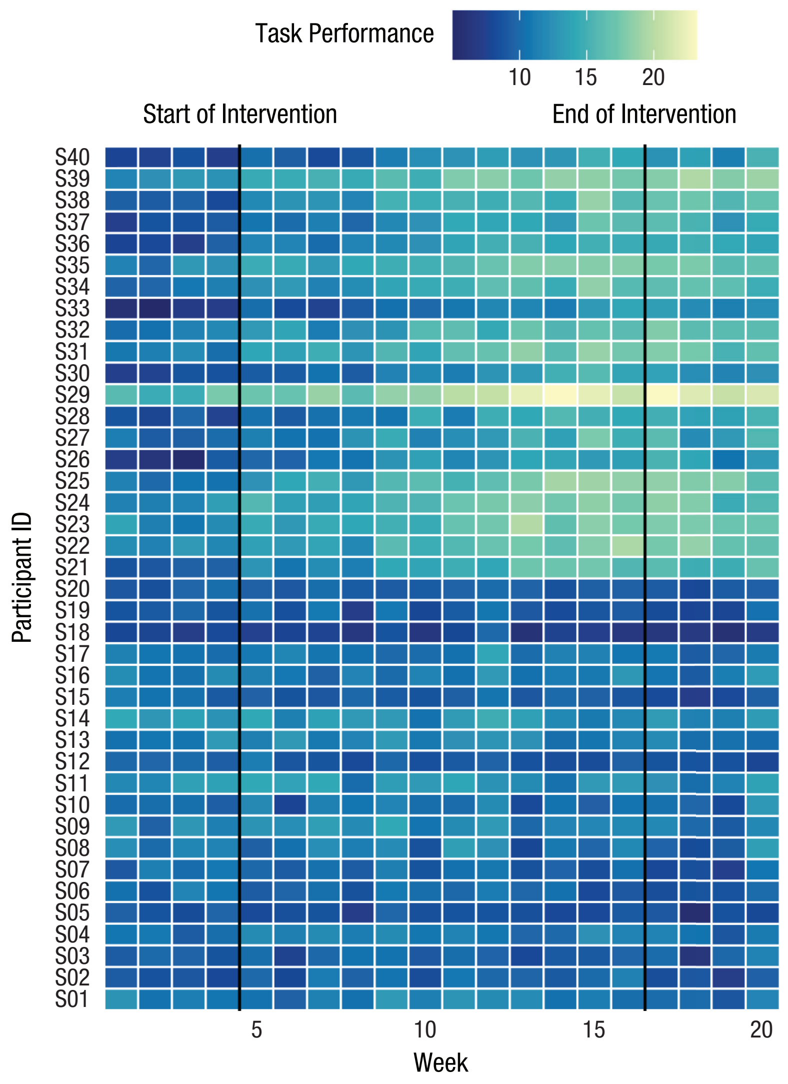


Fig. 6. Heat map displaying individual performance over time.

```
geom_boxplot(
  aes(fill = Group),
  alpha = .4,
  width = .5,
  position = position_dodge(.7),
  outlier.color = "gray"
) +
xlab("Week") +
ylab("Task performance") +
custom_theme
```

The resulting plot (see Fig. 7) suggests no differences at the first two time points, after which task performance appears to be higher for the intervention group than for the control group.

Research Question 6: Can people's performance on the task be predicted from their alertness level?

Our last question is about the relationship between task performance and alertness level. An important aspect to visualize here is whether this relationship changes over the course of the intervention to check whether the intervention changes the relationship between the variables (e.g., whether task performance becomes less correlated with alertness level).

To that end, we construct a scatter plot correlating task performance with alertness level using different colors to indicate different group and weeks:

```
# Static scatter plot      (code
  snippet 16)

ggplot(intervention_data, aes(x =
  Alertness, y = Task, group =
  Group)) +
  # Individual points
```

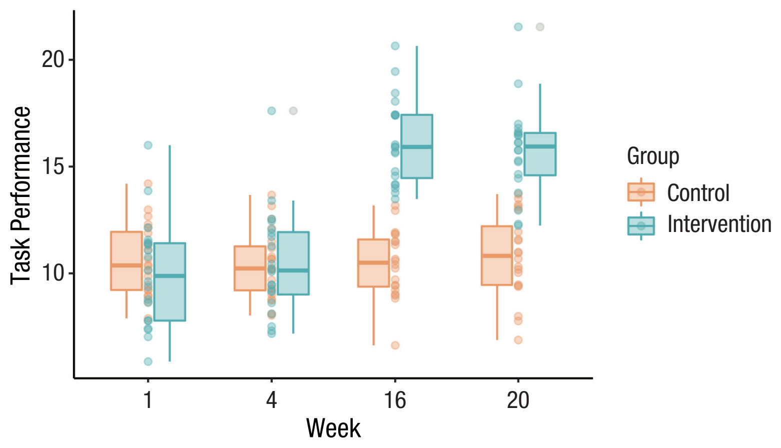


Fig. 7. Static box plot displaying potential group differences at specific time points.

```
    geom_point(aes(fill = Week, shape =
      Group),
                size = 2,
                alpha = .4) +
    # Group trendlines
    geom_smooth(aes(color = Group),
      method = "lm", fullrange = TRUE) +
    ylab("Task Performance") +
    theme_classic() +
    scale_fill_distiller(palette =
    "YlGnBu") +
    scale_color_manual(values =
      c("#eeaa7b", "#66b9bf")) +
    scale_shape_manual(values = c(22, 23))
```

Positive correlations across all weeks for both the intervention and the control groups can easily be identified (see Fig. 8), but how these correlations change across weeks is less easily discernible.

Although all of these plots convey useful information, adding dynamic content can extend functionality by either making the plots more suitable for data exploration or making the data more easily digestible for an audience during presentations. We illustrate these two aspects of dynamic plotting in the next sections.

Make It Pop: Interactive Plots

In this section, we demonstrate how to add interactive features to some of the plots above. Interactive plots are especially useful during the data-exploration phase, for example, to identify outliers by highlighting data of particular participants or to get a better overview of the data by visualizing descriptive statistics on the plots.

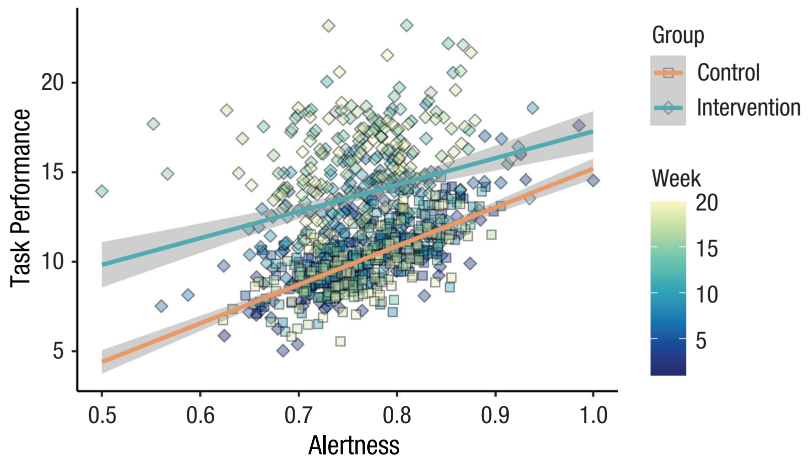


Fig. 8. Scatter plot displaying the correlation between task and underlying construct for the two groups across all weeks.

Example 1: future-imagination data set

Let us start with the violin plot we constructed for our first research question (see Fig. 1; code snippet 8). This plot is useful to inspect differences between conditions, but it is not as easy to identify data of specific individuals. We could assign different colors or shapes to the individual data points, but especially with large groups, this gets messy very quickly. A nice alternative is to make this plot interactive. Using the package *ggiraph*, we can highlight the data of single individuals by hovering over data points.

As mentioned earlier, *ggiraph* provides adapted geoms that will create the interactivity. To build an interactive version of the violin plot, we build the plot again, replacing *ggplot2*'s `geom_point()` with ggiraph's interactive alternative `geom_point_interactive()`:

```
# Interactive violin plot      (code
  snippet 17)

p_violin_interactive <-
  ggplot(imagination_data %>% filter
    (Time_point == 1),
       aes(x = fct_rev(Condition),
         y = RT)) +
  # Violins
  geom_violin(trim = FALSE, alpha =
    0.6, aes(fill = Condition)) +
  # Boxes
  geom_boxplot(width = 0.1) +
  # Individual data points
  geom_point_interactive(aes(tooltip
    = ID, data_id = ID), alpha = .4) +
  xlab("Condition") +
  ylab("Response time (ms)") +
  custom_theme +
  theme(legend.position = "none")
```

The code is identical to the previous version, apart from the name of the geom and the aesthetics that specify details of the interactivity. Inside `geom_point_interactive()`, we set `tooltip` and `data_id` to the subject identifier `ID`. This will display the participant's ID when one hovers over a data point and highlight all of this individual's data.

Handing this object to the `girafe()` function will create and display the interactive plot:

```
# Display interactive violin plot
  (code snippet 18)

girafe(ggobj = p_violin_interactive,
     options = list(
       opts_hover_inv(css = "opacity:
         0.2"),
       opts_hover(css = "stroke-
         width:1")
   ))
```

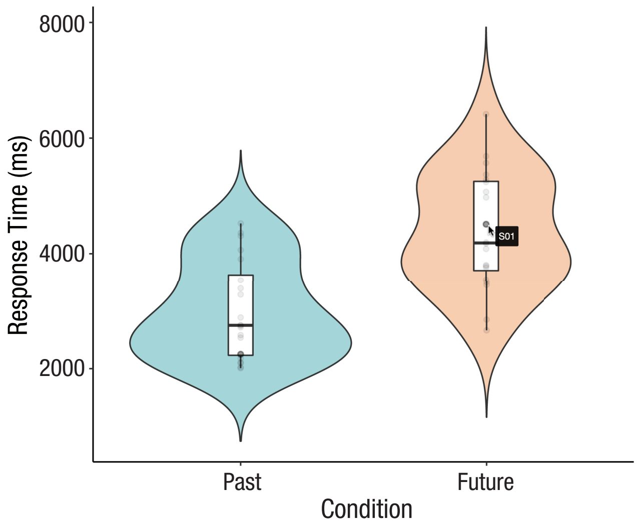


Fig. 9. Screenshot of interactive violin plot. Plot created with the *ggiraph* package. The interactive version is available at osf.io/tj2xr.

In addition to passing the name of the saved plot, the `girafe()` function also lets us specify further options. We added some hover options to make the nonselected data points transparent (`opts_hover_inv(css = "opacity:0.2")`) and to slightly increase the outline of the points (`opts_hover(css = "stroke-width:1")`) when highlighting individuals.

The result is a plot with which we can interact. Using this interactive version of the plot, we can inspect individuals' data points along with the participant label (see Fig. 9), which might be more cumbersome to find out using the static version of the plot or inspecting the data themselves.

Using the same strategy, we can also make the second violin plot (see Fig. 2; code snippet 9) interactive. This time, we use `geom_line_interactive()` in addition to *ggiraph*'s interactive version of `geom_point()`.

```
# Interactive violin plot with lines
  (code snippet 19)

p_violin_with_lines_interactive <-
  ggplot(imagination_data %>%
    filter(Condition == "Future"),
     aes(x = Time_point, y = RT)) +
  # Violins
  geom_violin(trim = FALSE, alpha =
    0.6, fill = "#94618e") +
  # Boxes
  geom_boxplot(width = 0.1) +
  # Individual data points
  geom_point_interactive(
    aes(tooltip = ID, data_id = ID),
    alpha = .4) +
  # Individual lines
  geom_line_interactive(
  aes(group = ID, tooltip = ID, data_
    id = ID),
  alpha = .2,
  linetype = "dashed") +
  xlab("Time point") +
  ylab("Response time (ms)") +
  custom_theme
```

As before, `girafe()` will create and display the interactive plot:

```
# Display interactive violin plot
  with lines   (code snippet 20)

girafe(ggobj = p_violin_with_lines_
  interactive,
    options = list(
      opts_hover_inv(css = "opacity:
        0.2"),
      opts_hover(css = "stroke-width:1")
    ))
```

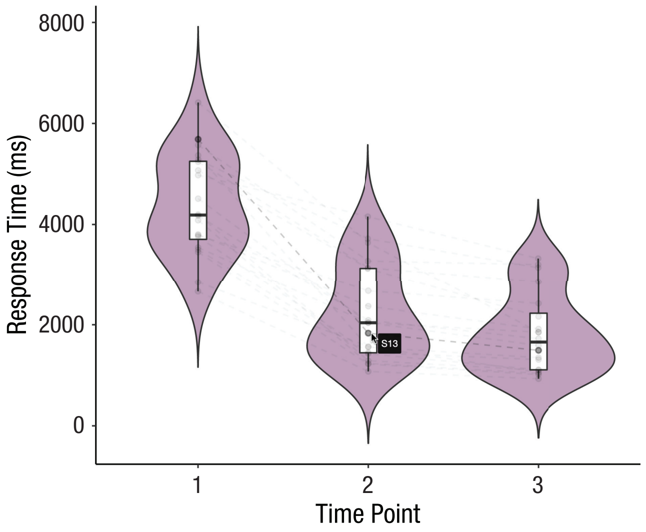


Fig. 10. Screenshot of interactive violin plot with lines. Plot created with the *ggiraph* package. The interactive version is available at osf.io/tj2xr.

In this interactive version of the plot (see Fig. 10), the lines connecting individuals' data points are highlighted in addition to the points, a feature that might be especially useful with bigger data sets.

Finally, *ggiraph* and its *geom_point_interactive()* can also be used to easily make the scatter plot (see Fig. 3; code snippet 11) interactive:

```
# Interactive scatter plot      (code
  snippet 21)

p_scatter_interactive <-
  ggplot(imagination_data_wide,
      aes(x = RT_Past, y = RT_Future,
        color = Time_point)) +
  # Individual points
  geom_point_interactive(
    aes(size = Detail_Mean, tooltip =
      ID, data_id = ID),
    alpha = .4) +
# Trendlines
  geom_smooth(method = "lm") +
# Distributions
geom_rug(alpha = .4) +
  xlab("Response time (ms) past
    events") +
  ylab("Response time (ms) future
    events") +
  custom_theme +
  guides(color = guide_legend(title =
    "Time point"),
  size = guide_legend(title =
    "Detail rating"))

girafe(ggobj = p_scatter_interactive,
    options = list(
      opts_hover_inv(css = "opacity:
        0.2"),
      opts_hover(css = "stroke-width:1")
    ))
```

This interactive version of the scatter plot (Fig. 11) allows us to highlight individuals' data points across the three time points.

Example 2: intervention data set

Let us look at the plots for the second data set. The line graph we constructed for Research Question 4 (see Fig. 4; code snippet 12) is useful to inspect group trends, but making it interactive makes it very easy to spot individual trajectories or atypical patterns in the data. To

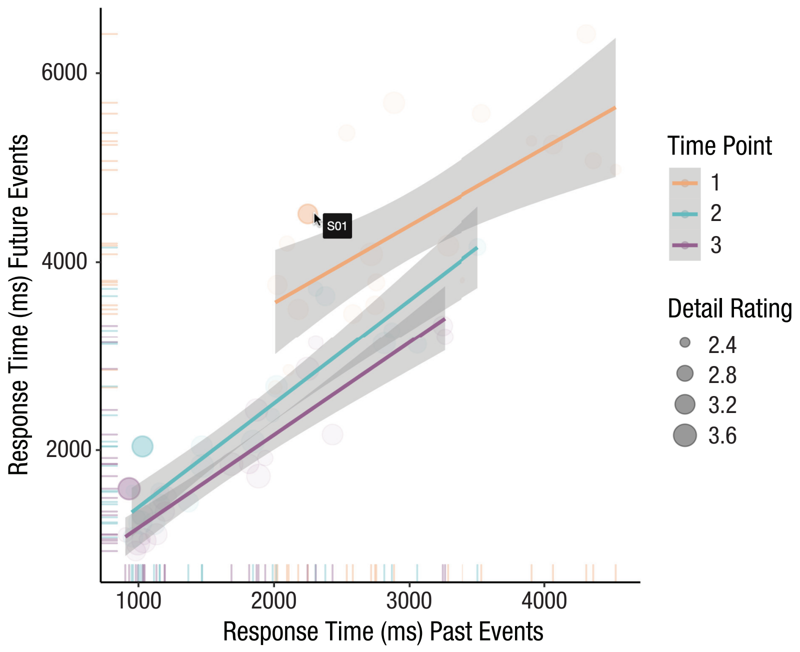


Fig. 11. Screenshot of interactive scatter plot. Plot created with the *ggiraph* package. The interactive version is available at osf.io/tj2xr.

build an interactive version of the line graph, we replace *ggplot2*'s `geom_line()` with *ggiraph*'s interactive alternative `geom_line_interactive()`:

```
# Interactive line graph      (code
  snippet 22)

p_line_interactive <-
  ggplot(intervention_data, aes(x =
    Week, y = Task, color = Group)) +

  # Gray rectangle to highlight
  intervention phase
  annotate(
    "rect",
    xmin = 4.5,
    xmax = 16.5,
    ymin = 0,
    ymax = Inf,
    alpha = 0.1,
    fill = "gray45"
  ) +
  # Label for rectangle
  annotate(
    geom = "text",
    x = 10.5,
    y = max(intervention_data$Task) + 1,
    label = "Intervention",
    size = 3.5,
    color = "gray35"
  ) +
  # Individual lines
  geom_line_interactive(aes(
    group = ID,
    tooltip = ID,
    data_id = ID
  ),
  size = .5,
  alpha = .15) +
  # Group average lines
  stat_summary(geom = "line",
               fun = "mean") +
  # Group average points
  stat_summary(geom = "point",
               fun = "mean") +
  ylab("Task performance") +
  custom_theme +
  theme(legend.position = "top")
```

As with the other data set, handing this object to the `girafe()` function will create and display the interactive plot:

```
# Display interactive line graph
  (code snippet 23)

girafe(ggobj = p_line_interactive,
     options = list(
        opts_hover_inv(css = "opacity:
          0.2"),
        opts_hover(css = "stroke-
          width:1.5")
     ))
```

Using this interactive version of the plot (see Fig. 12), we can inspect individuals' trajectories across time easily, get the participant labels, and identify potential outliers.

In a similar way, the heat map displaying individual performance (see Fig. 6; code snippet 14) can also be made interactive. As before, we use *ggiraph* and change only the geom layer:

```
# Interactive heatmap      (code
snippet 24)

p_tile_interactive <-
  ggplot(intervention_data, aes(
    x = Week,
    y = ID,
    group = Group,
    fill = Task
  )) +
  # Tiles
  geom_tile_interactive(color = "white",
          size = 0.35,
          aes(tooltip = Task, data_
            id = ID)) +
  # Line indicating the start of the
    intervention
  geom_vline(xintercept = c(4.5,
    16.5), col = "black") +
  # Label for start line
  geom_label(
    x = 4.5,
    y = 42,
    label = "Start of intervention",
    size = 3.5,
    fill = "white",
    label.size = NA
  ) +
  # Label for end line
  geom_label(
    x = 16.5,
    y = 42,
    label = "End of intervention",
    size = 3.5,
    fill = "white",
    label.size = NA
  ) +
```

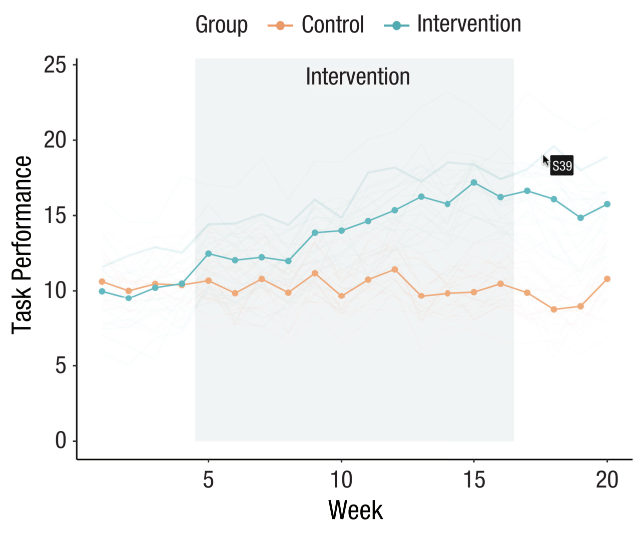


Fig. 12. Screenshot of interactive line graph. Plot created with the *ggiraph* package. The interactive version is available at osf.io/tj2xr.

```
  scale_fill_distiller(palette =
    "YlGnBu") +
  theme_minimal() +
  scale_x_continuous(expand = c(0,
    0)) +
  coord_cartesian(clip = "off") +
  theme(legend.position = "top") +
  ylab("Participant ID") +
  guides(fill = guide_colorbar(title
    = "Task performance")) +
  theme(panel.grid = element_blank())
```

This time, `geom_tile()` is replaced by `geom_tile_interactive()`. Inside this function, we specify that we want to display the task-performance values (`tooltip = Task`). The line `data_id = ID` ensures that all data of one individual is highlighted; changing `data_id` to `Week` instead will highlight a particular week. Let us use the `girafe()` function to create the interactive plot:

```
# Display interactive heatmap
  (code snippet 25)

girafe(ggobj = p_tile_interactive,
     options = list(
        opts_hover_inv(css = "opacity:
          0.4"),
        opts_hover(css = "stroke-
          width:.5")
     ))
```

Apart from slightly adapting the hover options, the code is the same as for the line graph (see Fig. 13).

In the data-exploration phase, it might also be useful to add summary statistics to a plot so that specific values can be directly inspected without the need to generate them separately, for example, in the form of a table. This can be rendered easily with the *plotly* package. *Plotly*'s syntax is slightly different to the one used by *ggplot2*, but the plot is also created in layers. Let us use *plotly* to create an interactive version of the box plot from earlier (see Fig. 7; code snippet 15):

```
# Interactive box plot      (code
  snippet 26)

# For this plot, we are only using
  data from the critical time points
# and the boxplots need the variable
  Week to be a factor
plot_ly(
  intervention_data %>% filter(Week
    %in% c(1, 4, 16, 20)),
  y = ~Task,
  x = ~as.factor(Week),
  color = ~Group,
  # Boxes
  type = "box",
  colors = c("#eeaa7b", "#66b9bf")
) %>%
  # Grouping of boxes
  layout(boxmode = "group") %>%
  # Points indicating individual
    performance
  add_trace(type = "scatter") %>%
  layout(xaxis = list(title =
    list(text = "Week")),
      yaxis = list(title = list(text =
        "Task performance")))
```

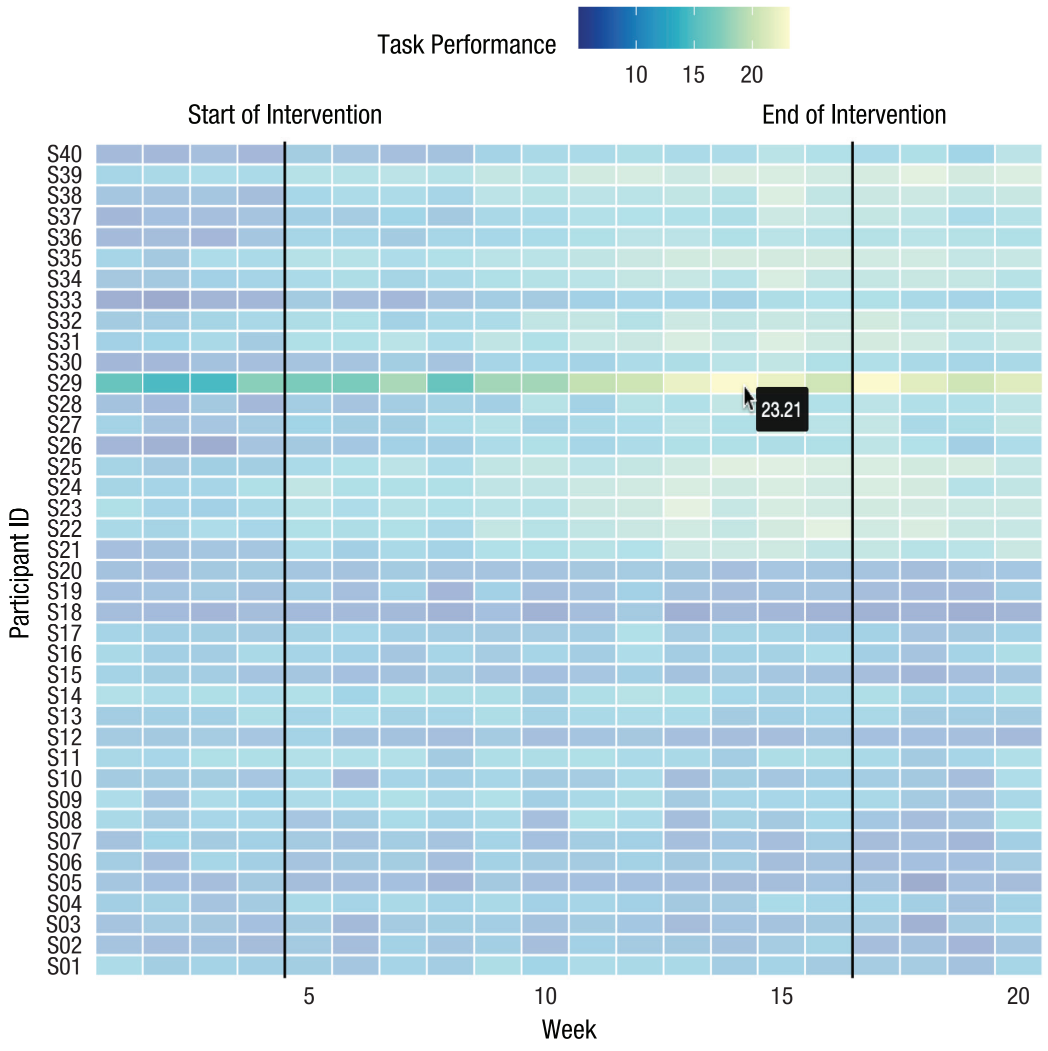


Fig. 13. Screenshot of interactive heat map. Plots created with the *ggiraph* package. The interactive version is available at osf.io/tj2xr.

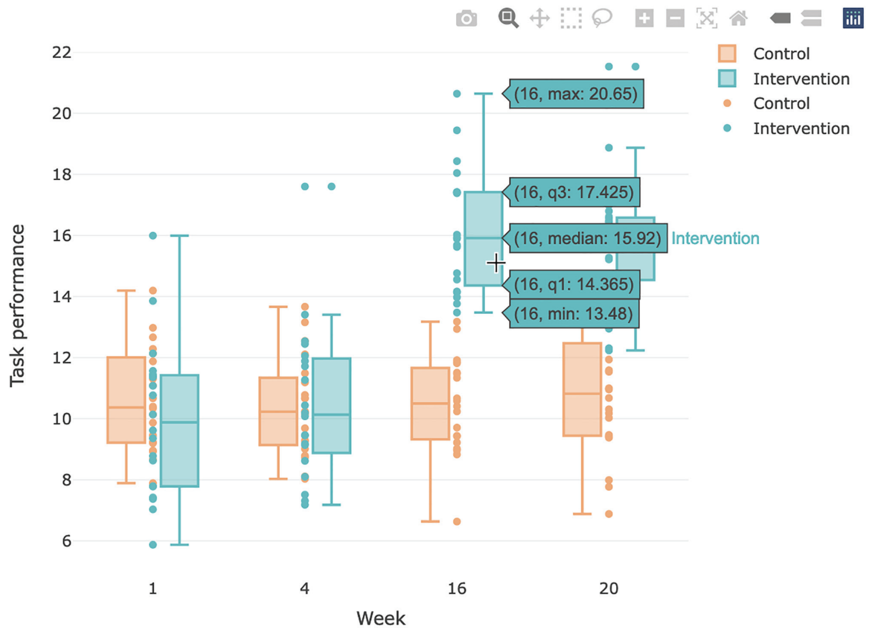


Fig. 14. Screenshot of interactive box plot. Plot created with the *plotly* package. The interactive version can is available at osf.io/tj2xr.

We use the `plot_ly()` instead of the `ggplotly()` function here because although the latter works well with most simple types of plots, it does not handle the grouping properly in this case. As with *ggplot2*, we first specify which data set and variables we want to use. We also specify that we want to create a box plot (`type = "box"`), the line `layout(boxmode = "group")` handles the grouping, and the last line adds points to the plot (`add_trace(type = "scatter")`). For this plot, we additionally specify the colors because our custom theme is *ggplot2*-specific. The code chunk produces the plot displayed in Figure 14. Hovering over different boxes in the plot displays useful summary statistics. The values of single data points can also be displayed. Some additional interesting features provided by *plotly* are that parts of the plot can be hidden by clicking on legend entries and zoomed by clicking and dragging parts of the plot with the mouse. The toolbar on the top right can be used for further functionalities, such as saving the plot.

Make It Smooth: Building Animations

Animations, which allow visualizing the evolution of data across a variable such as time, are particularly useful for the presentation of findings, for example, during talks. Several packages exist now that make turning plots into animations fairly straightforward.

Example 1: future-imagination data set

We start with the violin plot from earlier (see Fig. 2; code snippet 9) and adapt it so that the three violin plots are revealed sequentially. Using the *gganimate* package, the *ggplot2* code can be reused, with minimal additional code that specifies the desired animation type. In this case, we add *gganimate*'s layer `transition_time()` to the *ggplot2* code, which declares that data should be revealed along a continuous variable (in this case, `Time_point`). If no animation gets created and you get the error message `file_renderer failed to copy frames to the destination directory`, check for writing permissions or try running Rstudio as administrator.

There are two parts to creating animated plots with *gganimate*. First, we specify the plot and the animation type we want to use:

```
# Animated violin plot with lines
  (code snippet 27)

p_violin_animated <-
  ggplot(imagination_data %>%
    filter(Condition == "Future"),
     aes(x = as.numeric(Time_point),
       y = RT)) +
  # Violins
```

```
geom_violin(trim = FALSE, alpha =
  0.6, fill = "#94618e") +
# Boxes
geom_boxplot(width = 0.1) +
# Points indicating individual
  performance
geom_point(alpha = .4) +
xlab("Time point") +
ylab("Response time (ms)") +
custom_theme +
transition_time(as.numeric(Time_
  point)) +
shadow_mark() +
ease_aes('cubic-in-out')
```

Using `Time_point` inside *gganimate's* `transition_time()` specifies that the plot should be revealed along the three time points of the experiment during which the same event was imagined. The layer `shadow_mark()` ensures that once the plots are revealed, they persist instead of disappearing, and the `ease_aes()` function defines the easing of the animation. To find all possible easing options, see Pedersen and Robinson (n.d.).

The animated version of the plot can be displayed in Rstudio or in an html file by either printing the variable name or using *gganimate*'s `animate()` function:

```
# Display animated violin plot with
  lines   (code snippet 28)

animate(p_violin_animated, end_pause
  = 40)
```

Using this function, we can specify additional arguments. Here, we added a pause of 40 frames at the end of the animation, before it starts from the beginning. The renderer can also be changed if a different file format is preferred. The renderer `av_renderer()`, for example, will return a video as output, which can be useful to pause the animation during presentations. The resulting animation is visualized in Figure 15.

Finally, the animation can be saved to file using:

```
# Save animated violin plot with
  lines    (code snippet 29)

anim_save("violin_anim.gif",
  animation = last_animation())
```

The function `last_animation()` is used analogously to *ggplot2*'s `last_plot()` to retrieve the most recently created animation. Alternatively, the animation can be saved into a variable, and the variable name can then be used in `anim_save()`.

In a similar way, we can also use *gganimate* to animate the scatter plot from the first example (see Fig. 3; code snippet 11) to cycle through the time points. This time, we use *gganimate*'s `transition_states()`, which allows animating over a categorical variable:

```
# Animated scatter plot      (code
  snippet 30)

p_scatter_animated <-
  ggplot(imagination_data_wide,
    aes(x = RT_Past, y = RT_Future,
      color = Time_point)) +
  # Individual points
  geom_point(alpha = .4, aes(group =
    ID, size = Detail_Mean)) +
  # Trendlines
  geom_smooth(method = "lm", se =
    FALSE, aes(group = Time_point)) +
  # Distributions
  geom_rug(alpha = .4, aes(group =
    ID)) +
  xlab("Past construction time
    (ms)") +
  ylab("Future construction time
    (ms)") +
  custom_theme +
  guides(color = guide_legend(title =
    "Time point"),
    size = guide_legend(title =
      "Detail rating")) +
  transition_states(Time_point,
    wrap = FALSE) +
  shadow_mark() +
  ease_aes('cubic-in-out')
```

As before, let us view the animation using the `animate()` function:

```
# Display animated scatter plot
  (code snippet 31)

animate(p_scatter_animated, end_
  pause = 40)
```

The animation is visualized in Figure 16.

Example 2: intervention data set

For the intervention data set, we illustrate the same process using *gganimate* for the line graph, the ridgeline plot, and the scatter plot. We start with the line graph (see Fig. 4; code snippet 12) and adapt it so that the data are gradually revealed across the weeks of the

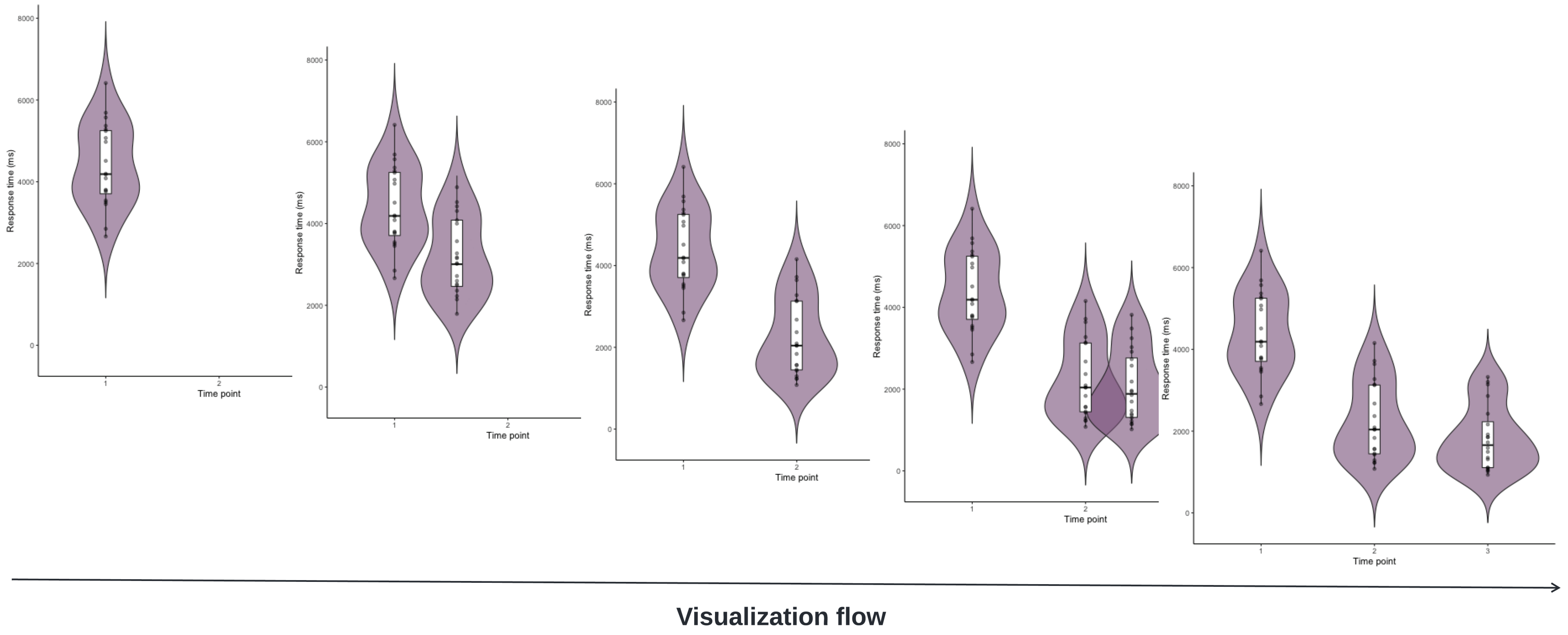


Fig. 15. Screenshots of selected frames of the animated violin plot. Plot created with the *gganimate* package. The corresponding animated version of the plot is available at osf.io/tj2xr.

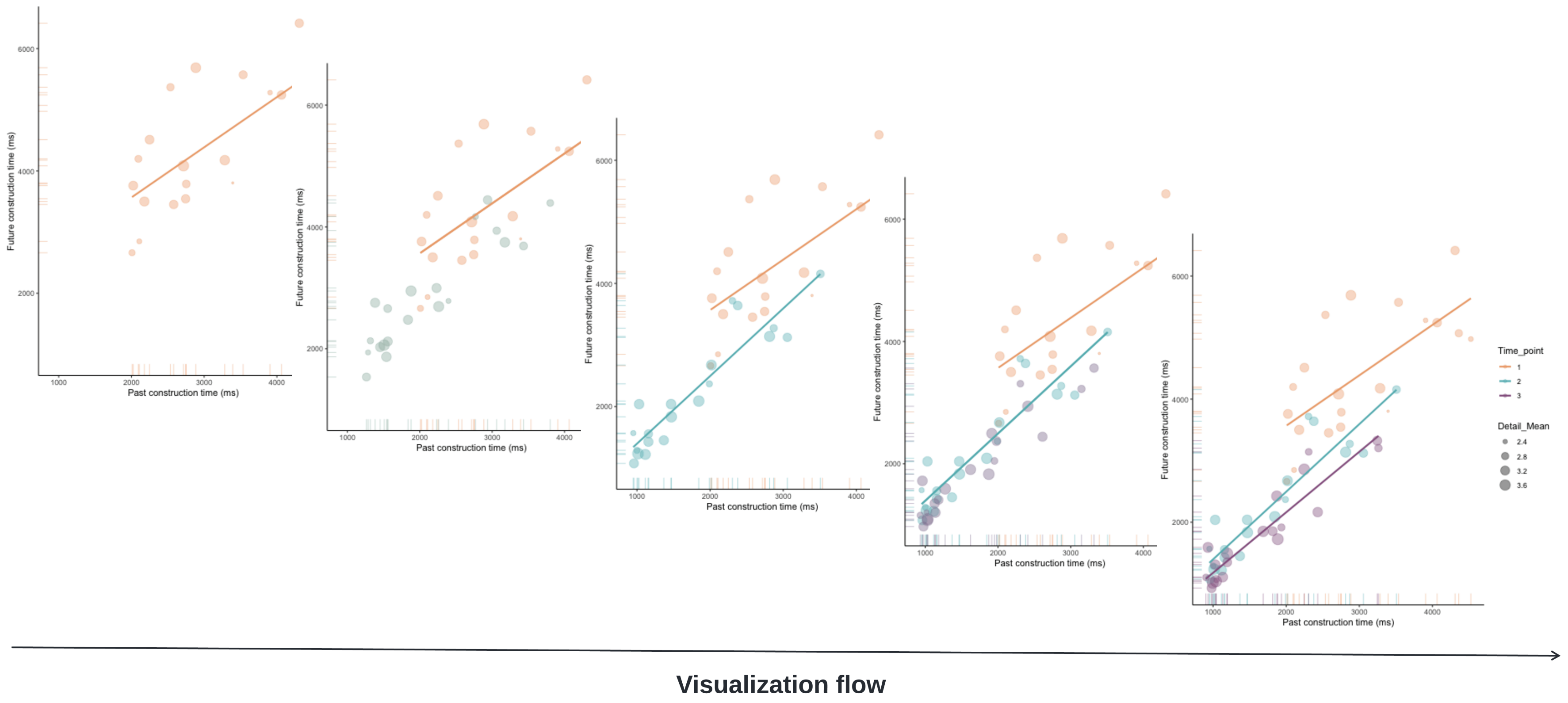


Fig. 16. Screenshots of selected frames of the animated scatter plot. Plot created with the *gganimate* package. The corresponding animated version of the plot is available at osf.io/tj2xr.

intervention. This time, we use *gganimate*'s layer `transition_reveal()` to the *ggplot2* code, which is typically used to gradually reveal time-series data:

```
# Animated line graph      (code
  snippet 32)

p_line_animated <-
  ggplot(intervention_data, aes(x =
    Week, y = Task, color = Group))+
  # Gray rectangle to highlight
    intervention phase
  annotate(
    "rect",
    xmin = 4.5,
    xmax = 16.5,
    ymin = 0,
    ymax = Inf,
    alpha = 0.1,
    fill = "gray45"
  ) +
  # Label for rectangle
  annotate(
  geom = "text",
    x = 10.5,
    y = max(intervention_data$Task) + 1,
    label = "Intervention",
    size = 3.5,
    color = "gray35"
  ) +
  # Individual lines
  geom_line(aes(group = ID), size =
    .5, alpha = .15) +
  # Group average lines
  stat_summary(geom = "line", fun =
    "mean") +
  # Group average points
  stat_summary(geom = "point", fun =
    "mean") +
  ylab("Task performance") +
  custom_theme +
  theme(legend.position = "top") +
  transition_reveal(Week)
```

Using `Week` inside `transition_reveal()` specifies that the plot should be revealed along the weeks of the intervention. Let us display the resulting animation:

```
# Display animated line graph
  (code snippet 33)

animate(p_line_animated, fps = 6,
  end_pause = 40)
```

In addition to the pause of 40 frames at the end of the animation, we specified that six frames per second should be displayed (10 is the default). See the resulting animation in Figure 17.

Instead of gradually revealing parts of the plot over time, we can animate the ridgeline plot (see Fig. 5; code snippet 13) so that the distributions dynamically shift across the weeks of the intervention. This animation style is similar to the animation of the scatter plot of the future-imagination data set used earlier (code snippet 30), so we use *gganimate*'s `transition_states()` layer again:

```
# Animated ridgeline plot      (code
  snippet 34)

p_ridge_anim <-
  ggplot(intervention_data, aes(
    x = Task,
    y = 0,
    color = Group,
    fill = Group
  )) +
  geom_density_ridges(alpha = .4) +
  xlab("Task Performance") +
  custom_theme +
  theme(
    axis.title.y = element_blank(),
    axis.text.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.line.y = element_blank()
  ) +
  transition_states(Week, transition_
    length = 3, state_length = 0) +
  labs(title = "Week {closest_state}")
```

In addition to the `transition_states()` layer, we also added a title to the plot that indicates which intervention week is being displayed (`labs(title = "Week {closest_state}")`), and we used the `theme()` layer to indicate that we do not want to display a *y*-axis. Note that in the `aes` layer, we also set `y` to 0.

After building the plot, let us use `animate()` to display it and control animation-related options:

```
# Display animated ridgeline plot
  (code snippet 35)

animate(p_ridge_anim, fps = 3,
  width = 500, height = 150, end_
  pause = 40)
```

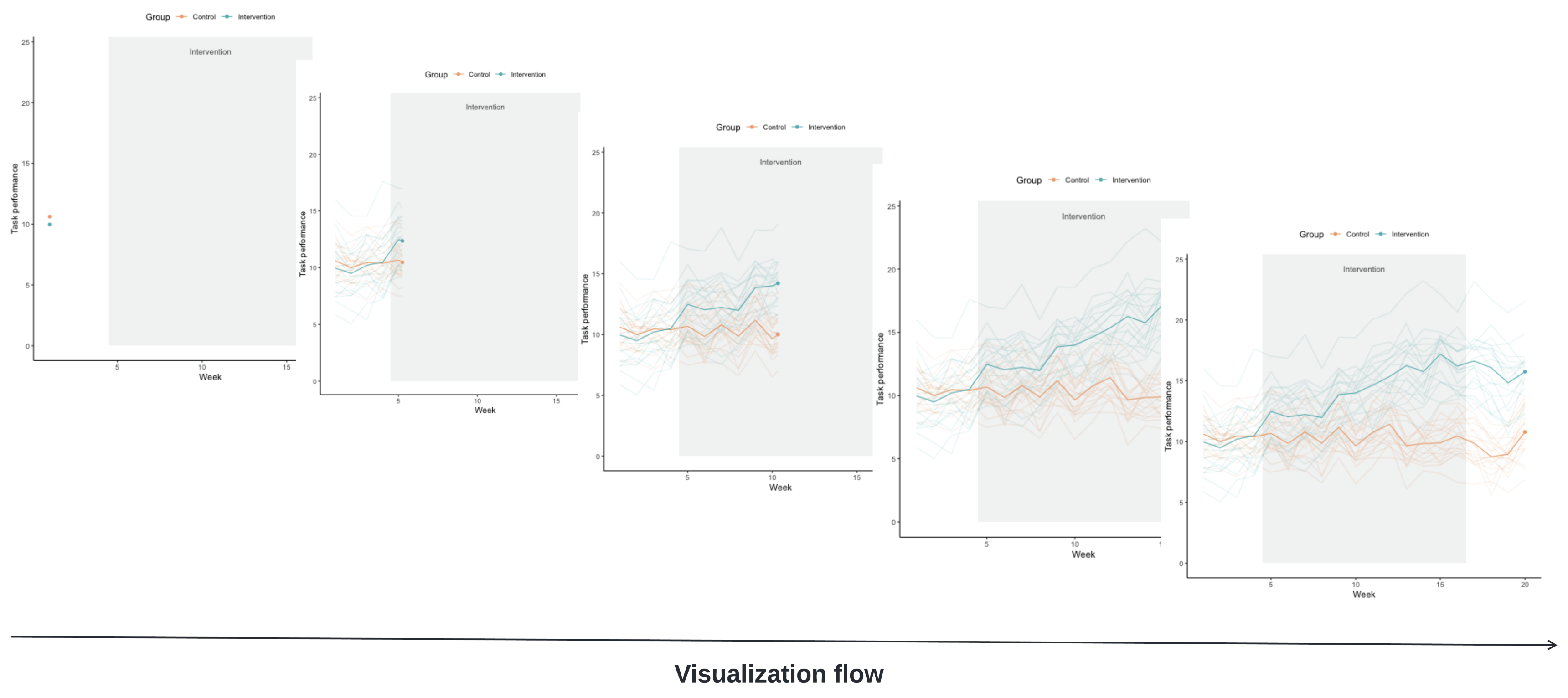


Fig. 17. Screenshots of selected frames of the animated line graph. Plot created with the *gganimate* package. The corresponding animated version of the plot is available at osf.io/tj2xr.

This time, we also specified the optional arguments `width` and `height` to illustrate additional customization options (see Fig. 18).

Beyond making the presentation of plots more visually appealing or more easily digestible during presentations, animations also allow us to examine and present new aspects of our data, such as the evolution of the intervention-data scatter plot (see Fig. 8; code snippet 16) over time. Similar to the previous example, we can cycle through the weeks, constructing a separate correlation plot for each week. Given that we already animated a correlation plot and that we have used *gganimate* in several different examples now, we use *plotly* this time to showcase another way to create animations. With *plotly*, a version of the static plot first needs to be recreated, with the aesthetics argument `frame` added to `geom_point()`, which specifies which dimension to animate over (in our case, `Week`):

```
# Animated scatter plot      (code
  snippet 36)

p_scatter2_animated <-
  ggplot(intervention_data, aes(
    x = Alertness,
    y = Task,
    group = Group,
    color = Group
  )) +
  # Individual points
  geom_point(aes(frame = Week), alpha =
    .6) +
  # Group trendlines
  geom_smooth(
    aes(frame = Week, ids = ID),
    method = "lm",
    se = FALSE,
    fullrange = T
  ) +
  custom_theme
```

We can then use `ggplotly()` to animate and display the plot:

```
# Animate and display the scatter
  plot    (code snippet 37)

ggplotly(p_scatter2_animated)
```

Using the animated version of the scatter plot (see Fig. 19), it becomes more easily apparent that the correlation between task performance and alertness level decreased across the intervention for the intervention group but not to the same extent for the control group.

Make It Shine: Blending Interactive and Animated Features

Once a project is complete, data and materials are often shared alongside the article that describes the project and presents its findings. Packaging these up into a dashboard or web app for a project is a great way not only to share additional material and let others recreate plots and statistics from the article but also to allow for additional exploration and manipulation of the data, such as examining the effect of outliers or particular modeling choices on the results. The R package *shiny* makes building interactive dashboards and web apps straightforward, without the need for web development skills and deep knowledge of web technologies such as HTML or CSS.

Extensive and detailed shiny tutorials are provided elsewhere (see e.g., *Shiny Learning Resources*, n.d.; Wickham, 2021). Briefly, Shiny apps are written in a single R script called `app.R`, which consists of three main components: a `ui` (user interface) object that contains information about the layout of the app, a `server` function that contains all code needed to build and update the objects in the app, and a call to the `shinyApp` function to build the app. As mentioned earlier, *shiny* uses a programming style called reactive programming. Reactive programming lets you control which parts of your app update (i.e., which parts of your code are rerun) and when using the inputs provided by the user. For our purposes, the main reactive components are reactive sources (defined as `input` in the `ui` object) and reactive endpoints (defined as `output` in the `server` function). A reactive endpoint can, for example, be a plot that is rendered by *shiny*'s `renderPlot()` function that is then displayed in the app using `plotOutput()`. The code for the plot will be rerun every time the input changes. When combining *shiny* with any of the dynamic plotting packages used in this tutorial, these endpoint functions have to be adapted to their package-specific alternative. We provide an example below.

To demonstrate some of Shiny's features, we built a Shiny app that includes dynamic plots for each of the questions we addressed for the intervention-data-set example (Example 2), additionally allowing for one or several individuals to be excluded to check robustness (see Fig. 20). For simplicity purposes, we provide instructions for a simplified version of this app below, which displays just the interactive line graph created with *ggiraph* but still allows for the exclusion of individuals from the data set. Whenever we reuse code from earlier parts of the tutorial, we use a placeholder, indicating which code snippets should be inserted (e.g., `# <Set up custom theme – code snippet 3>`). The full

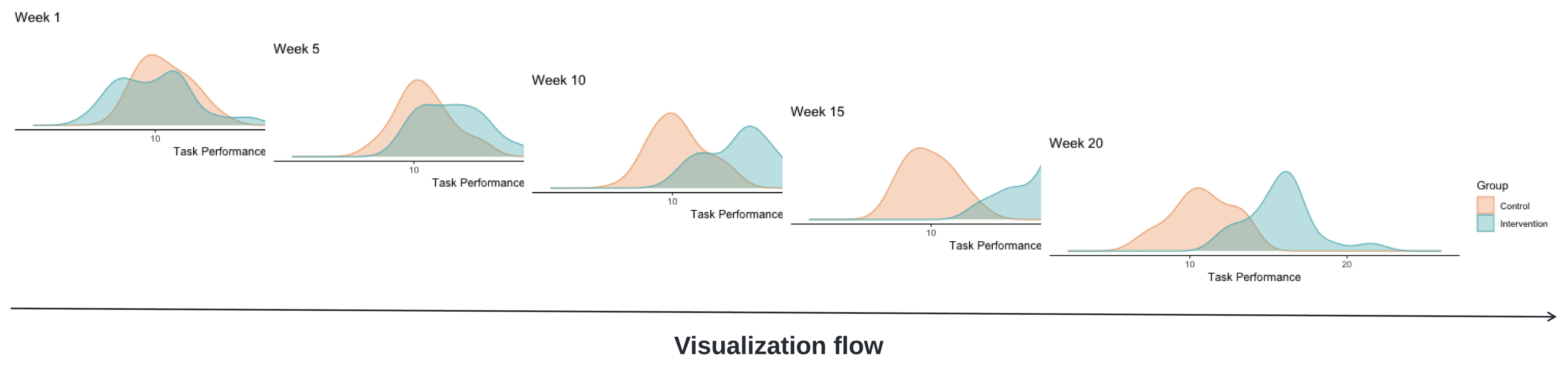


Fig. 18. Screenshots of selected frames of the animated ridgeplot. Plot created with the *gganimate* package. The corresponding animated version of the plot is available at osf.io/tj2xr.

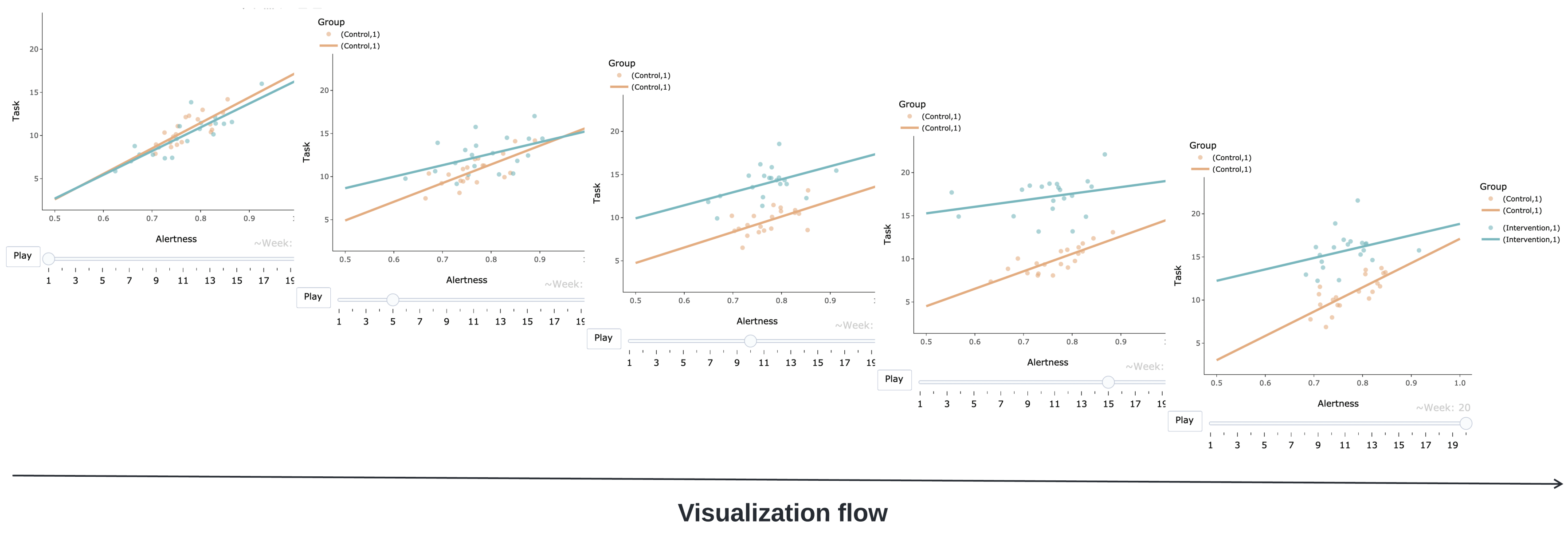
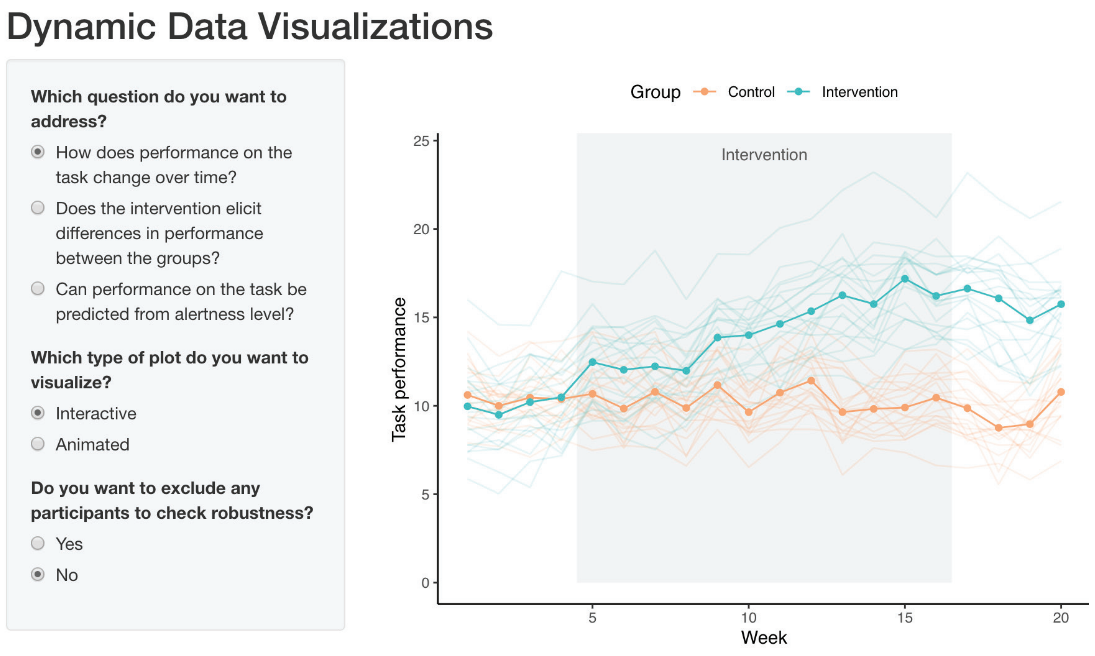


Fig. 19. Screenshots of selected frames of the animated scatter plot. Plot created with the *plotly* package. The corresponding animated version of the plot is available at osf.io/tj2xr.



Dynamic Data Visualizations

Which question do you want to address?

- [✓ How does performance on the task change over time?]
- Does the intervention elicit differences in performance between the groups?
- Can performance on the task be predicted from alertness level?

Which type of plot do you want to visualize?

- [✓ Interactive]
- Animated

Do you want to exclude any participants to check robustness?

- Yes
- [✓ No]

Fig. 20. Screenshot of Shiny app with dynamic plots.

Shiny app is available at kwiebels.shinyapps.io/Dynamic_Data_Visualizations, and the code for the simplified and the full app is available at osf.io/fwy8j.

To start, create a new R script called `app.R`. Inside this script, we first need to load and prepare packages and data:

```
# Shiny app preamble      (code
  snippet 38)

# Load packages
library(tidyverse)
library(shiny)
library(ggplot2)
library(ggiraph)

# <Set up custom theme – code snippet
  3>

# Load data
intervention_data <- read_
  csv("intervention_study.csv")
intervention_data$ID <- as.factor
  (intervention_data$ID)
```

Everything needed for the preamble has already been covered in the tutorial, apart from the conversion of `ID` into a factor, which is needed to allow the exclusion of individuals from the data set. Next, let us add the `ui` object to the script:

```
# Shiny app ui object       (code
snippet 39)

ui <- fluidPage(
  titlePanel("Dynamic Data
    Visualizations"),

  sidebarLayout(
    sidebarPanel(
      radioButtons(
    inputId = "exclude_choice",
    label = "Do you want to exclude
      any participants to check
      robustness?",
    choices = list("Yes", "No"),
    selected = "No"
    ),
    conditionalPanel(
      condition = "input.exclude_
        choice == 'Yes'",
    selectInput(
      inputId = "exclude",
      label = "Which participant(s) do
        you want to exclude?",
      choices = levels(intervention_
        data$ID),
        multiple = TRUE
        )
      )
    ),

    mainPanel(girafeOutput("plot_
      linegraph"))
    )
  )
```

In the ui object, we specify everything related to the visual appearance of the app, including the title of the app, the general layout, the options available to users in the sidebar panel, and what should be displayed in the main panel. Note that we had to replace *shiny*'s reactive endpoint function plotOutput() with *ggiraph*'s girafeOutput().

Next, we add the server function:

```
  # Shiny app server function      (code
    snippet 40)

  server <- function(input, output) {
    output$plot_linegraph
      <- renderGirafe({
    if (input$exclude_choice == "Yes") {
      seq <- input$exclude
      intervention_data <-
         intervention_data[!
           (intervention_data$ID
           %in% seq),]
    }

    # <Interactive line graph - code
      snippet 22>

    # <Display interactive line graph -
      code snippet 23>

  })
  }
```

Only two things are required in the server function for our app: removing data of excluded individuals (if the user selects this option) and the code for the plot. Note that, as in the previous code snippet, we had to replace the *shiny* function renderPlot() with *ggiraph*'s renderGirafe().

Finally, we need to add code to the end of the script to build the app:

```
  # shinyApp function call      (code
    snippet 41)

  shinyApp(ui = ui, server = server)
```

Running this script will display the app (see Fig. 21).

The reactive endpoint functions for the other packages we used throughout this tutorial have to be changed similarly to the *ggiraph* ones when used in Shiny apps. For interactive and animated graphs with *plotly*, these are changed to plotlyOutput() and renderPlotly(). There are different options to display animations created with *gganimate*, one of which requires the gif image to be saved into a temporary image with renderImage() that can then be displayed in the app using imageOutput(). Refer to the code for the full version of the Shiny app for further details.

Discussion

In this tutorial, we described how to turn static figures into interactive and animated content. We used the R statistical language throughout and based our examples on common types of plots and widely used packages. The first part of the tutorial focused on building interactive content, an aspect particularly important to early stages of a scientific project, such as data exploration. In the second part of the tutorial, we showed how to animate plots, a useful feature for data communication to scientific and general audiences. Finally, we integrated these two components into a single Shiny app that let us combine the flexibility of interactive graphs with the visual appeal and conciseness of animations. These richer modes of visualization can be ideal for sharing findings, with prespecified features that can help users explore key aspects of a study results. To conclude, we provide a number of practical recommendations to help researchers navigate use and implementation of dynamic content into a research project and close with a few remarks about prospective challenges and opportunities in the field of psychology.

Practical recommendations

Fancier is not always better; sometimes, traditional, static figures are the best way to convey information in a clear and efficient manner (Bétrancourt & Tversky, 2000; Lewalter, 2003). Because dynamic visualizations have a cost—inasmuch as they represent additional time, effort, and sometimes resources compared with more traditional visual displays—it may be difficult to gauge what content to turn into interactive or animated displays and when. Here, we provide five recommendations that we hope can help guide the reader through this process.

Recommendation 1: understand the specifics of your data

Given the variety of options now available to psychologists to present their research, understanding the type of data that is to be displayed is key. If the data structure is complex, multifaceted, or layered, interactivity can often be valuable because it provides a tool to explore different aspects sequentially or in combination with one another (Ward et al., 2011). Animations can be especially beneficial to represent phenomena that change over time or processes that are being iterated over (Robertson et al., 2008; Yu et al., 2010). In the case of low-dimensional data or if the added features lead to an unnecessary cognitive burden, the value of dynamic visualizations over that of static displays might be less evident (Steele & Iliinsky, 2010).

Recommendation 2: know your audience

Just like the data, the intended audience is also a crucial element in deciding to use dynamic displays (Kennedy, 2012). Peers might value interactivity to be able to verify assumptions, check alternative explanations, or explore complementary findings. Animations might be better suited to large, eclectic audiences who may not have the time or expertise necessary to invest in active exploration of the data. Students and trainees might benefit from either or both of these features, depending on their specific goals and needs.

Recommendation 3: adapt visualizations to the current needs of the project

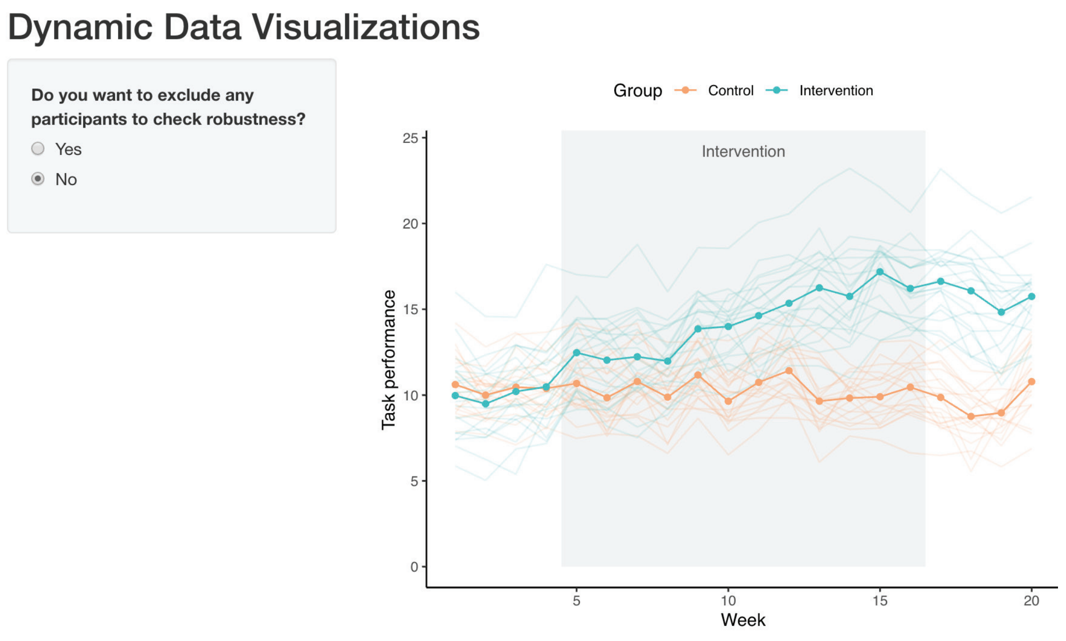
The requirements of a research project often differ across its life cycle. Features that are key to data exploration may not match those needed to discuss results with a team of collaborators or to present findings to a larger audience. Flexibility and creativity in the display of visual content can facilitate insights, help convey information in a more effective way, and make a presentation more memorable.

Recommendation 4: complement journal publication with online materials

In case journal platforms lack the capabilities to display dynamic graphs, it is relatively straightforward to complement the publication of an article with online materials, for example, in the form of a repository that can enable more sophisticated display. Tools such as Binder (mybinder.org) or Stencila (stenci.la) can help turn a static repository (e.g., from GitHub; github.com) into a collection of interactive notebooks or executable documents. We also provide an example of a repository including dynamic content with this article, hosted on OSF (osf.io). Along with alternatives such as Dryad (datadryad.org), FigShare (figshare.com), or Zenodo (zenodo.org), OSF allows creating a persistent digital object identifier for each submission, making repositories and their content easily citable.

Recommendation 5: consider packaging research findings into a Shiny app

In many cases, articles can benefit from alternate modes of presentation for the reported findings, which allow active exploration of the results. For this, we recommend


Dynamic Data Visualizations

Do you want to exclude any participants to check robustness?

Yes [✓ No]

Fig. 21. Screenshot of simplified version of Shiny app.

Shiny apps, which are extremely flexible and easy to implement in R. Exploring data with Shiny apps is especially relevant when results involve a large number of variables, analyses are complex, or contributions are methodological in nature, but the versatility of Shiny apps makes them useful for almost any research project. Having the possibility to interact with the data can enhance understanding, engagement, and ultimately the impact of a research project.

Concluding remarks

A vast literature indicates that when used appropriately, dynamic visualizations can promote understanding and retention of various scientific findings and concepts (e.g., Ryoo & Linn, 2012; Suits & Sanger, 2013; Yang et al., 2015). Not only can they facilitate conveying and streamlining information at the time of publication, but dynamic visualizations can also help communication among team members at earlier stages of a project and dissemination of findings via talks, conferences, and in the media after publication. Furthermore, when projects have the potential to be continuously updated—for example, in the case of longitudinal studies or meta-analyses (Braver et al., 2014)—figures that are based on dynamic code can get updated automatically as new data come in, thus ensuring the user has access to the latest, up-to-date information.

In many ways, current publishing models pose a number of challenges to the implementation of this type of content, and a number of journals and publishing platforms are currently working toward developing ways to enable more elaborate content (see e.g., Colomb & Brembs, 2014; Penfold, 2017). The move toward more sophisticated options for visual content is unquestionably the future of scientific publishing, with preregistrations, articles, code, and data hosted together to facilitate evaluation and active exploration of a research project. In the meantime, independent hosting platforms allow moving beyond the traditional format of presentation for scientific projects, and researchers should become familiar with the possibilities and capabilities they afford. We hope these developments will enable a more ubiquitous use of dynamic visualizations to help further the understanding of the complex, multifaceted relationships that govern brains and behaviors.

Transparency

Action Editor: David A. Sbarra
Editor: David A. Sbarra
Author Contributions

Kristina Wiebels: Conceptualization; Data curation; Formal analysis; Investigation; Methodology; Project administration; Resources; Software; Validation; Visualization; Writing – original draft; Writing – review & editing.

David Moreau: Conceptualization; Data curation; Formal analysis; Investigation; Methodology; Project administration; Resources; Software; Validation; Visualization; Writing – original draft; Writing – review & editing.

Declaration of Conflicting Interests

The author(s) declared that there were no conflicts of interest with respect to the authorship or the publication of this article.

Funding

D. Moreau and K. Wiebels are supported by a Marsden grant from the Royal Society of New Zealand and a University of Auckland Early Career Research Excellence Award awarded to D. Moreau.

Open Practices

This article has received the badges for Open Data and Open Materials. More information about the Open Practices badges can be found at http://www.psychologicalscience.org/publications/badges.

ORCID IDs

Kristina Wiebels https://orcid.org/0000-0002-5360-5965
David Moreau https://orcid.org/0000-0002-1957-1941

Notes

1. At the time of writing, the talk had received a combined 20 million views across TED and YouTube platforms.
2. See, for example, *Carbon Dioxide* (n.d.) and Center for Systems Science and Engineering Johns Hopkins University (n.d.).
3. Note that many alternatives exist outside the R language, such as D3.js (https://d3js.org/), dimple (http://dimplejs.org/), Vega (https://vega.github.io/vega/), Tableau (https://www.tableau.com/resource/data-visualization), or Google's Visualization API (https://developers.google.com/chart/interactive/docs/reference). Most of these alternatives provide functionalities above and beyond those available in R and thus may be attractive to users who plan on using dynamic data visualizations extensively and across a number of media (e.g., website, educational resources). However, because they typically require programming knowledge beyond what is assumed in this tutorial, we focus on implementations in R.
4. The use of *ggiraph* or *plotly* for interactive data visualizations depends largely on personal preference. *ggiraph* is often thought to be more straightforward to implement, especially for *ggplot2* users, whereas *plotly* has a number of additional functionalities that can be advantageous in specific circumstances (e.g., toolbar to hide parts of a plot, zooming feature). Here, we present the two alternatives because both are extremely popular and being actively developed, with new features and functionalities being released regularly.

References

Allen, E. A., Erhardt, E. B., & Calhoun, V. D. (2012). Data visualization in the neurosciences: Overcoming the curse of dimensionality. *Neuron*, *74*(4), 603–608. https://doi.org/10.1016/j.neuron.2012.05.001

Bétrancourt, M., & Tversky, B. (2000). Effect of computer animation on users' performance: A review. *Le Travail Humain*, *63*(4), 311–329.

Blok, C. A. (2005). *Dynamic visualization variables in animation to support monitoring of spatial phenomena*. Netherlands Geographical Studies.

Borland, D., & Taylor, M. R., II. (2007). Rainbow color map (still) considered harmful. *IEEE Computer Graphics and Applications*, *27*(2), 14–17.

Braver, S. L., Thoemmes, F. J., & Rosenthal, R. (2014). Continuously cumulating meta-analysis and replicability. *Perspectives on Psychological Science*, *9*(3), 333–342.

Carbon dioxide. (n.d.). https://climate.nasa.gov/vital-signs/carbon-dioxide/

Chang, W., Cheng, J., Allaire, J. J., Sievert, C., Schloerke, B., Xie, Y., Allen, J., McPherson, J., Dipert, A., & Borges, B. (2022). *Shiny: Web application framework for R*. https://CRAN.R-project.org/package=shiny

Colomb, J., & Brembs, B. (2014). Sub-strains of *Drosophila* Canton-S differ markedly in their locomotor behavior. *F1000Research*, *3*, Article 176. https://doi.org/10.12688/f1000research.4263.2

Cordero, R. J. B., de León-Rodriguez, C. M., Alvarado-Torres, J. K., Rodriguez, A. R., & Casadevall, A. (2016). Life science's average publishable unit (APU) has increased over the past two decades. *PLOS ONE*, *11*(6), Article e0156983 https://doi.org/10.1371/journal.pone.0156983

Center for Systems Science and Engineering (CSSE) at Johns Hopkins University (JHU). (n.d.). *COVID-19 dashboard*. https://www.arcgis.com/apps/opsdashboard/index.html#/bda7594740fd40299423467b48e9ecf6

Deploying Shiny apps to the web. (2017). https://shiny.rstudio.com/articles/deployment-web.html

Doumont, J., & Vandenbroeck, P. (2002). Choosing the right graph. *IEEE Transactions on Professional Communication*, *45*(1), 1–6.

Fawcett, L. (2018). Using interactive shiny applications to facilitate research-informed learning and teaching. *Journal of Statistics Education*, *26*(1), 2–16.

Few, S. (2004). *Show me the numbers*. Analytics Press.

Friendly, M. (2008). A brief history of data visualization. In C.-H. Chen, W. Härdle, & A. Unwin (Eds.), *Handbook of data visualization* (pp. 15–56). Springer.

Gohel, D., & Skintzos, P. (2022). *Ggiraph: Make "ggplot2" graphics interactive*. https://CRAN.R-project.org/package=ggiraph

Gohel, D., & Skintzos, P. (2023). Ggiraph: Make 'ggplot2' *graphics interactive. R package version 0.8.7*. https://davidgohel.github.io/ggiraph/

Isenberg, P., Elmqvist, N., Scholtz, J., Cernea, D., Ma, K.-L., & Hagen, H. (2011). Collaborative visualization: Definition, challenges, and research agenda. *Information Visualization*, *10*(4), 310–326.

Kamath, C. (2001). On mining scientific datasets. In R. L. Grossman, C. Kamath, P. Kegelmeyer, V. Kumar, & R. R. Namburu (Eds.), *Data mining for scientific and engineering applications* (pp. 1–21). Springer.

Kelleher, C., & Wagener, T. (2011). Ten guidelines for effective data visualization in scientific publications. *Environmental Modelling & Software*, *26*(6), 822–827.

Kennedy, S. J. (2012). *Transforming big data into knowledge: Experimental techniques in dynamic visualization*. Massachusetts Institute of Technololgy. http://hdl.handle.net/1721.1/73818

Kerren, A., & Stasko, J. T. (2002). Algorithm animation. In S. Diehl (Ed.), *Software visualization* (pp. 1–15). Springer.

Lewalter, D. (2003). Cognitive strategies for learning from static and dynamic visuals. *Learning and Instruction*, *13*(2), 177–189.

Midway, S. R. (2020). Principles of effective data visualization. *Patterns*, *1*(9), Article 100141. https://doi.org/10.1016/j.patter.2020.100141

Moreau, D. (2015). When seeing is learning: Dynamic and interactive visualizations to teach statistical concepts. *Frontiers in Psychology*, *6*, Article 342. https://doi.org/10.3389/fpsyg.2015.00342

Newman, G. E., & Scholl, B. J. (2012). Bar graphs depicting averages are perceptually misinterpreted: The within-the-bar bias. *Psychonomic Bulletin & Review*, *19*(4), 601–607.

Nordmann, E., McAleer, P., Toivo, W., Paterson, H., & DeBruine, L. M. (2021). *Data visualisation using R, for researchers who don't use R*. PsyArXiv. https://doi.org/10.31234/osf.io/4huvw

Ospina, R., Larangeiras, A. M., & Frery, A. C. (2014). Visualization of skewed data. *Revista Colombiana de Estadistica*, *37*(2Spe), 399–417.

Pedersen, T. L., & Robinson, D. (n.d.). *Control easing of aesthetics*. https://gganimate.com/reference/ease_aes.html

Pedersen, T. L., & Robinson, D. (2020). *Gganimate: A grammar of animated graphics*. https://CRAN.R-project.org/package=gganimate

Pedersen, T L., & Robinson, D. (2022). *Gganimate: A grammar of animated graphics*. https://github.com/thomasp85/gganimate.

Penfold, N. (2017). *Reproducible Document Stack – Supporting the next-generation research article*. eLife Sciences Publications Limited. https://elifesciences.org/labs/7dbeb390/reproducible-document-stack-supporting-the-next-generation-research-article

Plotly R open source graphing library. (n.d.). https://plotly.com/r

R Core Team. (2020). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Robertson, G., Fernandez, R., Fisher, D., Lee, B., & Stasko, J. (2008). Effectiveness of animation in trend visualization. *IEEE Transactions on Visualization and Computer Graphics*, *14*(6), 1325–1332.

Rolfes, T., Roth, J., & Schnotz, W. (2020). Learning the concept of function with dynamic visualizations. *Frontiers in Psychology*, *11*, Article 693. https://doi.org/10.3389/fpsyg.2020.00693

Rosling, H. (2006). *The best stats you've ever seen* [Video]. TED. https://www.ted.com/talks/hans_rosling_the_best_stats_you_ve_ever_seen

Rougier, N. P., Droettboom, M., & Bourne, P. E. (2014). Ten simple rules for better figures. *PLOS Computational Biology*, *10*(9), Article e1003833. https://doi.org/10.1371/journal.pcbi.1003833

Rstudio Team. (2020). *Rstudio: Integrated development environment for R*. Rstudio, Inc. http://www.rstudio.com/

Ryoo, K., & Linn, M. C. (2012). Can dynamic visualizations improve middle school students' understanding of energy in photosynthesis? *Journal of Research in Science Teaching*, *49*(2), 218–243.

Sharing apps to run locally. (2014). https://shiny.rstudio.com/articles/deployment-local.html

Shiny learning resources. (n.d.). https://shiny.rstudio.com/tutorial/

Sievert, C. (2020). *Interactive web-based data visualization with R, plotly, and shiny*. Chapman and Hall/CRC.

Steele, J., & Iliinsky, N. (2010). *Beautiful visualization: Looking at data through the eyes of experts*. O'Reilly Media, Inc.

Suits, J. P., & Sanger, M. J. (2013). Dynamic visualizations in chemistry courses. In J. P. Suits & M. J. Sanger (Eds.), *Pedagogic roles of animations and simulations in chemistry courses* (Vol. 1142, pp. 1–13). American Chemical Society.

Ushey, K. (2021). *Renv: Project environments*. https://CRAN.R-project.org/package=renv

Ward, M. O., Grinstein, G., & Keim, D. (2011). *Interactive data visualization*. A K Peters/CRC Press.

Weiss, R. E., Knowlton, D. S., & Morrison, G. R. (2002). Principles for using animation in computer-based instruction: Theoretical heuristics for effective design. *Computers in Human Behavior*, *18*(4), 465–477.

Weissgerber, T. L., Milic, N. M., Winham, S. J., & Garovic, V. D. (2015). Beyond bar and line graphs: Time for a new data presentation paradigm. *PLOS Biology*, *13*(4), Article e1002128. https://doi.org/10.1371/journal.pbio.1002128

Wickham, H. (2016). *Ggplot2: Elegant graphics for data analysis*. Springer-Verlag.

Wickham, H. (2021). *Mastering Shiny*. O'Reilly Media.

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., Woo, K., Yutani, H., & Dunnington, D. (n.d.). *Complete themes*. https://ggplot2.tidyverse.org/reference/ggtheme.html#ref-usage

Wiebels, K., Addis, D. R., Moreau, D., van Mulukom, V., Onderdijk, K. E., & Roberts, R. P. (2020). Relational processing demands and the role of spatial context in the construction of episodic simulations. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *46*(8), 1424–1441. https://doi.org/10.1037/xlm0000831

Wiebels, K., & Moreau, D. (2021). Leveraging containers for reproducible psychological research. *Advances in Methods and Practices in Psychological Science*, *4*(2). https://doi.org/10.1177/25152459211017853

Xie, Y. (2013). Animation: An R package for creating animations and demonstrating statistical methods. *Journal of Statistical Software*, *53*(1), 1–27. https://doi.org/10.18637/jss.v053.i0

Yang, J., Lee, Y., Hicks, D., & Chang, K. H. (2015). Enhancing object-oriented programming education using static and dynamic visualization. In *2015 IEEE Frontiers in education conference (FIE)*. IEEE. https://doi.org/10.1109/FIE.2015.7344152

Yu, L., Lu, A., Ribarsky, W., & Chen, W. (2010). Automatic animation for time-varying data visualization. *Computer Graphics Forum*, *29*(7), 2271–2280. https://doi.org/10.1111/j.1467-8659.2010.01816.x