DATA SOCIETY®

Interactive visualization - R

"One should look for what is and not what he thinks should be."
-Albert Einstein.

Module completion checklist

Objective	Complete
Transform data using dplyr, tidyr to prepare for compound visualizations with ggplot2	
Visualize transformed data using compound univariate visualizations	
Visualize transformed data using compound bivariate visualizations	
Explain the integration of the highcharter package	
Create basic interactive visualizations with transformed data	
Create interactive visualizations with transformed summary data	
Create complex interactive visualizations with multiple metrics and variables	

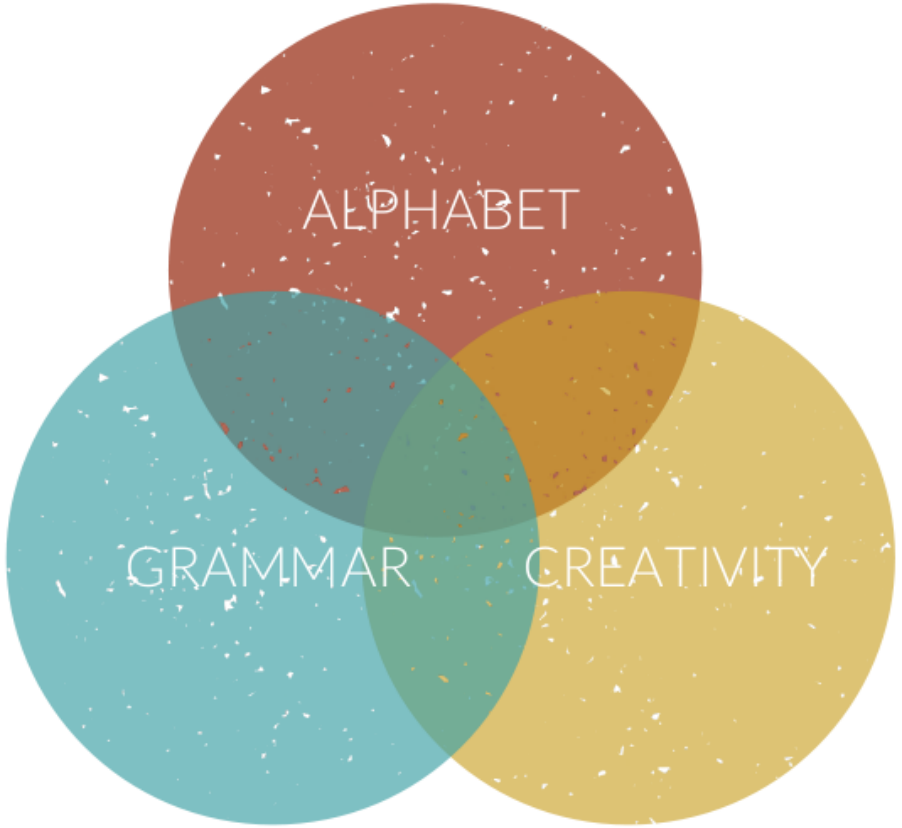
Directory settings

- In order to maximize the efficiency of your workflow, you may want to encode your directory structure into `variables`
- Let the `main_dir` be the variable corresponding to your `af-werx` folder

```
# Set `main_dir` to the location of your `af-werx` folder (for Mac/Linux).
main_dir = "~/Desktop/af-werx"

# Set `main_dir` to the location of your `af-werx` folder (for Windows).
main_dir = "C:/Users/[username]/Desktop/af-werx"

# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory.
data_dir = paste0(main_dir, "/data")

# Make `plots_dir` from the `main_dir` and
# remainder of the path to plots directory.
plot_dir = paste0(main_dir, "/plots")
```

Recap of visualizing data with ggplot2 package

- Explore your data efficiently
- Communicate a visual story creatively and efficiently
- Layer raw, summarized, contextual data and demonstrate relationships
- Reproduce and extend your work easily



Recap of working with ggplot2 package

Setup

1. Specify data
2. Link data to visuals
3. Assign shapes

Adjust

1. Visual effects
2. Axes
3. Legend

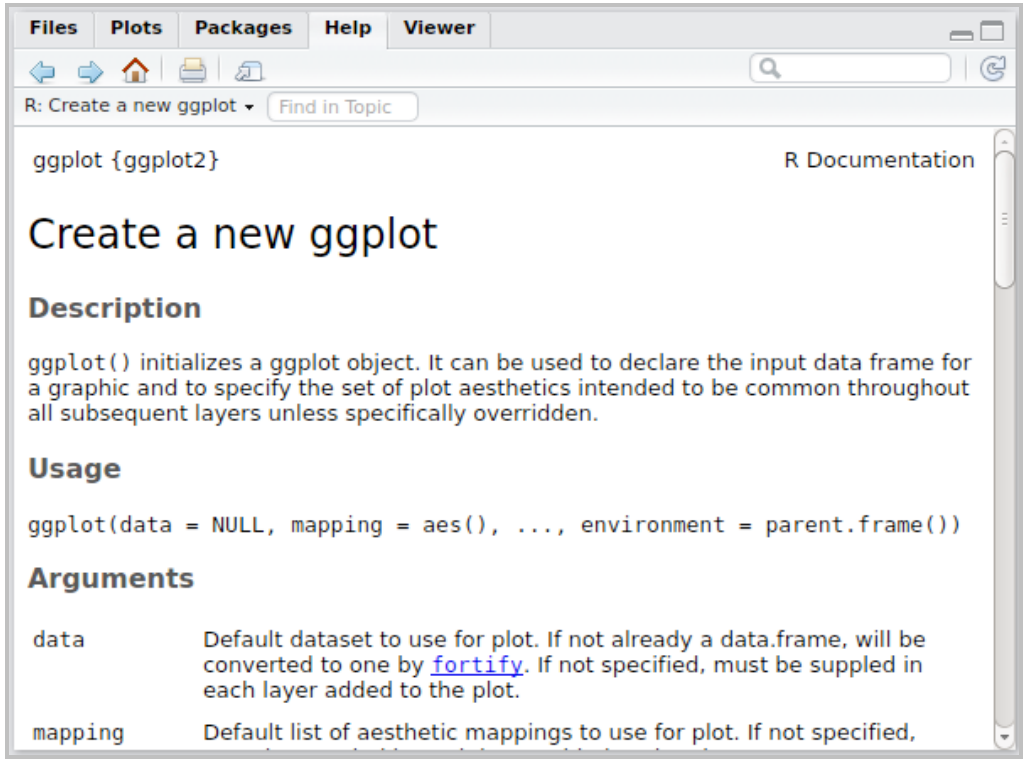
Polish

1. Customize theme
2. Layer statistics
3. Text

Syntax of ggplot2 package

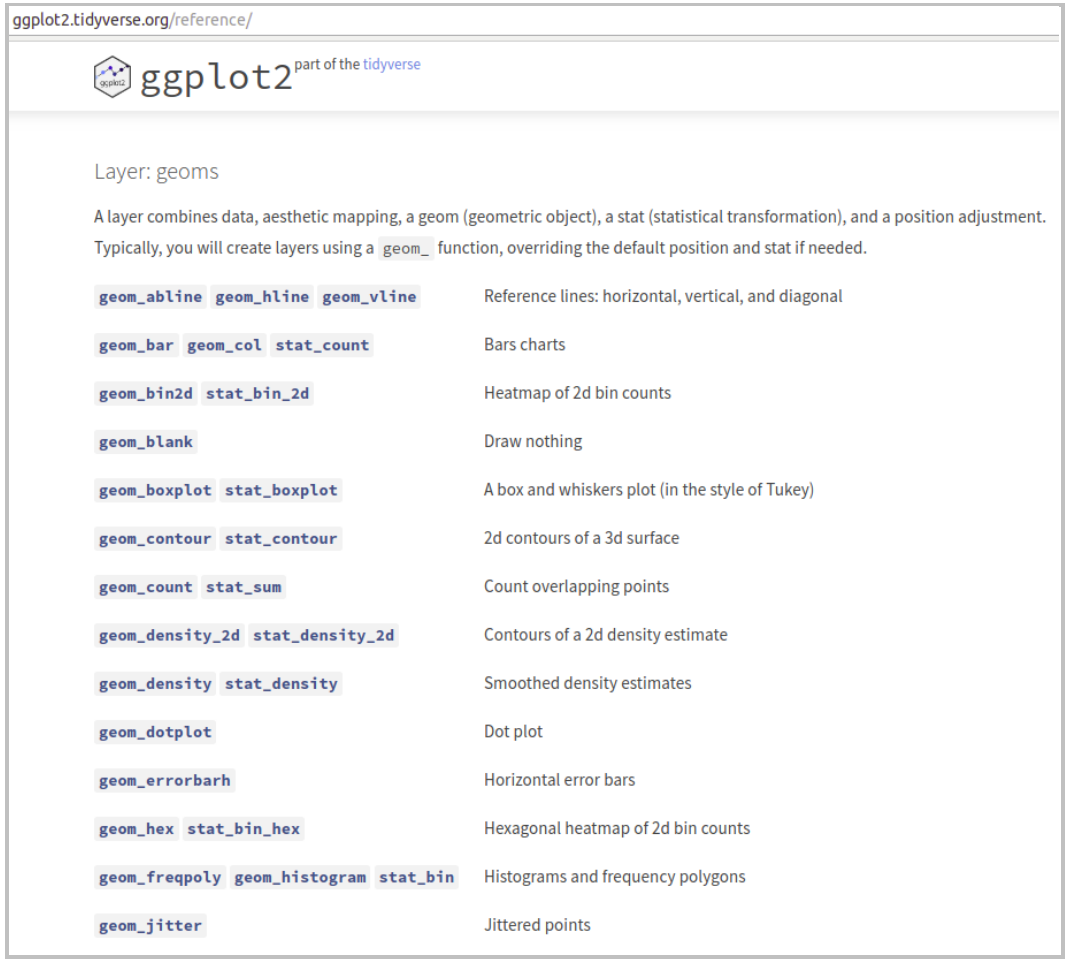
```
?ggplot
```

1. The main function in the package is `ggplot`, it creates the initial plot object (i.e. an empty canvas)
2. The main 2 arguments in the function are
   i. `data`: tells `ggplot` the data to be plotted
   ii. `mapping`: a named list of `aes[thetics]` that lets `ggplot` know which variables are to be mapped to which axes



ggplot2 works best with tidy data

- Each chart in the `ggplot2` package is like a layered cake
- `ggplot2` is a part of the `tidyverse`
- `ggplot2` works best with **tidy data** because it allows us to create complex multi-layered visualizations when we **summarize** and **transform** data


ggplot2.tidyverse.org/reference/

ggplot2 part of the tidyverse

Layer: geoms

A layer combines data, aesthetic mapping, a geom (geometric object), a stat (statistical transformation), and a position adjustment. Typically, you will create layers using a `geom_` function, overriding the default position and stat if needed.

`geom_abline` `geom_hline` `geom_vline`	Reference lines: horizontal, vertical, and diagonal
`geom_bar` `geom_col` `stat_count`	Bars charts
`geom_bin2d` `stat_bin_2d`	Heatmap of 2d bin counts
`geom_blank`	Draw nothing
`geom_boxplot` `stat_boxplot`	A box and whiskers plot (in the style of Tukey)
`geom_contour` `stat_contour`	2d contours of a 3d surface
`geom_count` `stat_sum`	Count overlapping points
`geom_density_2d` `stat_density_2d`	Contours of a 2d density estimate
`geom_density` `stat_density`	Smoothed density estimates
`geom_dotplot`	Dot plot
`geom_errorbarh`	Horizontal error bars
`geom_hex` `stat_bin_hex`	Hexagonal heatmap of 2d bin counts
`geom_freqpoly` `geom_histogram` `stat_bin`	Histograms and frequency polygons
`geom_jitter`	Jittered points

Load tidyverse and set up ggplot2 theme

- Load `tidyverse` package that includes `ggplot2`

```
# Load tidyverse library
# (it includes ggplot2).
library(tidyverse)
```

```
── Attaching packages ─tidyverse 1.2.1
✔ ggplot2 2.2.1     ✔ purrr   0.2.4
✔ tibble  1.4.2     ✔ dplyr   0.7.4
✔ tidyr   0.8.0     ✔ stringr 1.3.0
✔ readr   1.1.1     ✔ forcats 0.3.0
── Conflicts ────tidyverse_conflicts()
✖ tidyr::expand() masks Matrix::expand()
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
```

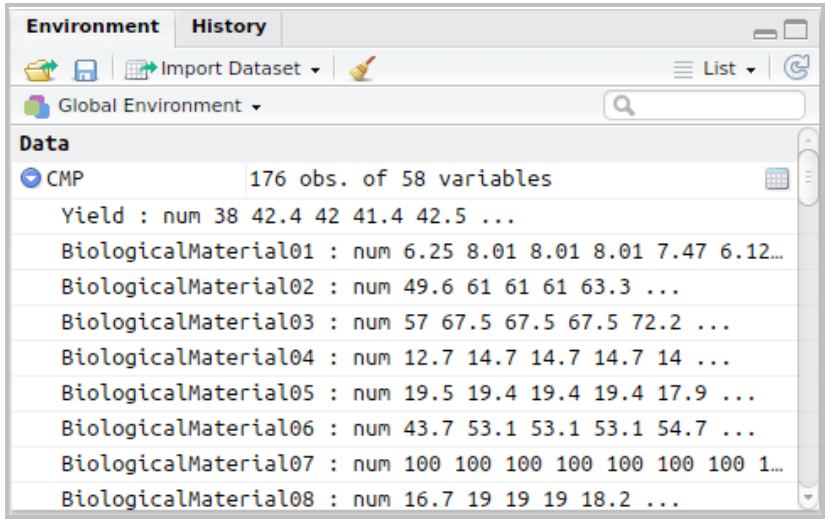
- Remember our saved `ggplot2` theme from a previous lesson

```
# Save our custom `ggplot` theme to a variable.
my_ggtheme = theme_bw() +
  theme(axis.title = element_text(size = 20),
        axis.text = element_text(size = 16),
        legend.text = element_text(size = 16),
        legend.title = element_text(size = 18),
        plot.title = element_text(size = 25),
        plot.subtitle = element_text(size = 18))
```

Setup: load dataset for EDA

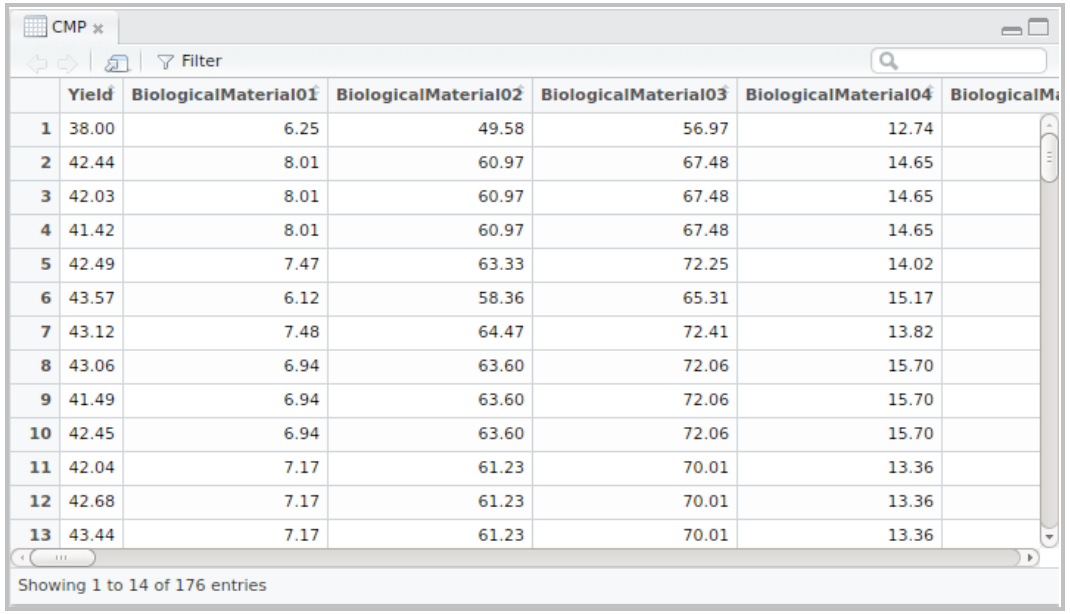
- Let's load the dataset from our `data_dir` into R's environment

```
# Set working directory to where we store data.
setwd(data_dir)

# Read CSV file called
#"ChemicalManufacturingProcess.csv".
CMP =
  read.csv("ChemicalManufacturingProcess.csv",
           header = TRUE)
```



- The dataset consists of `176` observations and `58` variables

```
# View CMP dataset in the tabular data explorer.
View(CMP)
```



	Yield	BiologicalMaterial01	BiologicalMaterial02	BiologicalMaterial03	BiologicalMaterial04	BiologicalMa
1	38.00	6.25	49.58	56.97	12.74	
2	42.44	8.01	60.97	67.48	14.65	
3	42.03	8.01	60.97	67.48	14.65	
4	41.42	8.01	60.97	67.48	14.65	
5	42.49	7.47	63.33	72.25	14.02	
6	43.57	6.12	58.36	65.31	15.17	
7	43.12	7.48	64.47	72.41	13.82	
8	43.06	6.94	63.60	72.06	15.70	
9	41.49	6.94	63.60	72.06	15.70	
10	42.45	6.94	63.60	72.06	15.70	
11	42.04	7.17	61.23	70.01	13.36	
12	42.68	7.17	61.23	70.01	13.36	
13	43.44	7.17	61.23	70.01	13.36	

Setup: select variables

In this module, we will explore a subset of this dataset, which includes the following variables

- **yield**
- **3 material** variables, and
- **3 process** variables

We will use `select` from `dplyr`

CMP

Filter

	Yield	BiologicalMaterial01	BiologicalMaterial02	BiologicalMaterial03
1	38.00	6.25	49.58	56.97
2	42.44	8.01	60.97	67.48
3	42.03	8.01	60.97	67.48
4	41.42	8.01	60.97	67.48
5	42.49	7.47	63.33	72.25
6	43.57	6.12	58.36	65.31
7	43.12	7.48	64.47	72.41
8	43.06	6.94	63.60	72.06
9	41.49	6.94	63.60	72.06

Showing 1 to 10 of 176 entries

...

CMP

Filter

ManufacturingProcess01	ManufacturingProcess02	ManufacturingProcess03
NA	NA	NA
0.0	0.0	NA
0.0	0.0	NA
0.0	0.0	NA
10.7	0.0	NA
12.0	0.0	NA
11.5	0.0	1.56
12.0	0.0	1.55
12.0	0.0	1.56

Setup: subset data with select

```
# Select variables from CMP data.
CMP_subset = select(CMP,                                          #<- set data
                    Yield:BiologicalMaterial03,                   #<- range of columns to select
                    ManufacturingProcess01:ManufacturingProcess03) #<- range of columns to select

# Inspect the first few observations in the subset.
head(CMP_subset)
```

```
  Yield BiologicalMaterial01 BiologicalMaterial02 BiologicalMaterial03
1 38.00                6.25                49.58                56.97
2 42.44                8.01                60.97                67.48
3 42.03                8.01                60.97                67.48
4 41.42                8.01                60.97                67.48
5 42.49                7.47                63.33                72.25
6 43.57                6.12                58.36                65.31
  ManufacturingProcess01 ManufacturingProcess02 ManufacturingProcess03
1                     NA                     NA                     NA
2                    0.0                      0                     NA
3                    0.0                      0                     NA
4                    0.0                      0                     NA
5                   10.7                      0                     NA
6                   12.0                      0                     NA
```

Setup: wide to long data conversion with gather

```
CMP_subset_long = CMP_subset %>%
  gather(key = "variable",
         value = "value")
```

```
# Inspect the first few observations.
head(CMP_subset_long)
```

```
  variable value
1    Yield 38.00
2    Yield 42.44
3    Yield 42.03
4    Yield 41.42
5    Yield 42.49
6    Yield 43.57
```

```
# Inspect the last few observations.
tail(CMP_subset_long)
```

```
                     variable value
1227 ManufacturingProcess03  1.54
1228 ManufacturingProcess03  1.54
1229 ManufacturingProcess03  1.56
1230 ManufacturingProcess03  1.55
1231 ManufacturingProcess03  1.55
1232 ManufacturingProcess03  1.55
```

Setup: data cleaning with mutate

- Let's make a few edits to the data before we plot it

```
# Make names of processes and materials
# more user friendly and readable.
CMP_subset_long = CMP_subset_long %>%

     # Replace `Biological` with `Bio`.
     mutate(variable =
              str_replace(variable,     #<- in column `variable`
                          "Biological", #<- replace "Biological"
                          "Bio ")) %>%  #<- with "Bio "

     # Replace `Manufacturing` with `Man.`.
     mutate(variable =
              str_replace(variable,
                          "Manufacturing",
                          "Man. ")) %>%

     # Remove `0` from numbering.
     mutate(variable =
              str_replace(variable,
                          "0",
                          " "))
```

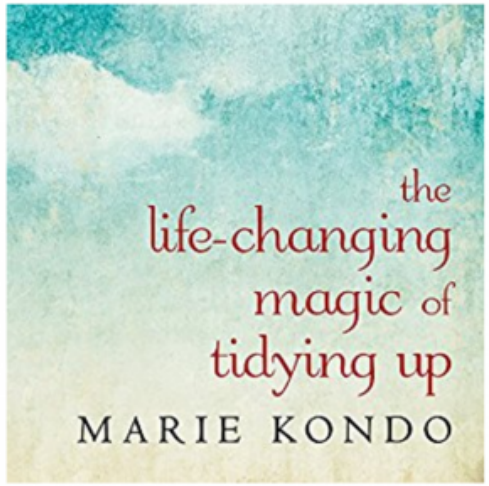
```
# Inspect few first
# entries in the data.
head(CMP_subset_long)
```

```
  variable value
1    Yield 38.00
2    Yield 42.44
3    Yield 42.03
4    Yield 41.42
5    Yield 42.49
6    Yield 43.57
```

```
# Inspect few last
# entries in the data.
tail(CMP_subset_long)
```

```
             variable value
1227 Man. Process 3  1.54
1228 Man. Process 3  1.54
1229 Man. Process 3  1.56
1230 Man. Process 3  1.55
1231 Man. Process 3  1.55
1232 Man. Process 3  1.55
```

Setup: tidy data is ready for ggplot



- Since we have transformed the `CMP_subset` data to be **tidy**, we can now discover `ggplot`'s full potential.

- In the next module, we will compare variables and their distributions to each other using `ggplot`

Module completion checklist

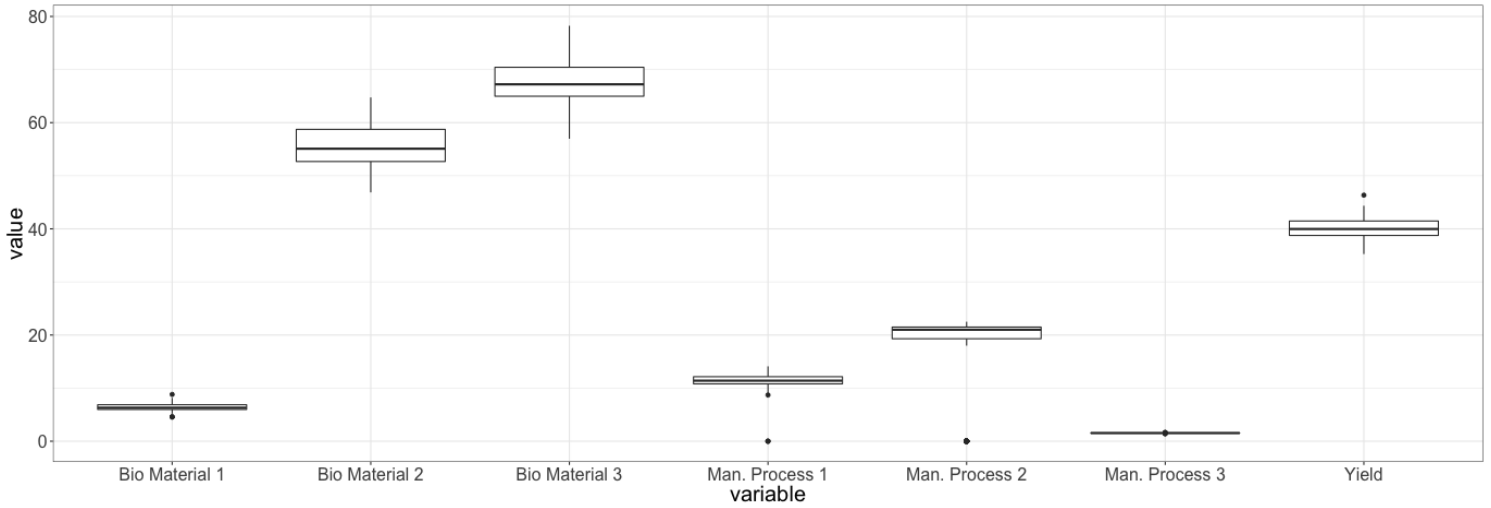
Objective	Complete
Transform data using dplyr, tidyr to prepare for compound visualizations with ggplot2	✔
Visualize transformed data using compound univariate visualizations	
Visualize transformed data using compound bivariate visualizations	
Explain the integration of the highcharter package	
Create basic interactive visualizations with transformed data	
Create interactive visualizations with transformed summary data	
Create complex interactive visualizations with multiple metrics and variables	

Using ggplot is like assembling a layered cake

- **Making data tidy** is similar to **getting ingredients** and following the recipe
- Now it is time for **assembly** - using `ggplot`
- Let's use a **Cinderella cake process**

Set up & link data: make boxplots

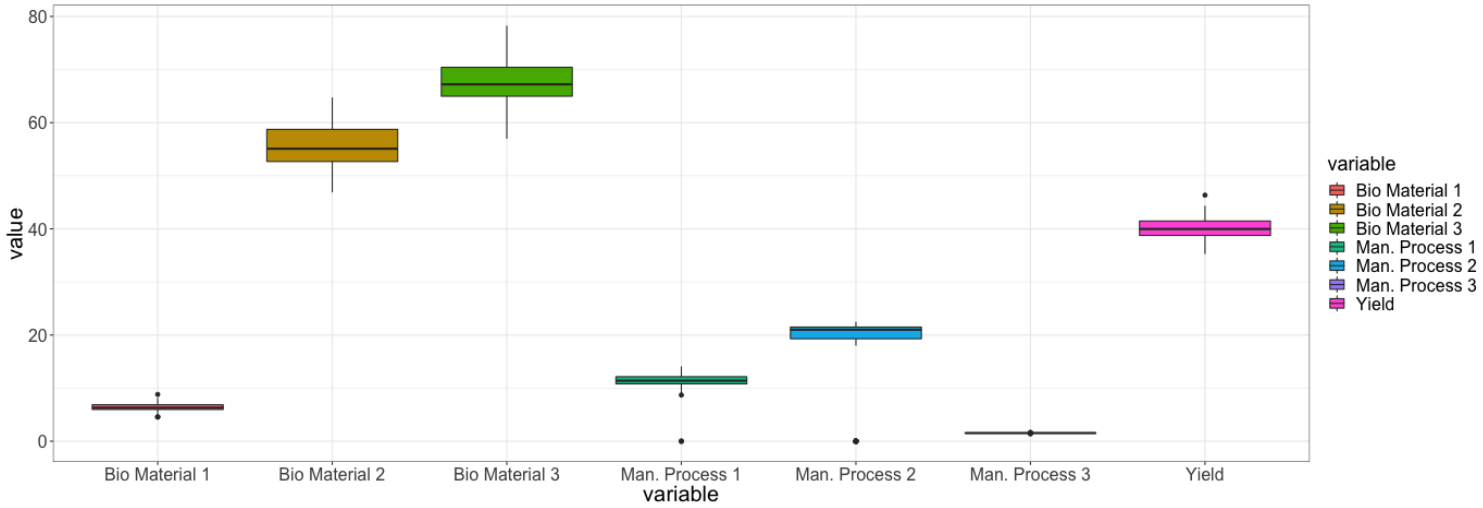
```
# A basic boxplot with pre-saved theme.
boxplots = ggplot(CMP_subset_long,    #<- set the base plot + data
                  aes(x = variable,   #<- map `variable` to x-axis
                      y = value)) +   #<- map `value` to y-axis
           geom_boxplot() +           #<- add boxplot geom
           my_ggtheme                 #<- add pre-saved theme

# View plot.
boxplots
```



- Start with a good foundation ✔

Adjust: boxplot aesthetics

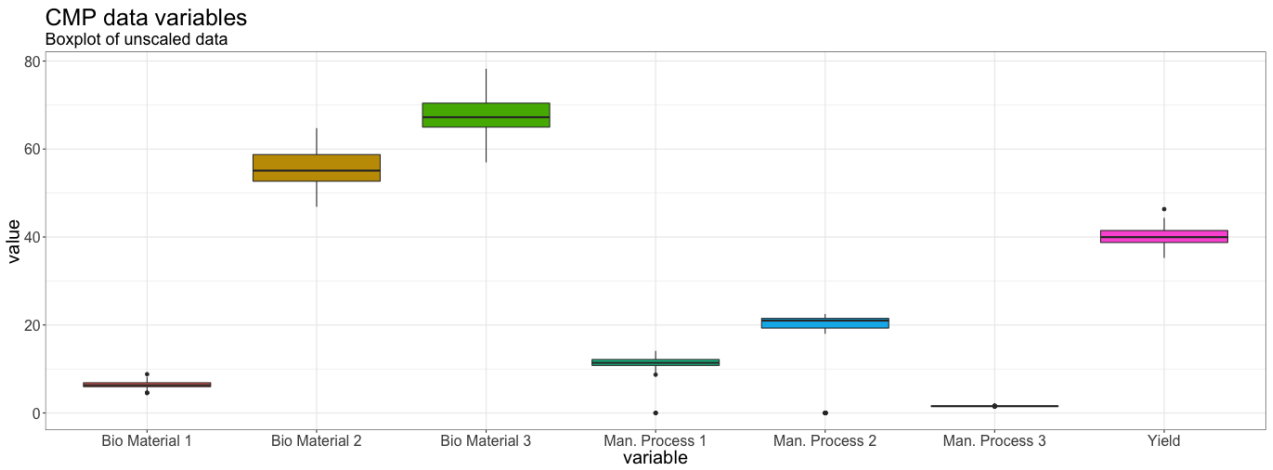
```
# Make color of fill based on variable.
boxplots = ggplot(CMP_subset_long,
                  aes(x = variable,
                      y = value,
                      fill = variable)) + #<- add fill to aesthetics
  geom_boxplot() +
  my_ggtheme

# View plot.
boxplots
```



- Assess and adjust ✔

Adjust & polish: boxplot legends

```
# Remove redundant legend.
boxplots = boxplots +
  labs(title = "CMP data variables", #<- add title and
subtitle
       subtitle = "Boxplot of unscaled data") +
  guides(fill = FALSE)                #<- remove fill legend

# View plot.
boxplots
```



- Adjust & polish ✔

Normalize the data

- The boxplots are hard to read, and interpret as some of them are too squished due to very big differences in value ranges of the variables
- We need to **normalize the data** in order to be able to see how each variable's distribution compares to each other on the same scale

Setup: normalize data with group_by + mutate

```
# Normalize the CMP data.
CMP_subset_long = CMP_subset_long %>%
  group_by(variable) %>%                #<- group values by variable
  mutate(norm_value =                   #<- make `norm_value` column
           value/max(value,             #<- divide value by group max
                     na.rm = TRUE))     #<- don't forget the NAs!

CMP_subset_long
```

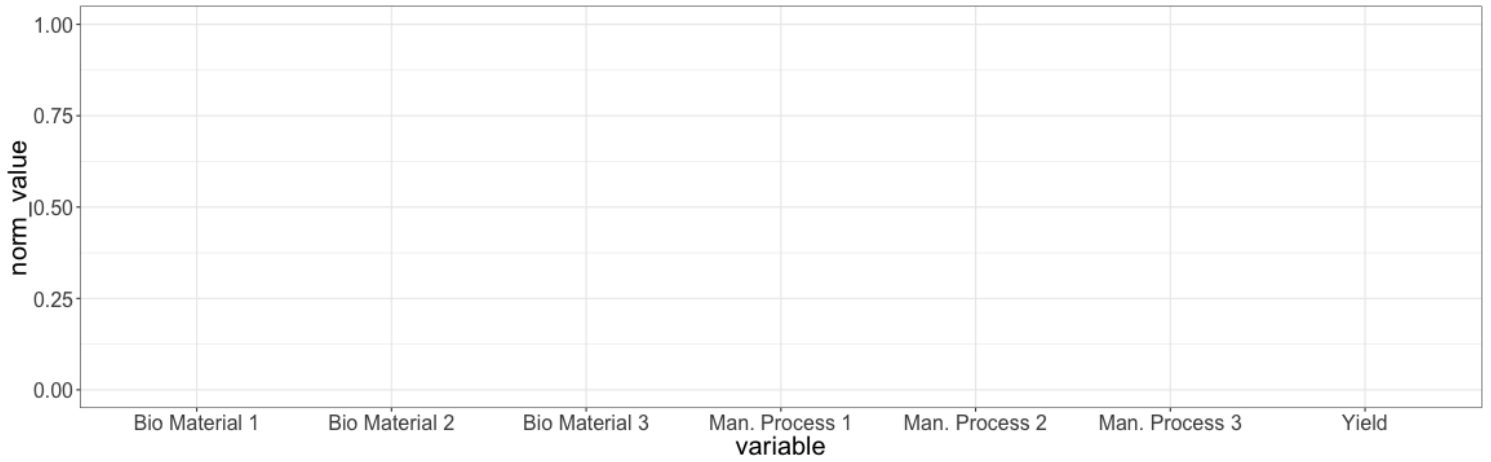
```
# A tibble: 1,232 x 3
# Groups:   variable [7]
   variable value norm_value
   <chr>    <dbl>      <dbl>
 1 Yield     38        0.820
 2 Yield     42.4      0.916
 3 Yield     42.0      0.907
 4 Yield     41.4      0.894
 5 Yield     42.5      0.917
 6 Yield     43.6      0.940
 7 Yield     43.1      0.931
 8 Yield     43.1      0.929
 9 Yield     41.5      0.895
10 Yield     42.4      0.916
# … with 1,222 more rows
```

- Often you need to go back and re-assess your ingredients ✔

Setup: make base plot with normalized data

- Let's save the base plot with normalized data to a variable
- Add `my_ggtheme` to it

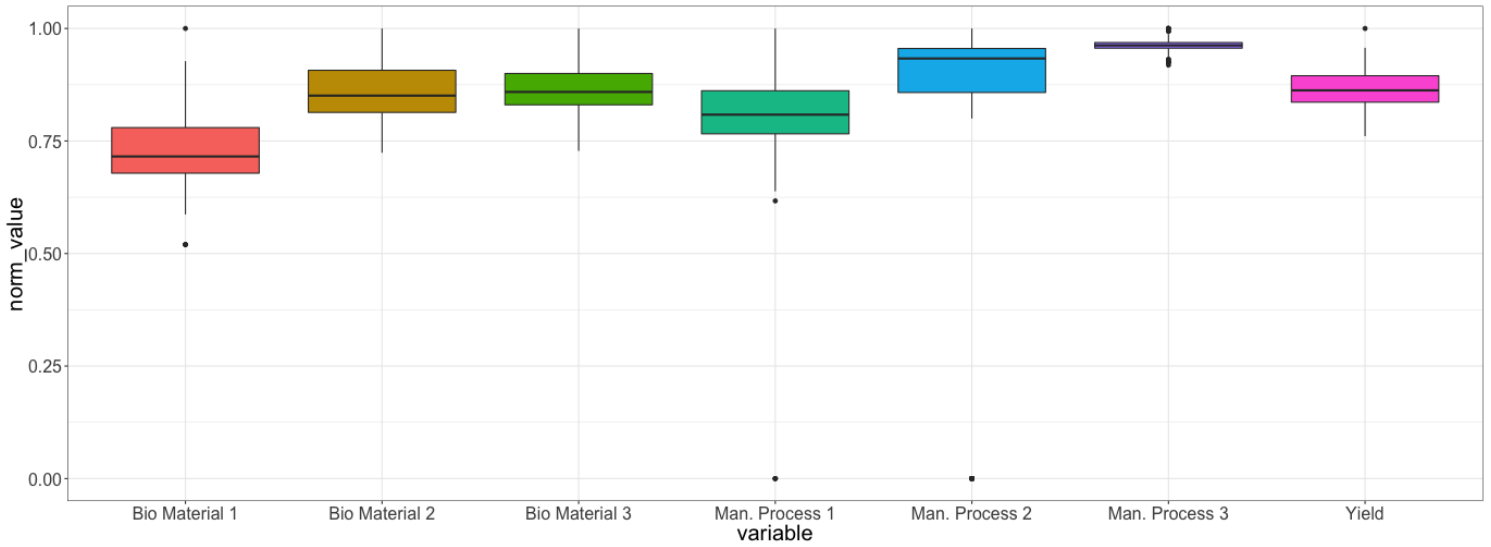
```
# Construct the base plot for normalized data.
base_norm_plot = ggplot(CMP_subset_long,          #<- set data
                        aes(x = variable,         #<- map `variable`
                            y = norm_value,       #<- map `norm_value`
                            fill = variable)) +   #<- set fill
                 my_ggtheme                       #<- add theme

# View base plot + theme.
base_norm_plot
```



- Then, start constructing the plot again from a good foundation ✔

Adjust: normalized boxplot's effects & legends

```
# Make color of fill based on variable.
boxplots_norm = base_norm_plot +      #<- set base plot
                geom_boxplot() +      #<- add geom
                guides(fill = FALSE) #<- remove redundant legend

# View updated plot.
boxplots_norm
```

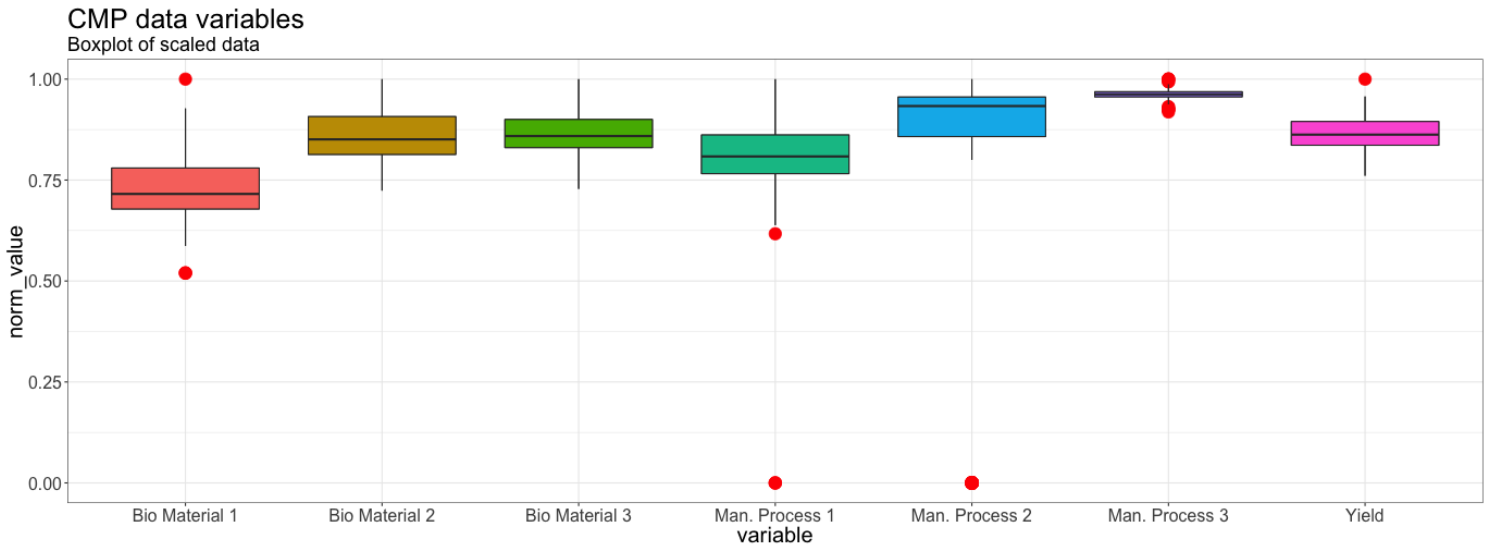


- It looks pretty good! But the outliers aren't very clear - let's adjust their parameters so it's easier for the users to see

- Assessment and adjustment ✔

Polish: normalized boxplot details

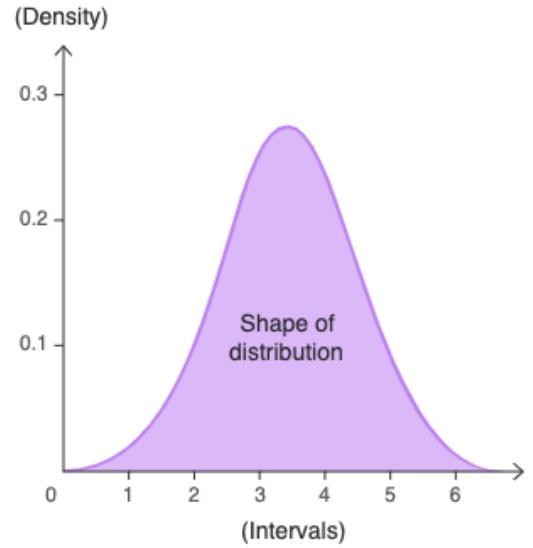
```
# Make outliers stand out with red color and bigger size.
boxplots_norm = boxplots_norm +        #<- previously saved plot
  geom_boxplot(outlier.color = "red", #<- adjust outlier color
               outlier.size = 5) +    #<- adjust outlier size
  labs(title = "CMP data variables",  #<- add title and subtitle
       subtitle = "Boxplot of scaled data")

# View updated plot.
boxplots_norm
```



- Add finishing touches to make end result visually appealing ✔

Density plots

- A density plot visualizes the distribution of data over continuous interval or time period
- This plot is a variation of histogram and allows smoother distributions
- The peaks of density plot help display where values are concentrated over the interval
- Density plots are best at determining the distribution shape

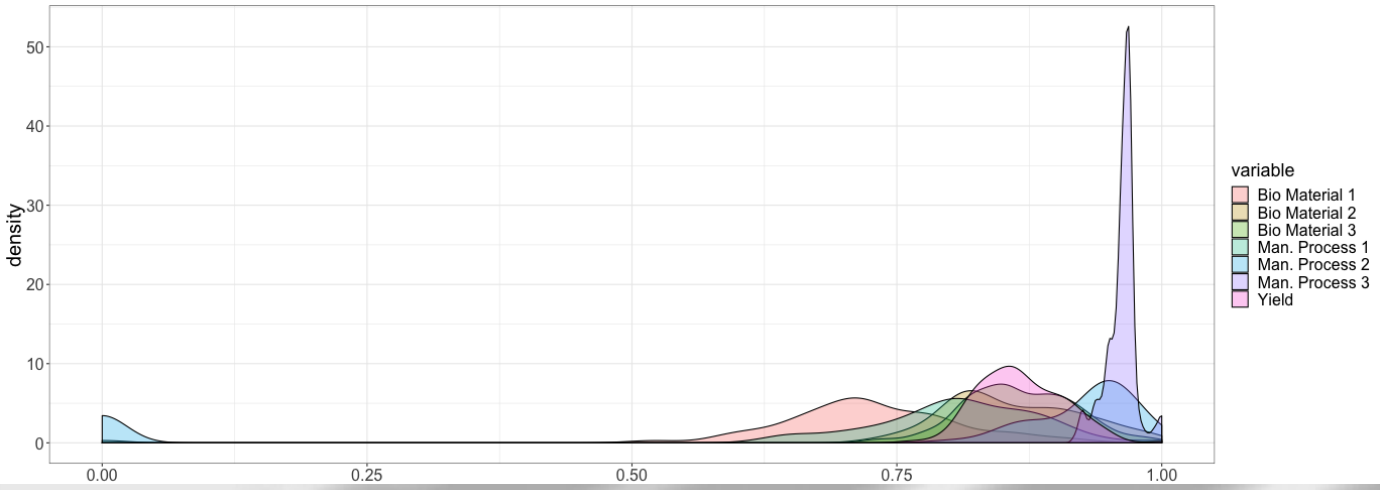


Setup the density plot of normalized data

- Start with your base and adjust
- Notice that this format might not be the best way to see as all plots are cluttered together

```
# Let's save base plot for density as well.
density_norm = ggplot(CMP_subset_long,    #<- set data
                      aes(x = norm_value, #<- map
                          fill = variable)) + #<- map fill
                  my_ggtheme  + #<- add theme
                 geom_density(alpha = 0.3) #<- adjust fill
```

```
density_norm
```

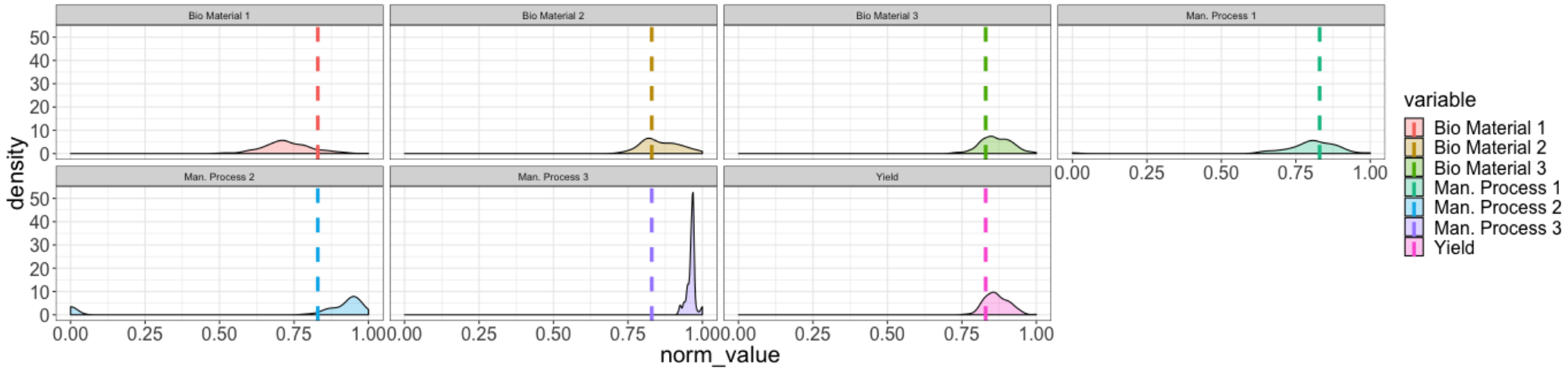


- `alpha` argument is available for every `geom`
- It controls how **opaque** / **transparent** the color is:
  - 1 is **100% opaque** or **0% transparent**
  - 0 is **100% transparent** or **0% opaque**
  - Any decimal value in between will make it more or less opaque / transparent

Adjust the axes and visual effects of density plot

```
# Instead of overlaying densities, split them into individual plots called facets.
density_norm = density_norm +                          #<- previously saved plot
               facet_wrap (~ variable,                 #<- make facets by variable
                           ncol = 4) +                 #<- set a 4-column grid
             geom_vline(data = CMP_subset_long,        #<- set data
                        aes(xintercept =               #<- set x-intercept
                              mean(norm_value,         #<- to mean
                                   na.rm = TRUE),      #<- handle NAs!
                            color = variable),         #<- map color
                        linetype = "dashed",           #<- set line type
                        size = 1.5)                    #<- set line size
```

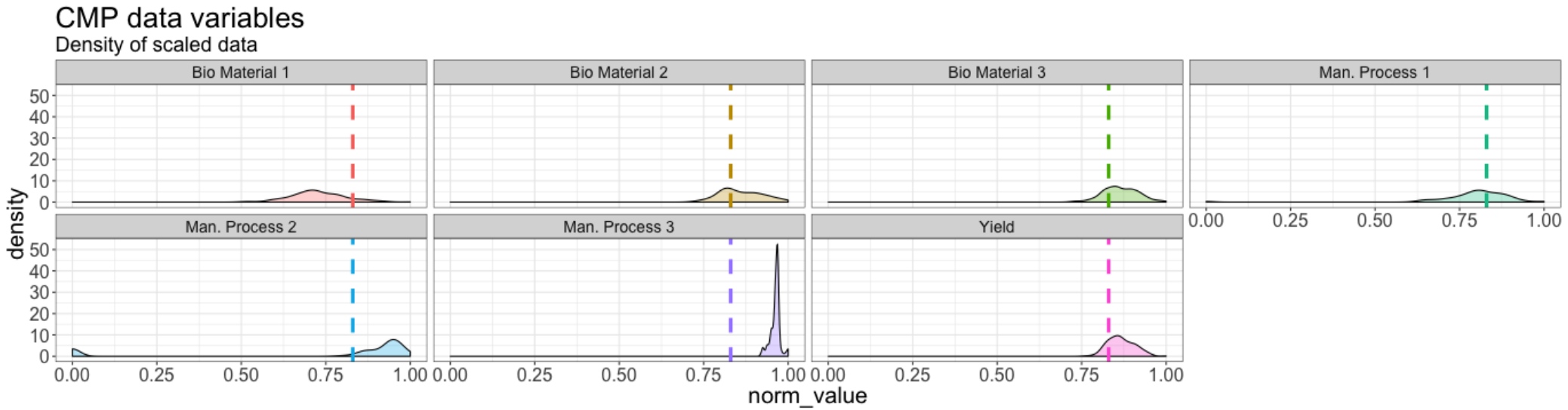
```
# View updated plot.
density_norm
```



Polish: legends of density plot

```
# Remove redundant legend.
density_norm = density_norm +            #<- previously saved plot
               guides(fill = FALSE,      #<- no legend for `fill` of density
                      color = FALSE) +   #<- no legend for `color` of line
              theme(strip.text.x =
          element_text(size = 14)) +     #<- set strip text size
  labs(title = "CMP data variables",     #<- add title and subtitle
       subtitle = "Density of scaled data")
```

```
density_norm
```



Knowledge check 1

Exercise 1

Module completion checklist

Objective	Complete
Transform data using dplyr, tidyr to prepare for compound visualizations with ggplot2	✔
Visualize transformed data using compound univariate visualizations	✔
Visualize transformed data using compound bivariate visualizations	
Explain the integration of the highcharter package	
Create basic interactive visualizations with transformed data	
Create interactive visualizations with transformed summary data	
Create complex interactive visualizations with multiple metrics and variables	

Setup: scatterplots work best with long data

We are going to use the `CMP` data to predict `Yield` when we learn more about classification and regression methods

- It would be beneficial to look at the **bivariate relationships** between the material and process variables and `Yield`
- We will use **scatterplots** to demonstrate such relationships
- In order to plot them, we need to do a couple of things

- Need differently structured ingredients

Setup: transform data for scatterplot

- Separate `Yield` from all other variables
- Make sure that each entry in `Yield` corresponds to an entry in all other variables
- The best way to do it is to convert the original wide data to long format excluding the `Yield` variable

Setup: transform data for scatterplot

```
CMP_subset_long2 = CMP_subset %>%
  gather(BiologicalMaterial01:ManufacturingProcess03, #<- gather all variables but `Yield`
         key = "variable",                            #<- set key to `variable`
         value = "value") %>%                         #<- set value to `value`
  # All other transformations we've done before.
  mutate(variable = str_replace(variable, "Biological", "Bio ")) %>%
  mutate(variable = str_replace(variable, "Manufacturing", "Man. ")) %>%
  mutate(variable = str_replace(variable, "0", " ")) %>%
  group_by(variable) %>%
  mutate(norm_value = value/max(value, na.rm = TRUE))

# Inspect the data.
head(CMP_subset_long2)
```

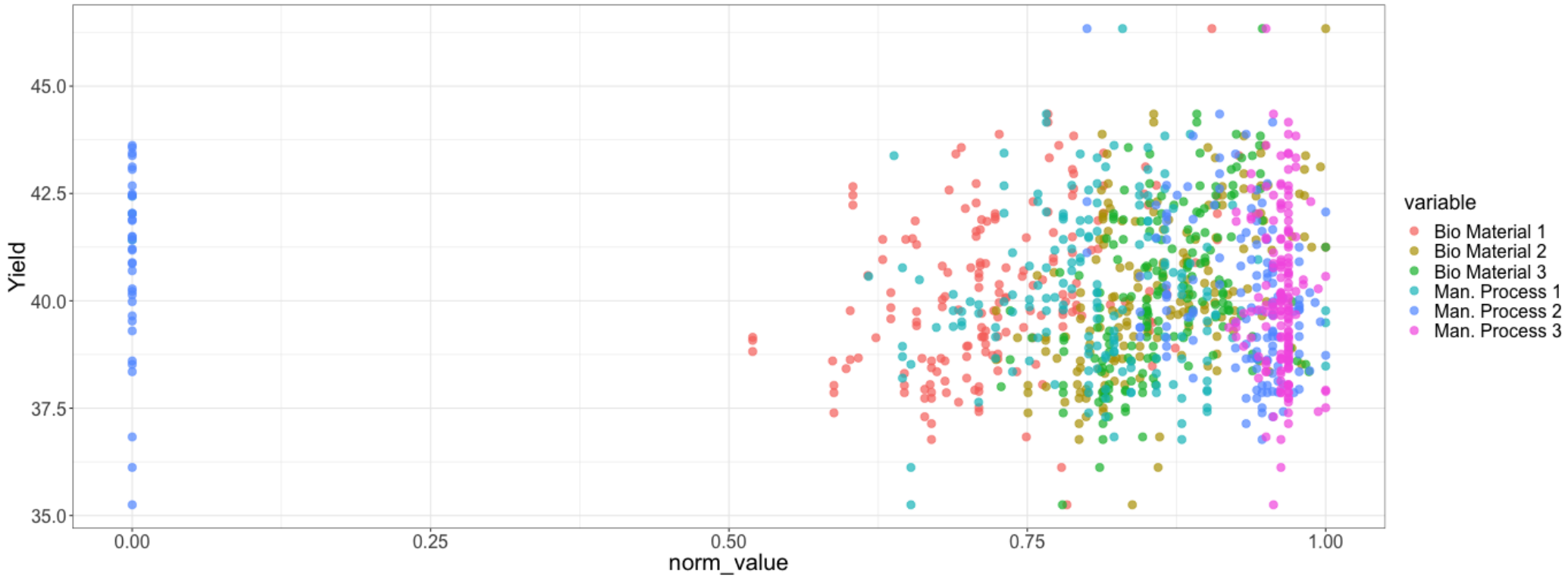
```
# A tibble: 6 x 4
# Groups:   variable [1]
  Yield variable          value norm_value
  <dbl> <chr>             <dbl>      <dbl>
1  38   Bio Material 1  6.25       0.709
2  42.4 Bio Material 1  8.01       0.909
3  42.0 Bio Material 1  8.01       0.909
4  41.4 Bio Material 1  8.01       0.909
5  42.5 Bio Material 1  7.47       0.848
6  43.6 Bio Material 1  6.12       0.695
```

Setup & link: normalized data base plot

```
# Create a base plot.
base_norm_plot = ggplot(data = CMP_subset_long2,   #<- set data
                        aes(x = norm_value,        #<- set x-axis to represent normalized value
                            y = Yield,             #<- y-axis to represent `Yield`
                            color = variable)) +   #<- set color to depend on `variable`
                        my_ggtheme                 #<- set theme
```

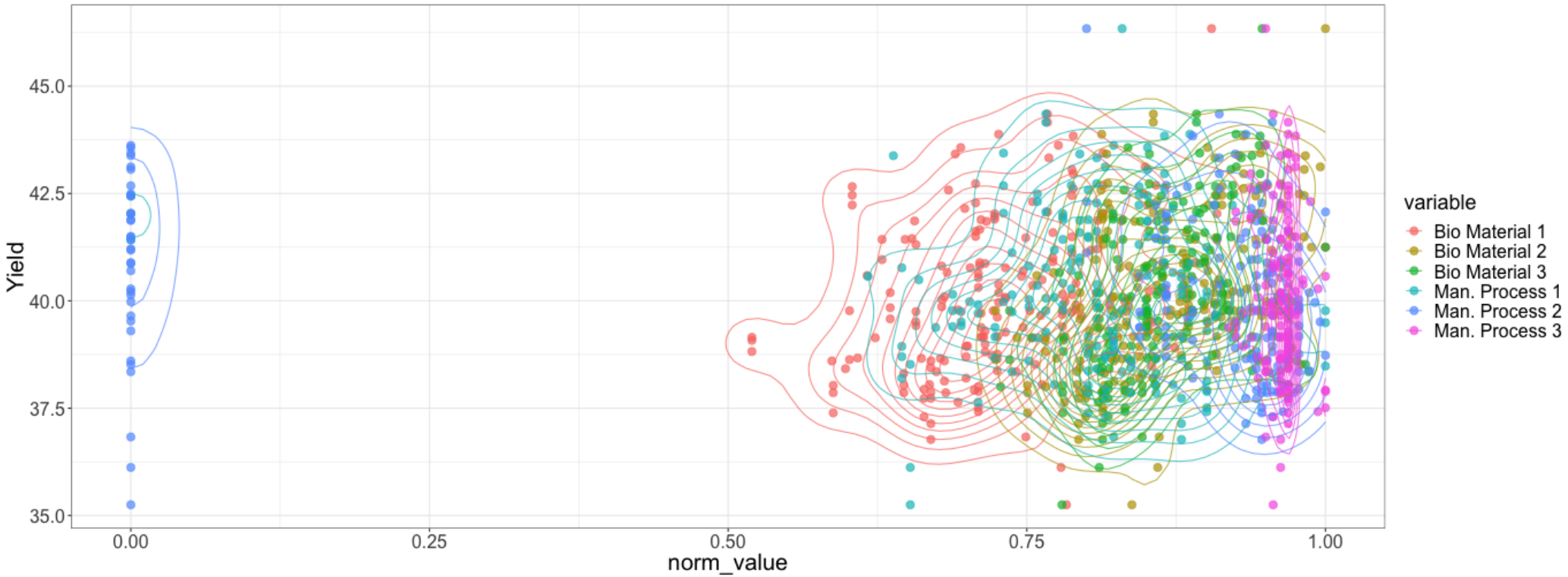
Setup & adjust: normalized data scatterplot

```
# Create a scatterplot.
scatter_norm = base_norm_plot +          #<- base plot
               geom_point(size = 3,      #<- add point geom with size of point = 3
                          alpha = 0.7)  #<- make it 70% opaque

# View updated plot.
scatter_norm
```



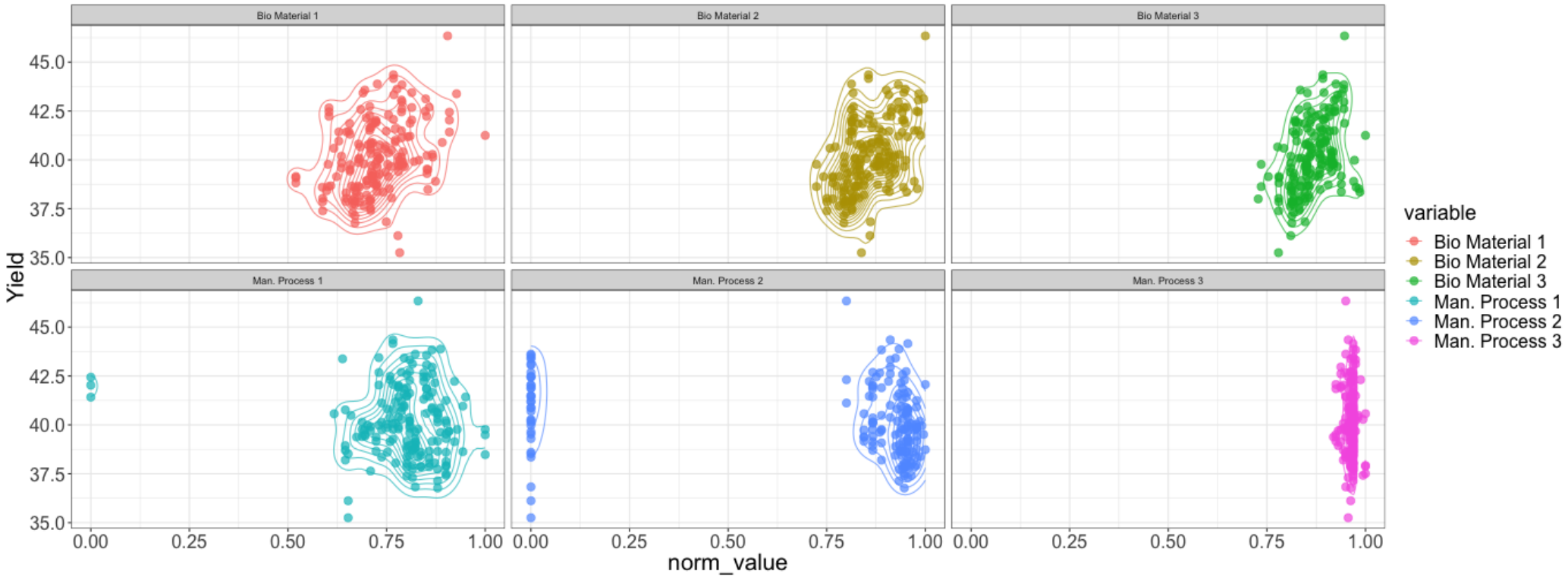
Adjust: add density geom to scatterplot

```
# Adjust scatterplot to include 2D density.
scatter_norm = scatter_norm +                #<- previously saved plot
               geom_density2d(alpha = 0.7)  #<- add 2D density geom with 70% opaque color

# View updated plot.
scatter_norm
```



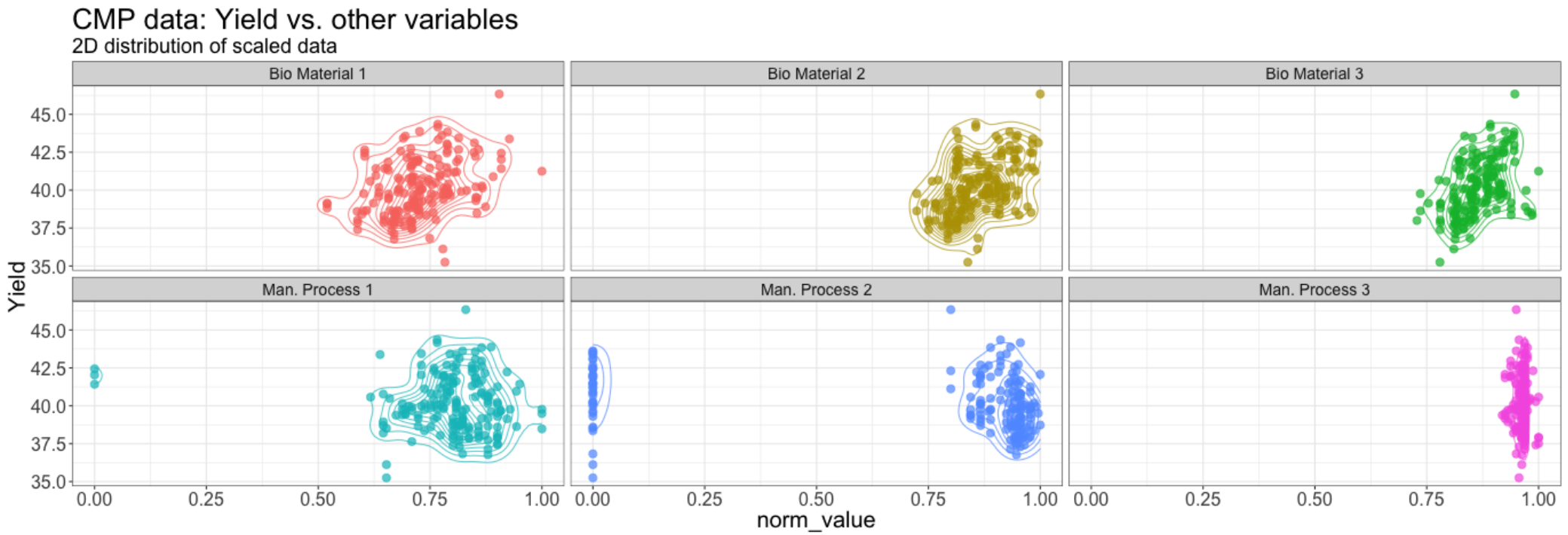
Adjust: wrap scatterplots in facets

```
# Wrap each scatterplot into a facet.
scatter_norm = scatter_norm +                    #<- previously saved plot
               facet_wrap(~ variable, ncol = 3) #<- wrap plots by variable into 3 columns

# View updated plot.
scatter_norm
```



Adjust & polish: legends and text in scatterplot

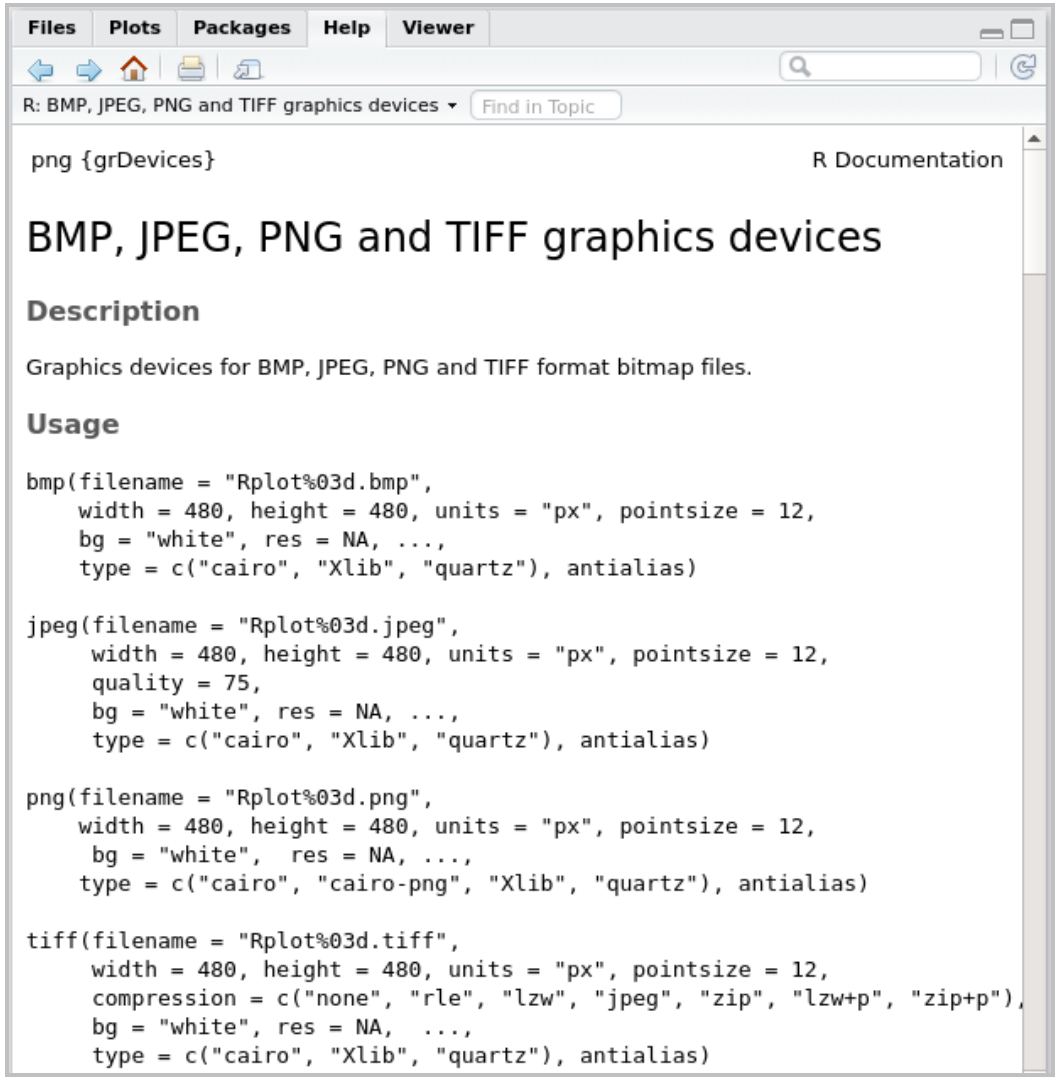
```
# Add finishing touches to the plot.
scatter_norm = scatter_norm +                          #<- previously saved plot
  guides(color = FALSE) +                              #<- remove legend for color mappings
  theme(strip.text.x = element_text(size = 14)) +      #<- increase text size in strips of facets
  labs(title = "CMP data: Yield vs. other variables",#<- add title and subtitle
       subtitle = "2D distribution of scaled data")

# View updated plot.
scatter_norm
```



Saving plots in R: starting with PNG

```
?png

png("Name_of_file.png", #<- name of file
    width = 400,        #<- width of image*
    height = 300,       #<- height of image*
    units = "px") #<- units for height & width*

call the plot object you want to export

dev.off() #<- closes R graphics device**
```

- `png` opens R graphics device
- `*` optional argument
- `**` required command that allows to clear R graphics device so that you can continue working with your plots
- `bmp`, `jpeg`, and other graphic export commands use similar command format



Saving plots in R: PNG exported

```
# Set working directory
# to where we want to save our plots.
setwd(plot_dir)

png("CMP_boxplots_norm.png",
    width = 1200,
    height = 600,
    units = "px")
boxplots_norm
dev.off()
```

```
quartz_off_screen
                2
```

```
png("CMP_density_norm.png",
    width = 1200,
    height = 600,
    units = "px")
density_norm
dev.off()

png("CMP_scatterplot_norm.png",
    width = 1200,
    height = 600,
    units = "px")
scatter_norm
dev.off()
```

- When the graphics device is cleared, you should get a similar message in your console

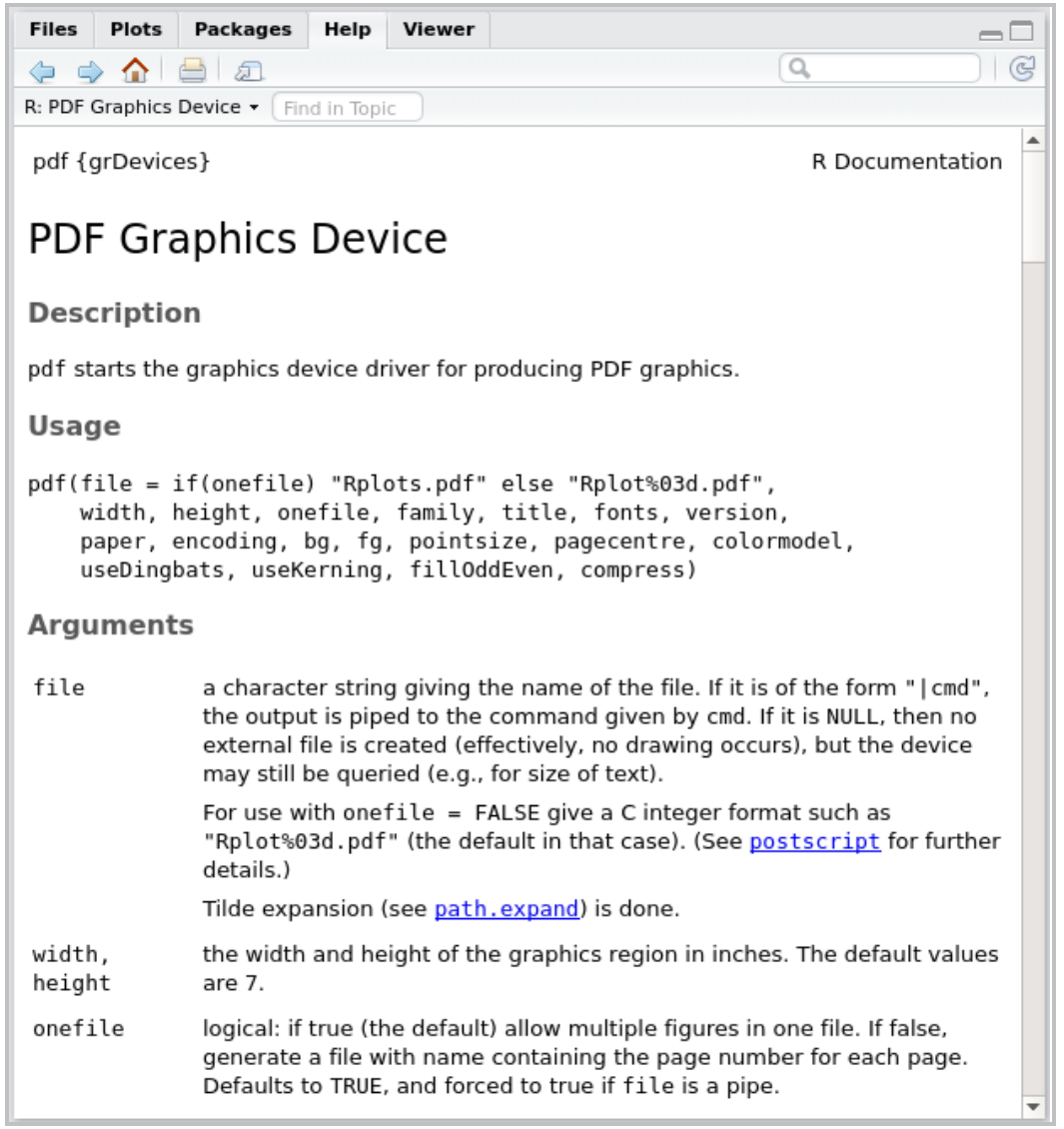
Saving plots in R: exporting to PDF

- There are a few advantages of saving your plots to a PDF document as opposed to an image:
  - PDF documents allow R's native **vector graphics** to shine - the quality of image will not be affected if you zoom in!
  - You can **save multiple plots into a single PDF document** with one command

Saving plots in R: PDF command syntax

```
?pdf

pdf("Name_of_file.pdf", #<- name of file
    width = 16,         #<- width in inches*
    height = 8)         #<- height in inches*

plot 1    #<- plot object or plotting function
plot 2    #<- plot object or plotting function
...       #<- arbitrary number of plots

dev.off() #<- clear graphics device**
```

- * optional argument
- ** required command



Files | Plots | Packages | Help | Viewer

R: PDF Graphics Device — Find in Topic

pdf {grDevices} R Documentation

PDF Graphics Device

Description

pdf starts the graphics device driver for producing PDF graphics.

Usage

```
pdf(file = if(onefile) "Rplots.pdf" else "Rplot%03d.pdf",
    width, height, onefile, family, title, fonts, version,
    paper, encoding, bg, fg, pointsize, pagecentre, colormodel,
    useDingbats, useKerning, fillOddEven, compress)
```

Arguments

file	a character string giving the name of the file. If it is of the form "|cmd", the output is piped to the command given by cmd. If it is NULL, then no external file is created (effectively, no drawing occurs), but the device may still be queried (e.g., for size of text). For use with onefile = FALSE give a C integer format such as "Rplot%03d.pdf" (the default in that case). (See postscript for further details.) Tilde expansion (see path.expand) is done.
width, height	the width and height of the graphics region in inches. The default values are 7.
onefile	logical: if true (the default) allow multiple figures in one file. If false, generate a file with name containing the page number for each page. Defaults to TRUE, and forced to true if file is a pipe.

Saving plots in R: PDF exported

```
# Set working directory to where you want to save plots.
setwd(plot_dir)

pdf("CMP_plots.pdf", #<- name of file
    width = 16,      #<- width in inches
    height = 8)      #<- height ...

boxplots_norm        #<- plot 1
density_norm         #<- plot 2
scatter_norm         #<- plot 3

dev.off()            #<- clear graphics device
```

```
quartz_off_screen
                2
```

Knowledge check 2

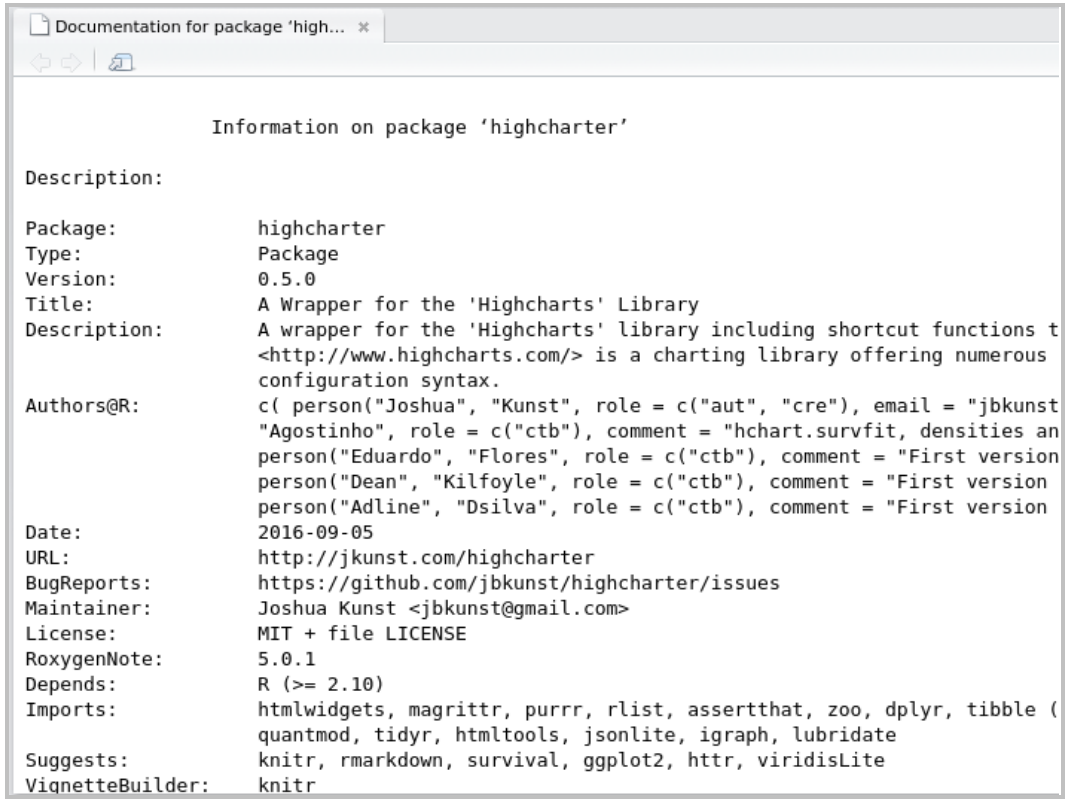
Exercise 2

Module completion checklist

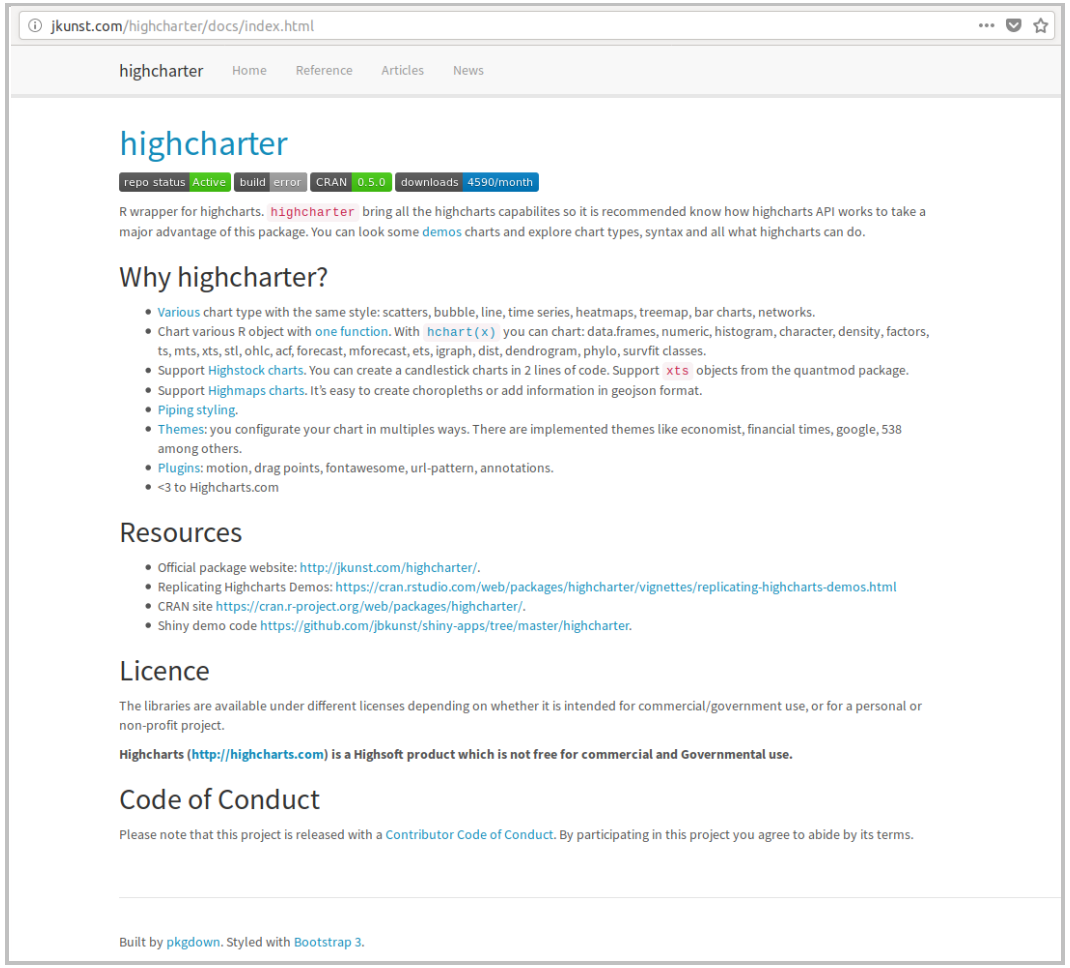
Objective	Complete
Transform data using dplyr, tidyr to prepare for compound visualizations with ggplot2	✔
Visualize transformed data using compound univariate visualizations	✔
Visualize transformed data using compound bivariate visualizations	✔
Explain the integration of the highcharter package	
Create basic interactive visualizations with transformed data	
Create interactive visualizations with transformed summary data	
Create complex interactive visualizations with multiple metrics and variables	

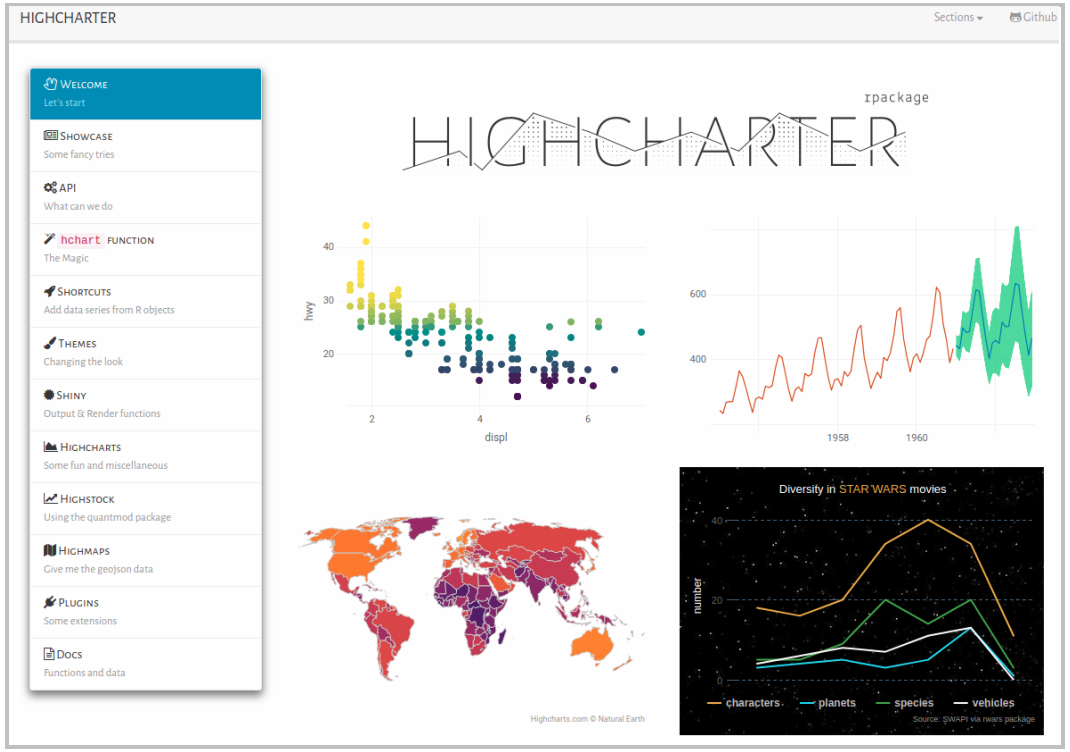
Interactive visualizations: highcharter

```r
# Install `highcharter` package.
install.packages("highcharter")

# Load the library.
library(highcharter)

# View documentation.
library(help = "highcharter")
```



Interactive visualizations with highcharter





Interactive visualizations with highcharter

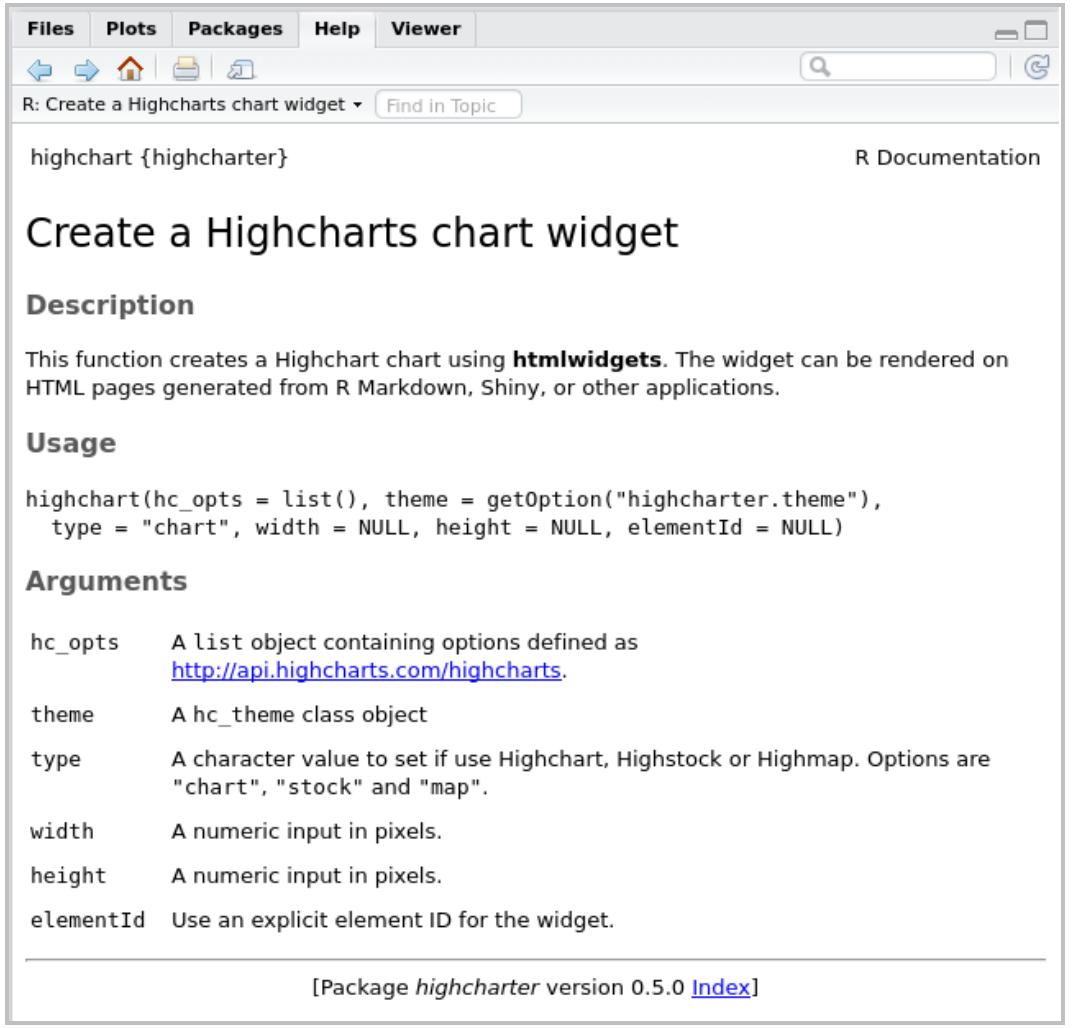
- `Highcharter` is an R wrapper that allows R users to tap into the beautiful and prolific world of one of the most comprehensive data visualization **Javascript based libraries**, `Highcharts`
- `Highcharts` is **free for individual research and non-profit purposes**, but there are cases and restrictions to its use and you may need to obtain a license if you decide to integrate it as part of your software or a company-wide product
- This library is similar to ggplot2 and other plotting libraries (including base R itself)
- Highcharter helps us build **complex, customized and meaningful visualizations with the layers** of graphical elements

```
> library(highcharter)
Highcharts (www.highcharts.com) is a Highsoft software product which is
not free for commercial and Governmental use
>
```

Main function highchart vs. hchart

```
?highchart
```

- In order to create a plot, you need to use the main plotting function (similar to `ggplot2`)
- In `highcharter`, it is `highchart`
- The function doesn't have any required arguments
- All of the graphic parameters and plotting options can be specified within the layers themselves



Files | Plots | Packages | Help | Viewer

R: Create a Highcharts chart widget | Find in Topic

highchart {highcharter} R Documentation

Create a Highcharts chart widget

Description

This function creates a Highchart chart using **htmlwidgets**. The widget can be rendered on HTML pages generated from R Markdown, Shiny, or other applications.

Usage

```
highchart(hc_opts = list(), theme = getOption("highcharter.theme"),
  type = "chart", width = NULL, height = NULL, elementId = NULL)
```

Arguments

hc_opts	A list object containing options defined as http://api.highcharts.com/highcharts.
theme	A hc_theme class object
type	A character value to set if use Highchart, Highstock or Highmap. Options are "chart", "stock" and "map".
width	A numeric input in pixels.
height	A numeric input in pixels.
elementId	Use an explicit element ID for the widget.

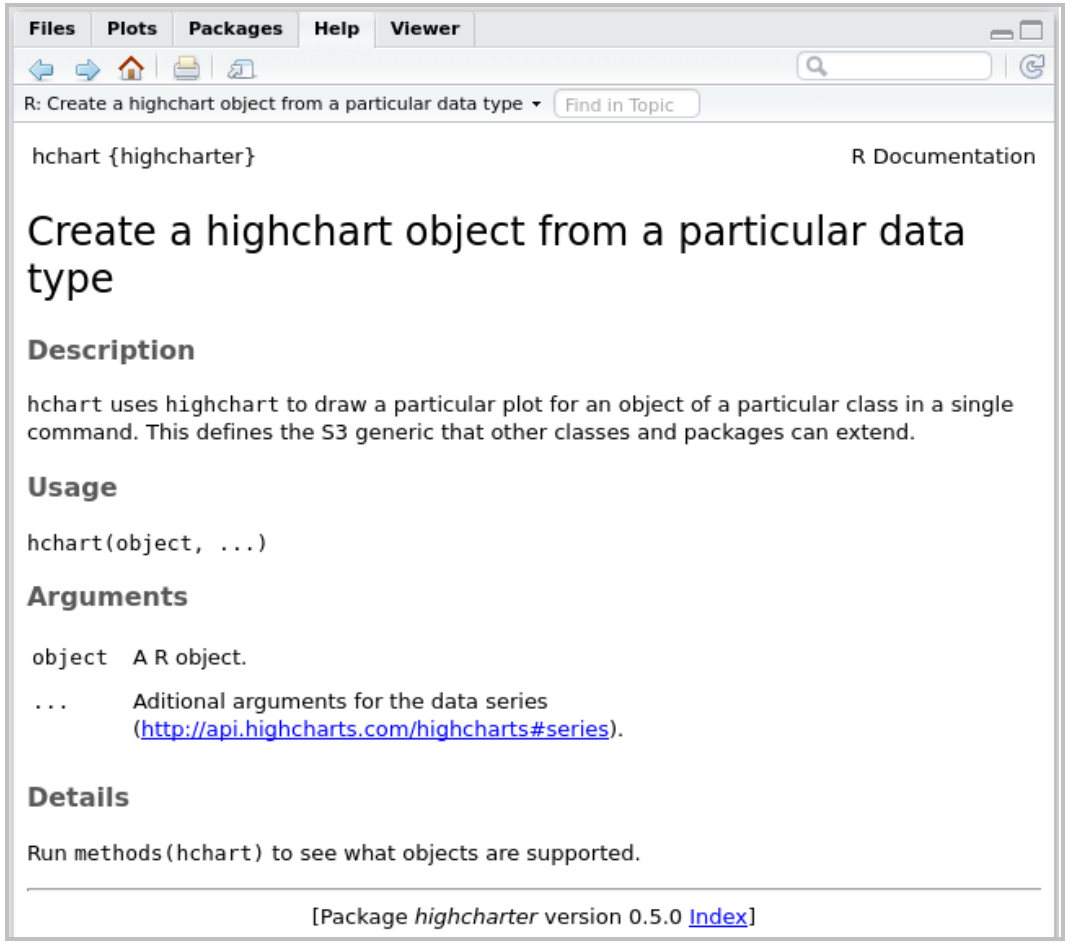
[Package *highcharter* version 0.5.0 Index]

Main function highchart vs. hchart

```
?hchart

hchart(Some_data,         #<- dataset to use
       "plot_type",       #<- plot type to use
       hcaes(x = variable1, #<- x-axis mapping
             y = variable2, #<- y-axis mapping
             group = variable3, #<- group by
             ...))
```

`hchart` is a shorthand version of the `highchart` function with some common settings and layers that takes a few key arguments to create a plot:

1. Data to use
2. Type of plot to create (i.e. `scatter`, `bar`, `column`, `line`, etc.)
3. `hcaes` (i.e. highcharts aesthetics) mapping of variables (works exactly the same way as with `ggplot2`!)

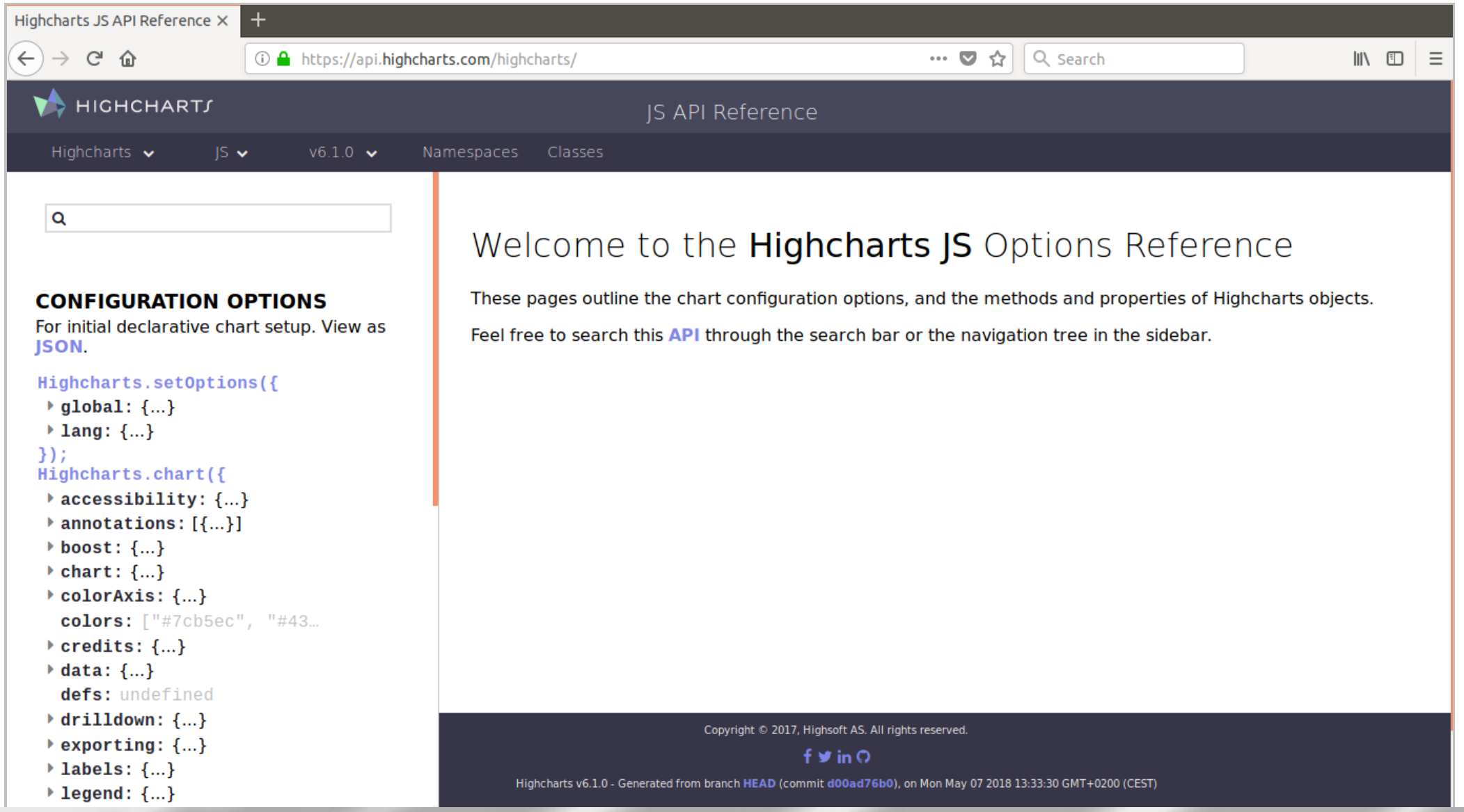


Layers in Highcharts: series

- Just like `ggplot2`, `highcharts` library has its own vocabulary
- Each new data or graphic layer in `highcharts` is called a `series`
- Each series can be a different `type`, here is a table of some widely used series types:

Highcharts series type	Plot type
scatter	scatterplot
line	line graph
boxplot	boxplot
column	bar plot
bar	horizontal bar plot
histogram	histogram
area	density

Highcharts: official documentation



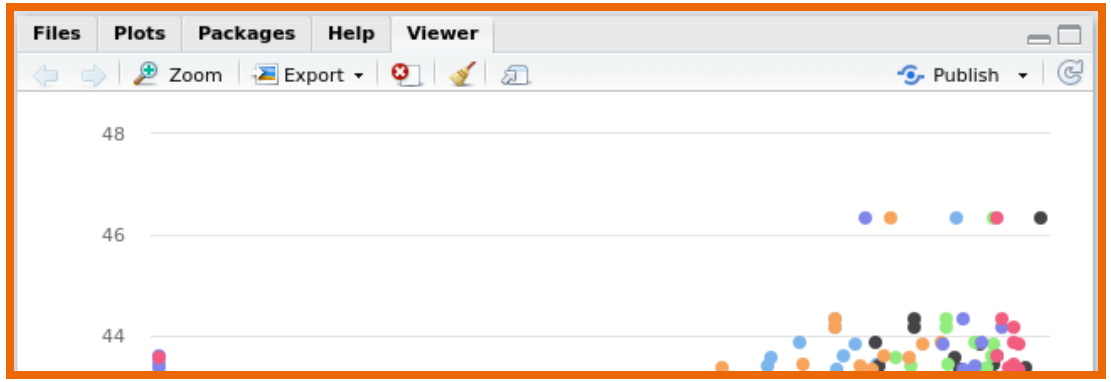
Module completion checklist

Objective	Complete
Transform data using dplyr, tidyr to prepare for compound visualizations with ggplot2	✔
Visualize transformed data using compound univariate visualizations	✔
Visualize transformed data using compound bivariate visualizations	✔
Explain the integration of the highcharter package	✔
Create basic interactive visualizations with transformed data	
Create interactive visualizations with transformed summary data	
Create complex interactive visualizations with multiple metrics and variables	

Highcharts generic function hchart: scatter

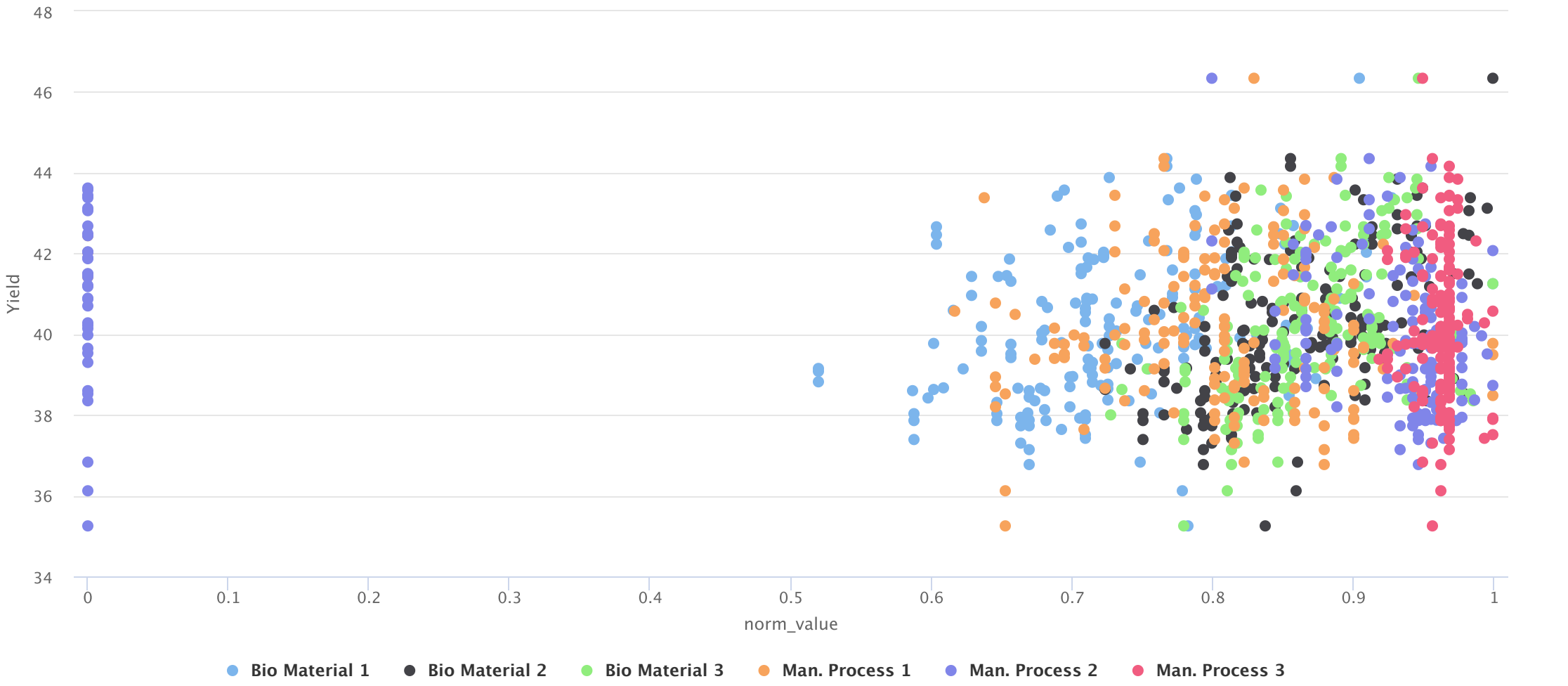
```
# Construct an interactive scatterplot.
scatter_interactive =                #<- name the plot
  hchart(CMP_subset_long2,           #<- set data
         "scatter",                  #<- make type "scatter"
          hcaes(x = norm_value,      #<- map x-axis
                y = Yield,           #<- map y-axis
                group = variable))   #<- group by
```

- In R, interactive charts appear in the `Viewer` pane right next to the `Help` tab

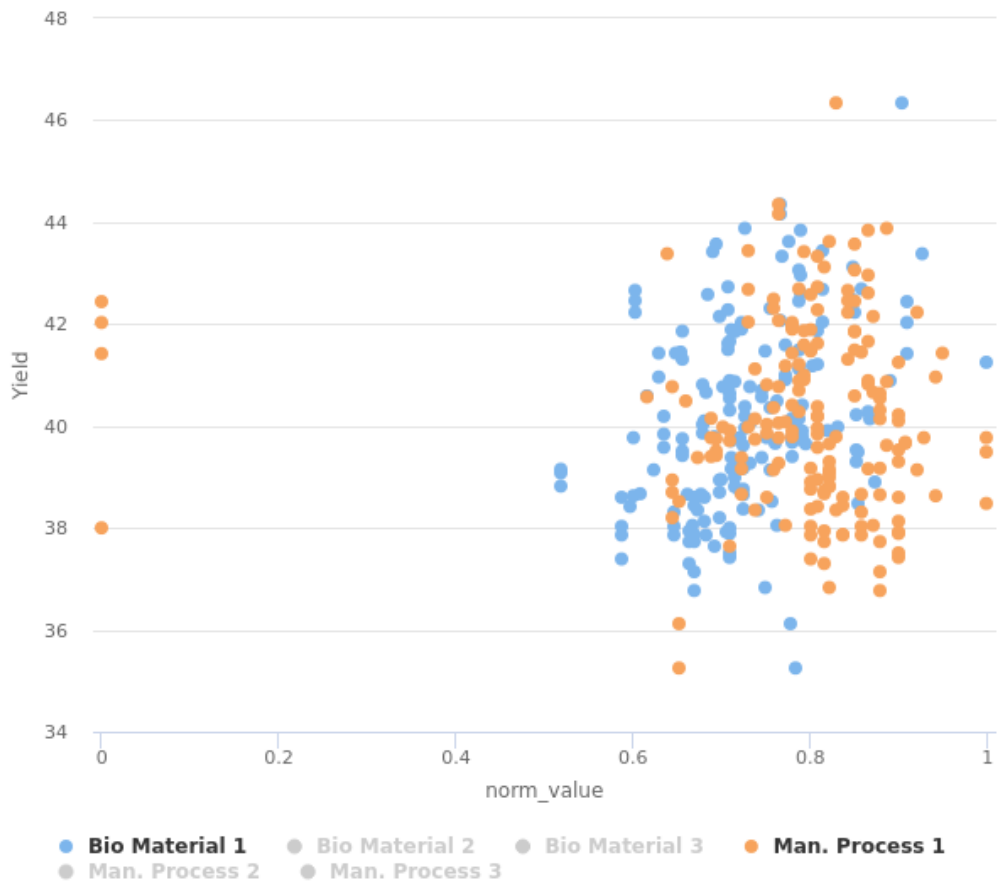


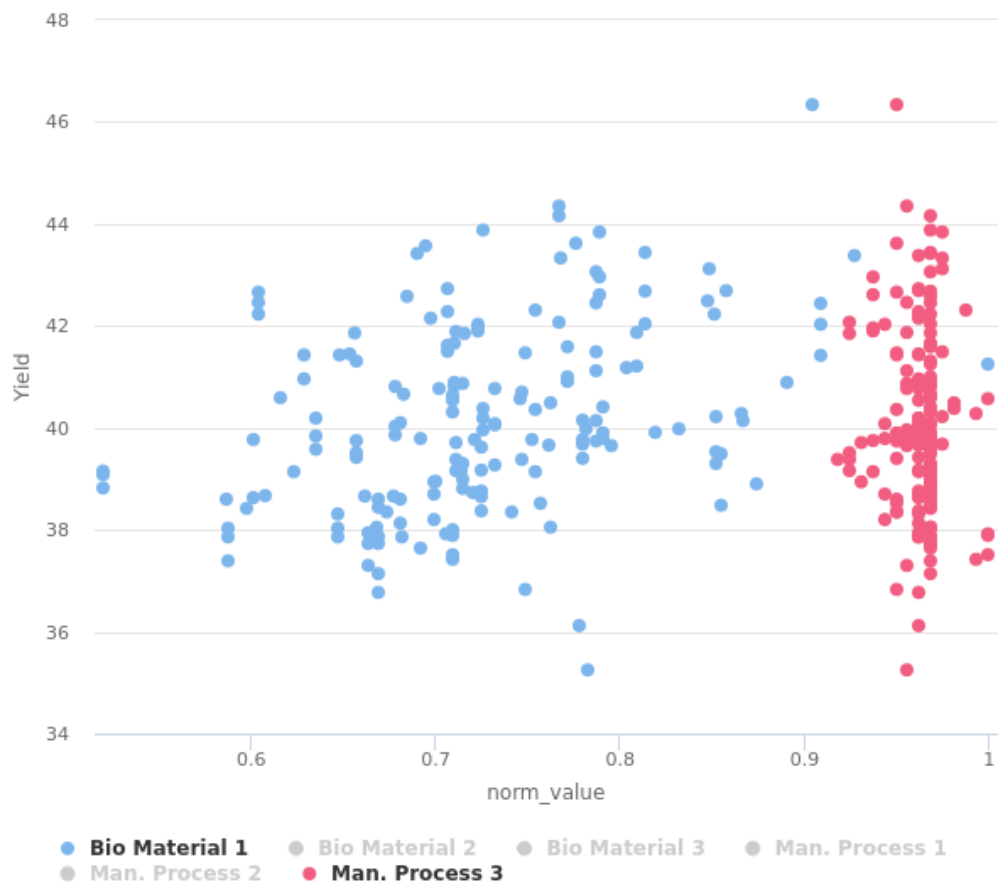
Highcharts generic function hchart: scatter

```
scatter_interactive
```



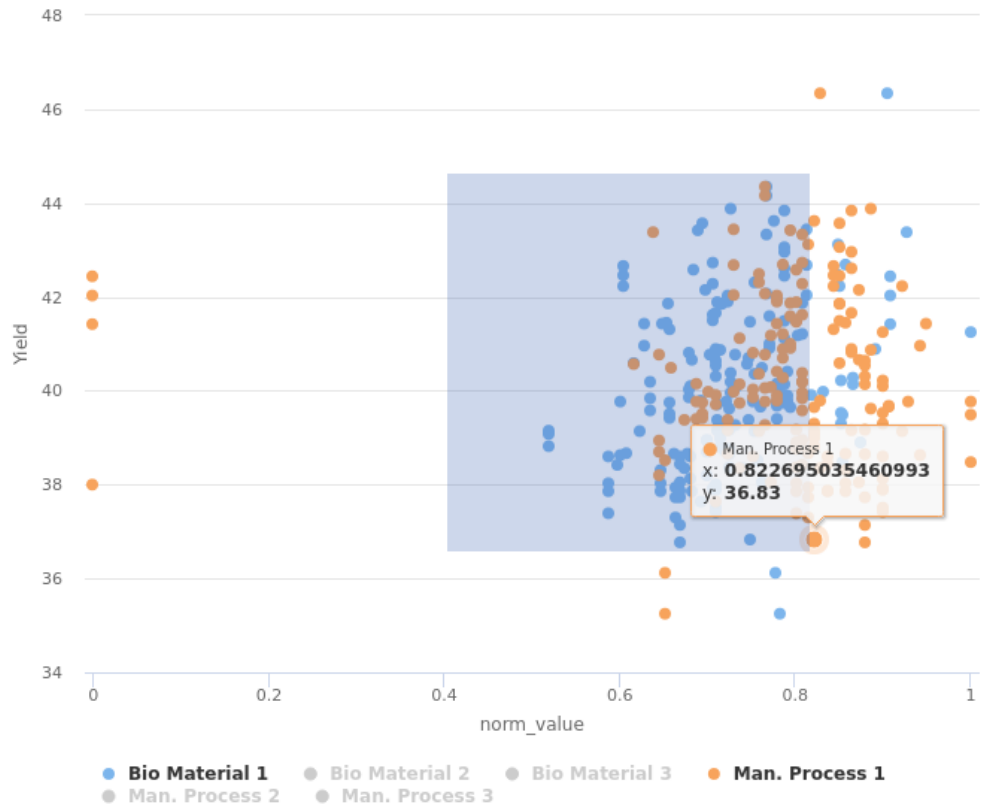
Highcharts generic function hchart: scatter





Highcharts generic function hchart: scatter

```
# Pipe chart options to original chart.
scatter_interactive = scatter_interactive %>%
  # Use chart options to specify zoom.
  hc_chart(zoomType = "xy")

scatter_interactive
```



Highcharts generic function hchart: scatter

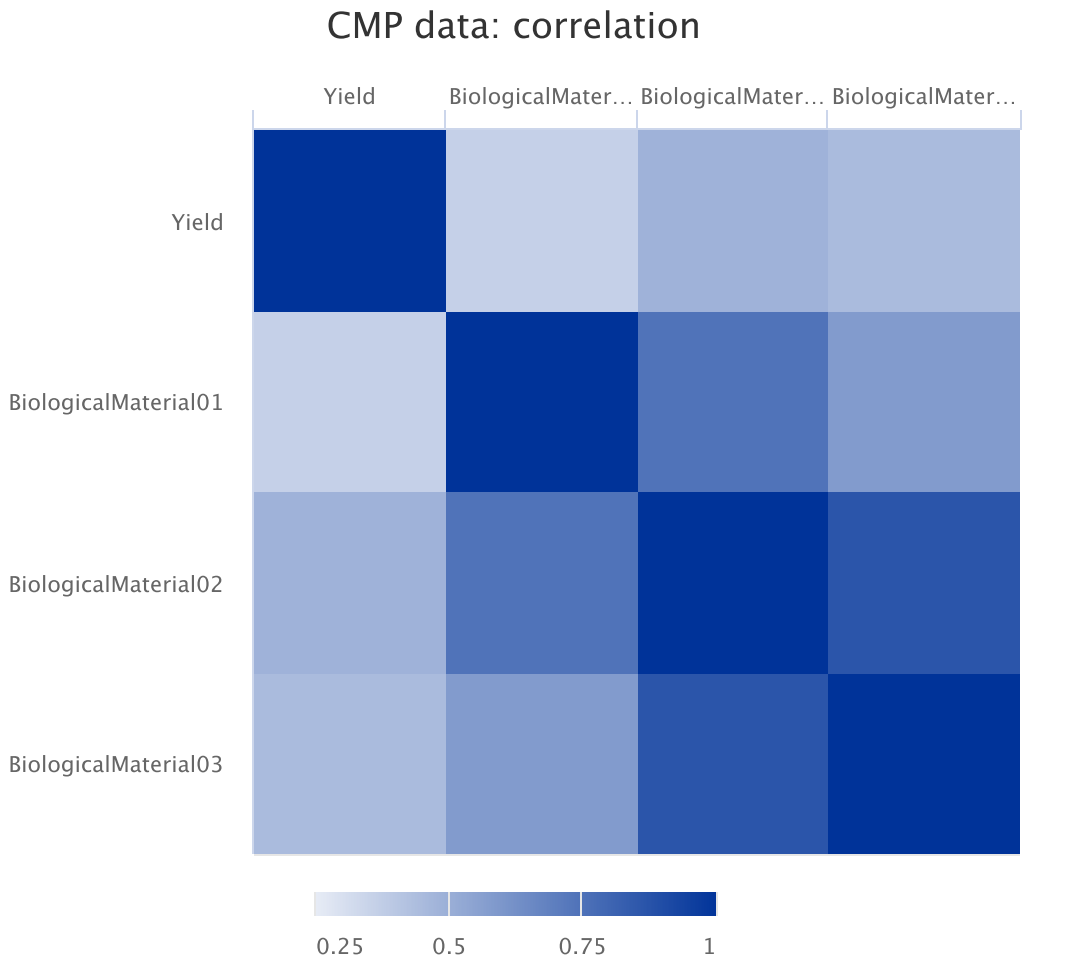
```
# Pipe chart options to original chart.
scatter_interactive = scatter_interactive %>%
 # Add title to your plot.
 hc_title(text = "CMP data: Yield vs. other variables")

scatter_interactive
```

Correlation matrix with hchart

```
# Compute a correlation matrix for the first
# 4 variables in our data.
cor_matrix = cor(CMP_subset[, 1:4])

# Construct a correlation plot by
# simply giving the plotting function
# a correlation matrix.
correlation_interactive = hchart(cor_matrix) %>%
 # Add title to your plot.
 hc_title(text = "CMP data: correlation")
```

- `hchart` is a universal plotting function
- It recognizes the type of data being given to it
- If you pass a correlation matrix, it will create a correlation plot
- No other arguments are necessary to create a basic plot!

```
correlation_interactive
```



Knowledge check 3

Exercise 3

Module completion checklist

Objective	Complete
Transform data using dplyr, tidyr to prepare for compound visualizations with ggplot2	✔
Visualize transformed data using compound univariate visualizations	✔
Visualize transformed data using compound bivariate visualizations	✔
Explain the integration of the highcharter package	✔
Create basic interactive visualizations with transformed data	✔
Create interactive visualizations with transformed summary data	
Create complex interactive visualizations with multiple metrics and variables	

Summary statistics visualization

- Let's create an interactive plot-summary of our data
- We need to compute the summary statistics like `min`, `max`, `mean`, etc.
- We need to transform and clean up the data to prepare for plotting
- Visualizing summary statistics are quick and easy way to understand our data
- We can analyze the important characteristics of the data and provide insights on data

Summary column plot with hchart

```
# Create data summary.
CMP_summary = summary(CMP_subset)

# Save it as a dataframe.
CMP_summary = as.data.frame(CMP_summary)

# Inspect the data.
head(CMP_summary)
```

```
  Var1      Var2            Freq
1          Yield Min.   :35.25
2          Yield 1st Qu.:38.75
3          Yield Median :39.97
4          Yield Mean   :40.18
5          Yield 3rd Qu.:41.48
6          Yield Max.   :46.34
```

```
# Remove an empty variable.
CMP_summary$Var1 = NULL

# Rename remaining columns.
colnames(CMP_summary) = c("Variable",
                          "Summary")
# Inspect updated data.
head(CMP_summary)
```

```
   Variable         Summary
1     Yield Min.   :35.25
2     Yield 1st Qu.:38.75
3     Yield Median :39.97
4     Yield Mean   :40.18
5     Yield 3rd Qu.:41.48
6     Yield Max.   :46.34
```

Summary column plot with hchart

```
# Separate `Summary` column into 2 columns.
CMP_summary = CMP_summary %>%                #<- set original data
  separate(Summary,                          #<- separate `Summary` variable
           into = c("Statistic", "Value"),   #<- into 2 columns: `Statistic`, `Value`
           sep = ":",                        #<- set separating character
           convert = TRUE)                   #<- where applicable convert data (to numeric)

# Inspect the first few entries in the data.
head(CMP_summary)
```

```
  Variable Statistic Value
1    Yield   Min.    35.25
2    Yield   1st Qu. 38.75
3    Yield   Median  39.97
4    Yield   Mean    40.18
5    Yield   3rd Qu. 41.48
6    Yield   Max.    46.34
```

```
# Inspect total number of rows in data including NAs.
nrow(CMP_summary)
```

```
[1] 49
```

Summary column plot with hchart

```
# Inspect `Value` column for `NAs`.
which(is.na(CMP_summary$Value) == TRUE)
```

```
[1]  7 14 21 28
```

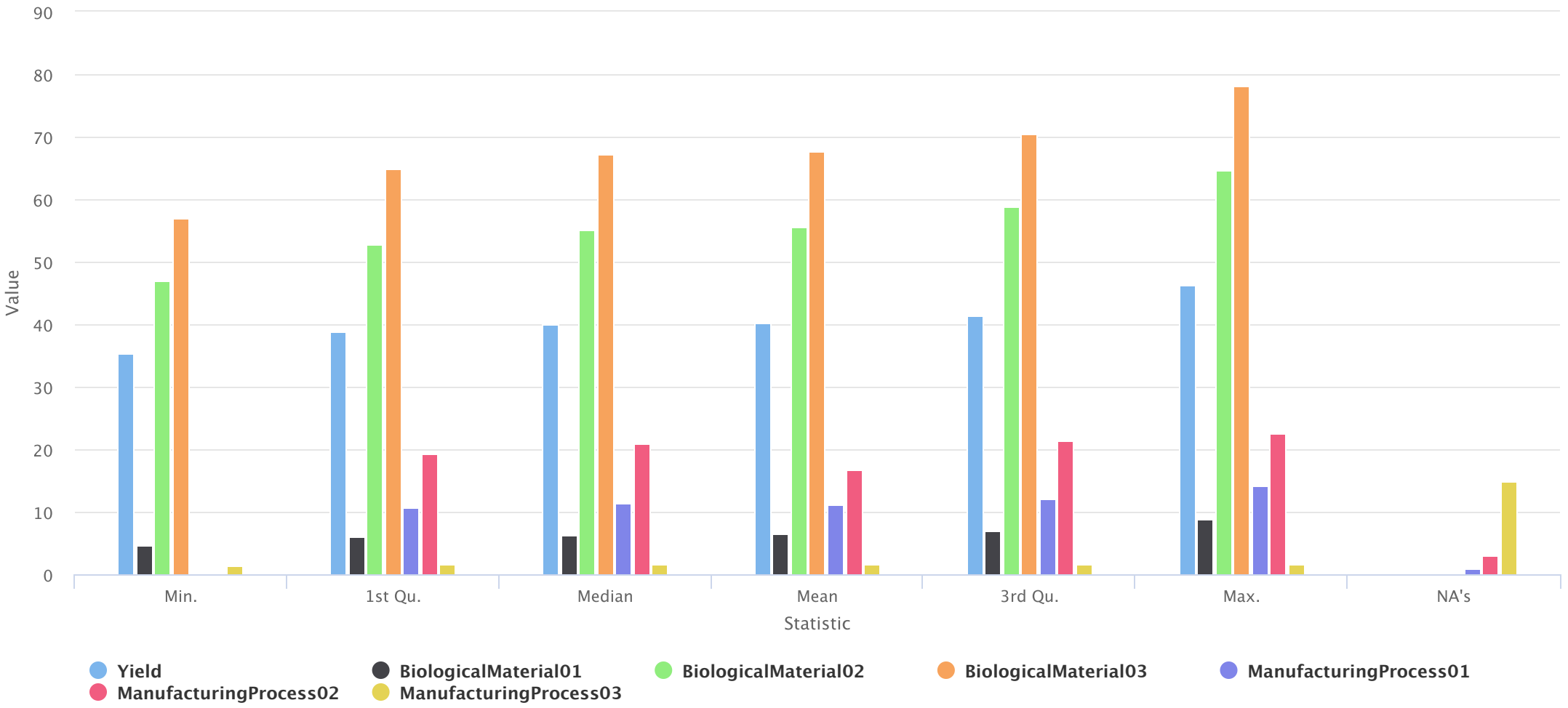
```
# Subset only rows where `Value` is not NAs.
CMP_summary = subset(CMP_summary, !is.na(Value))

# Now the number of rows should be 4 less.
nrow(CMP_summary)
```

```
[1] 45
```

```
# Construct the summary chart.
CMP_summary_interactive =
  hchart(CMP_summary,                #<- set data
         "column",                   #<- set type (`column` in highcharts)
         hcaes(x = Statistic,        #<- arrange `Statistics` across x-axis
               y = Value,            #<- map `Value` of each `Statistic` to y-axis
               group = Variable))    #<- group columns by `Variable`
```

Summary column plot with hchart

```
CMP_summary_interactive
```



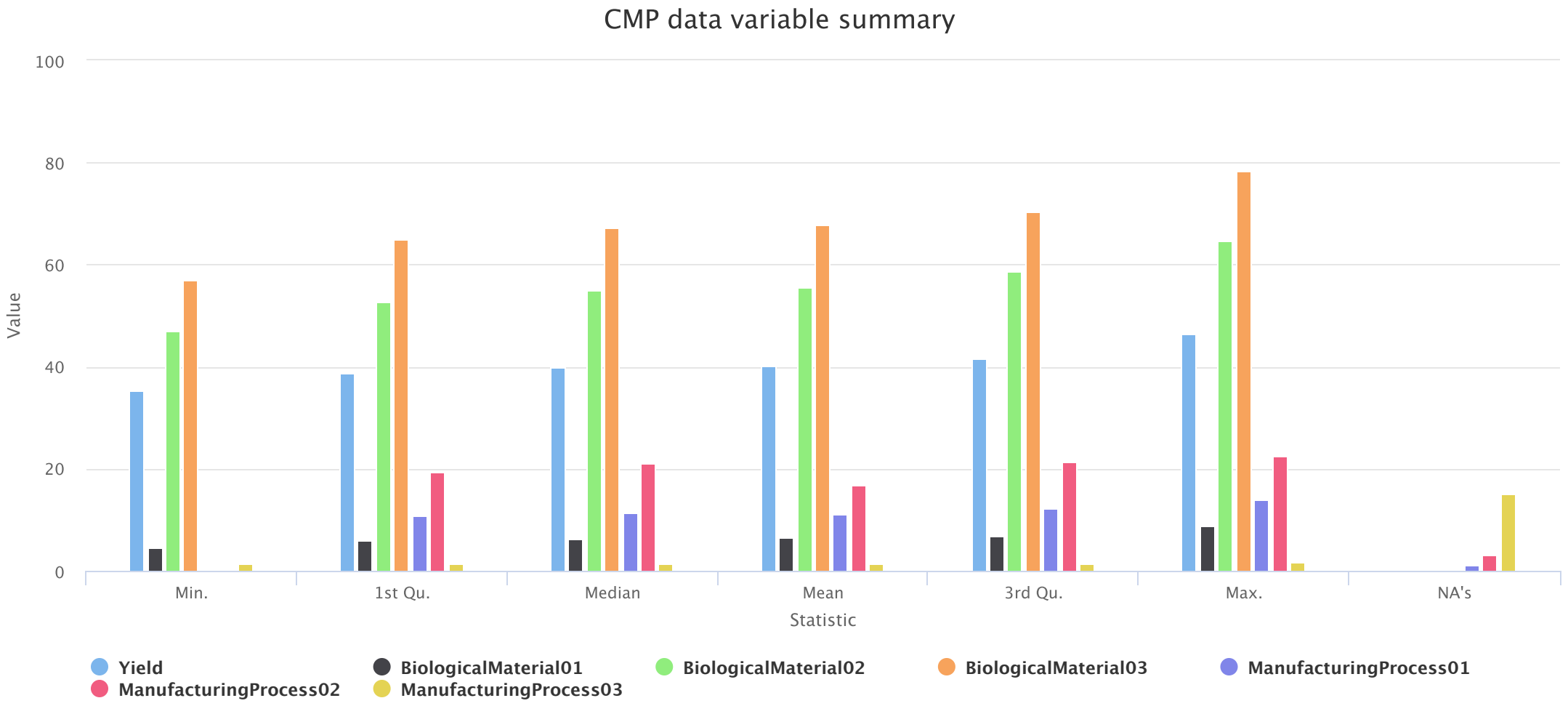
Summary column plot with hchart

- Since we are comparing the summary `statistics` of variables to each other, it would be more convenient to have the `tooltip` contain the information about the `group` rather than individual columns within the `group`
- We can control different `tooltip` options of the chart through `hc_tooltip` option
- One of the most used option is the `shared` option that allows us to have a shared tooltip between all members of the `group`

```
# Adjust tooltip options by piping `hc_tooltip` to base plot.
CMP_summary_interactive = CMP_summary_interactive %>%
  hc_tooltip(shared = TRUE)  %>%                #<- `shared` needs to be set to `TRUE`
  hc_title(text = "CMP data variable summary") #<- add title to your plot
```

Summary column plot with hchart

```
CMP_summary_interactive
```



Module completion checklist

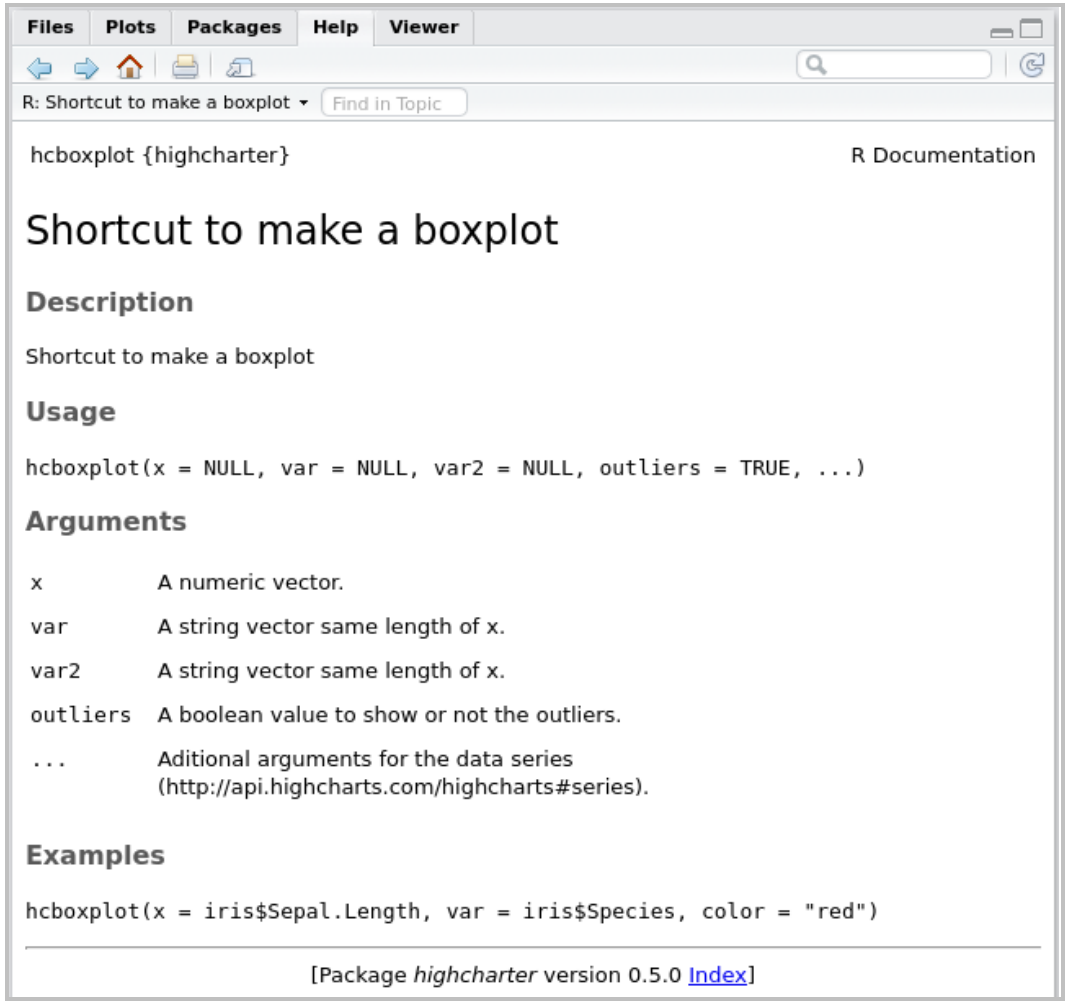
Objective	Complete
Transform data using dplyr, tidyr to prepare for compound visualizations with ggplot2	✔
Visualize transformed data using compound univariate visualizations	✔
Visualize transformed data using compound bivariate visualizations	✔
Explain the integration of the highcharter package	✔
Create basic interactive visualizations with transformed data	✔
Create interactive visualizations with transformed summary data	✔
Create complex interactive visualizations with multiple metrics and variables	

Highcharts boxplot: hcboxplot

```
?hcboxplot

hcboxplot(x = Numeric_data_vector,
          var = Categorical_data_vector,
          ...)
```

`hcboxplot` allows us to create an interactive boxplot

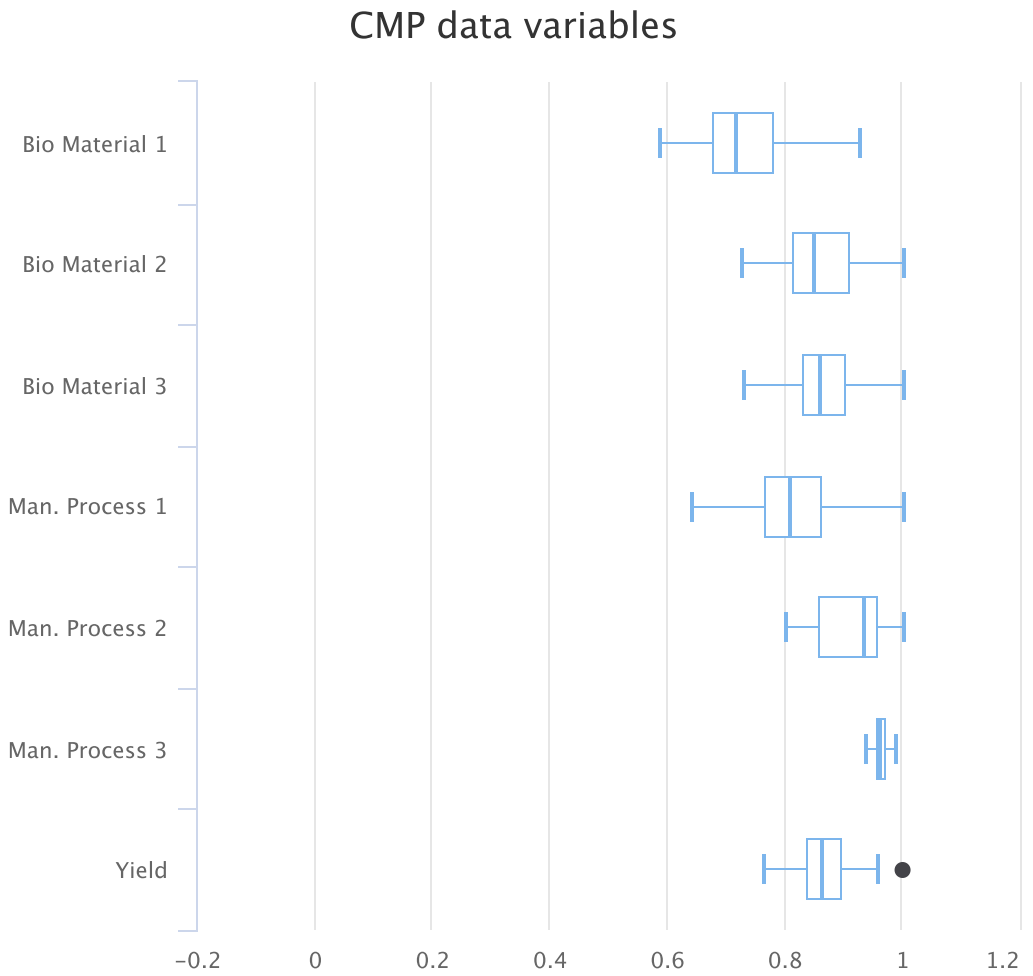
- `x` requires numeric data to be plotted along x-axis (`boxplot` in `Highcharts` is horizontal by default)
- `var` requires categorical data to be plotted along y-axis



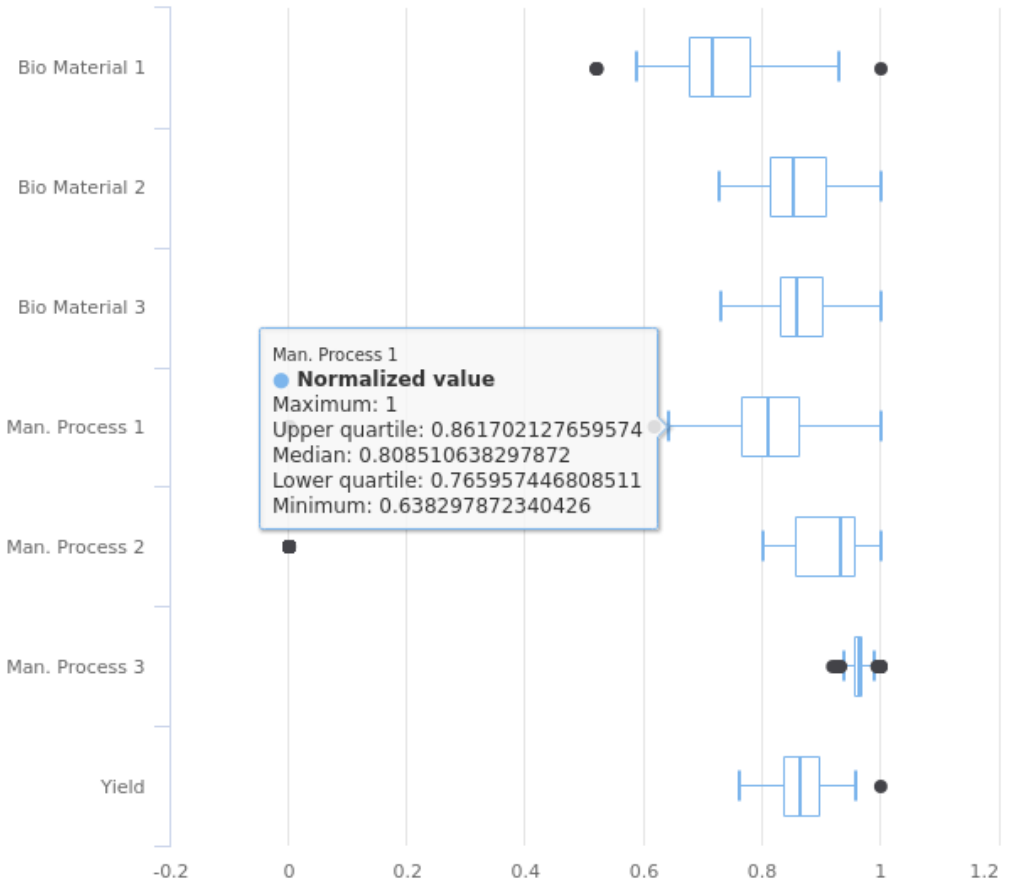
Highcharts boxplot: hcboxplot

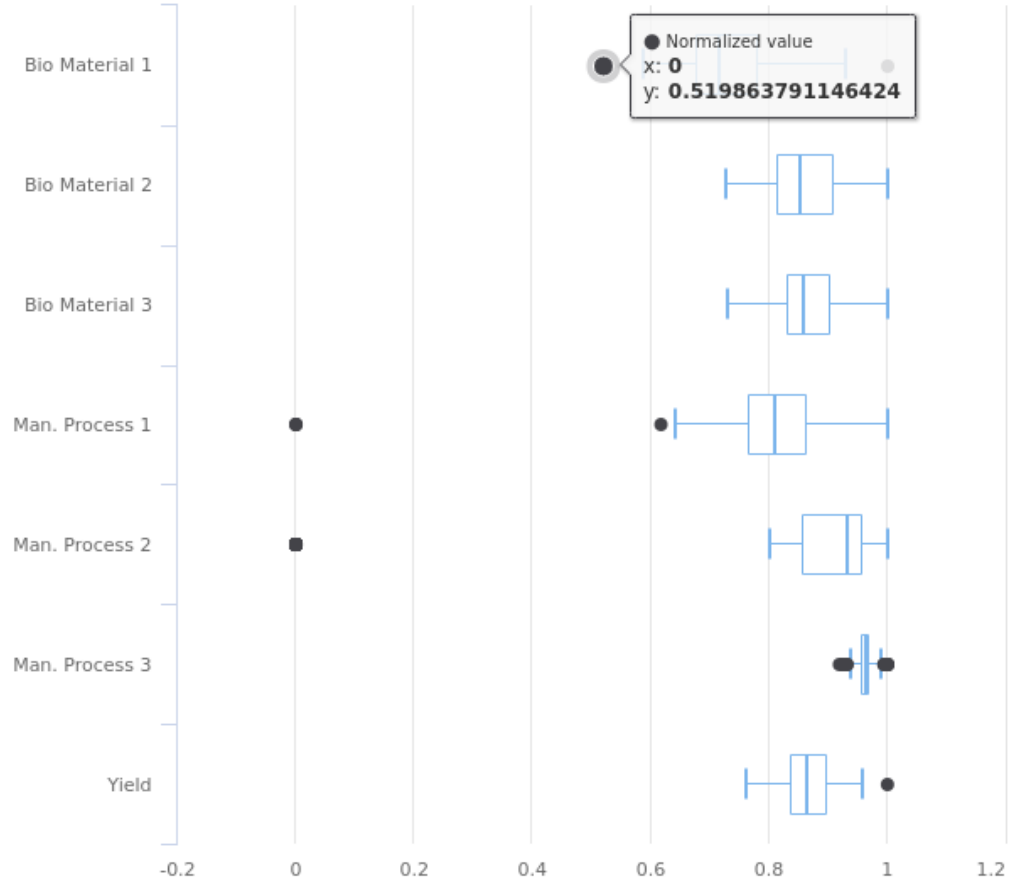
```
# Construct an interactive boxplot.
boxplot_interactive =
  hcboxplot(x = CMP_subset_long$norm_value,
            var = CMP_subset_long$variable,
            name = "Normalized value") %>%
  hc_title(text = "CMP data variables")
```

```
boxplot_interactive
```



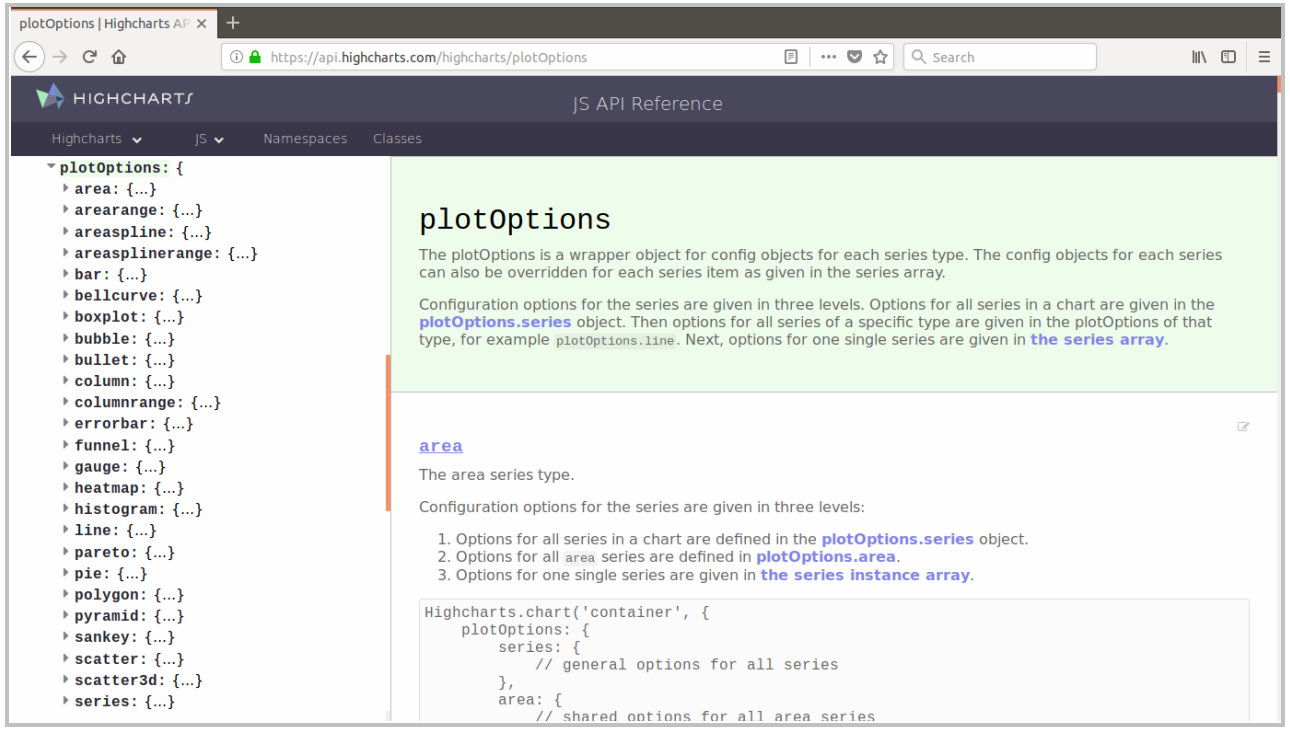
Highcharts boxplot: hcboxplot





Highcharts plotOptions

- The `hc_plotOptions` function controls individual layer / `series` options for various plot types
- You can pipe it to the original chart to **enhance your base plot**
- **Each series type** represented in the chart **can be given a unique set of options**
- **Every set of options is a list** of micro-level adjustments

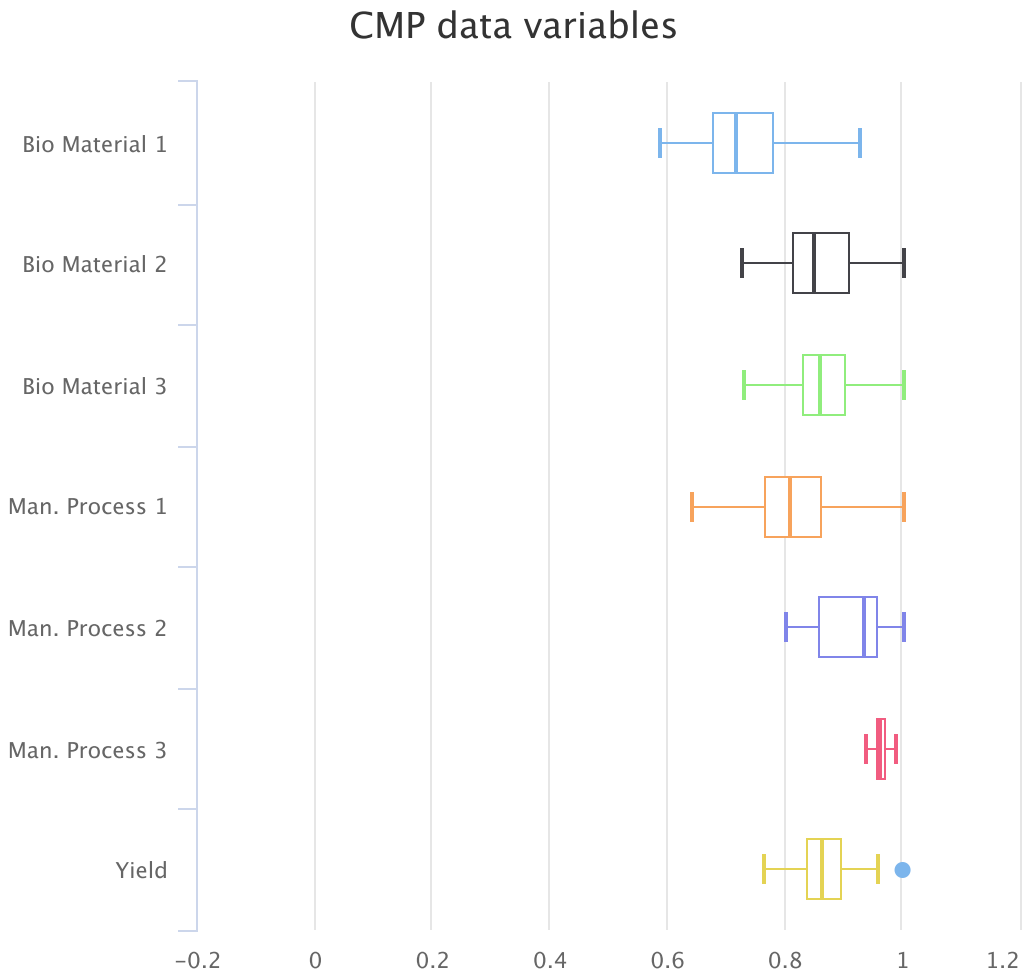


- The full list of plot options can be found in `Highcharts API` documentation ***https://api.highcharts.com/highcharts/plotOptions***

Highcharts boxplot: hcboxplot

```
# Enhance original boxplot with some options.
boxplot_interactive = boxplot_interactive %>%
  hc_plotOptions(       #<- plot options
    boxplot = list(     #<- for boxplot
      colorByPoint = TRUE)) #<- color each box
```

```
boxplot_interactive
```



Compound plots: highchart with layers

```
highchart() %>%                 #<- main plot
  hc_chart( ... )  %>%          #<- global chart options to apply to all layers
  hc_add_series( ... )  %>%     #<- plot an independent layer of data
  hc_add_series( ... )  %>%     #<- plot another independent layer of data
  ...
  hc_xAxis( ... ) %>%           #<- adjust x-axis options (if necessary)
  hc_yAxis( ... ) %>%           #<- adjust y-axis options (if necessary)
  hc_tooltip( ... ) %>%         #<- adjust tooltip (if necessary)
  hc_plotOptions( ... ) %>%     #<- adjust other plot options (if necessary)
  hc_legend( ... )  %>%         #<- adjust legend (if necessary)
  hc_title( ... )               #<- add/edit title (if necessary)
```

Compound plots: highchart with layers

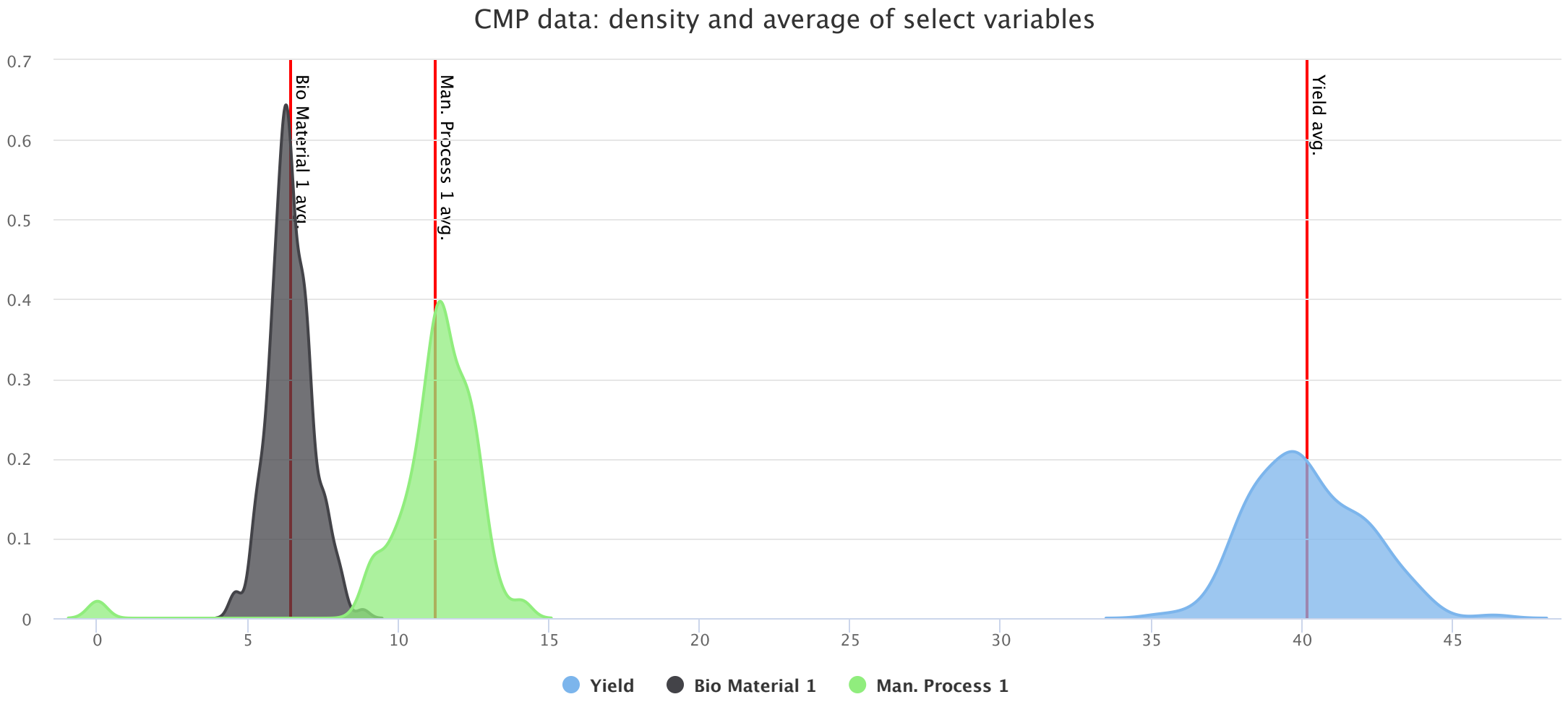
- Think of what we're doing as a big layered cake

Compound plots: density + lines example

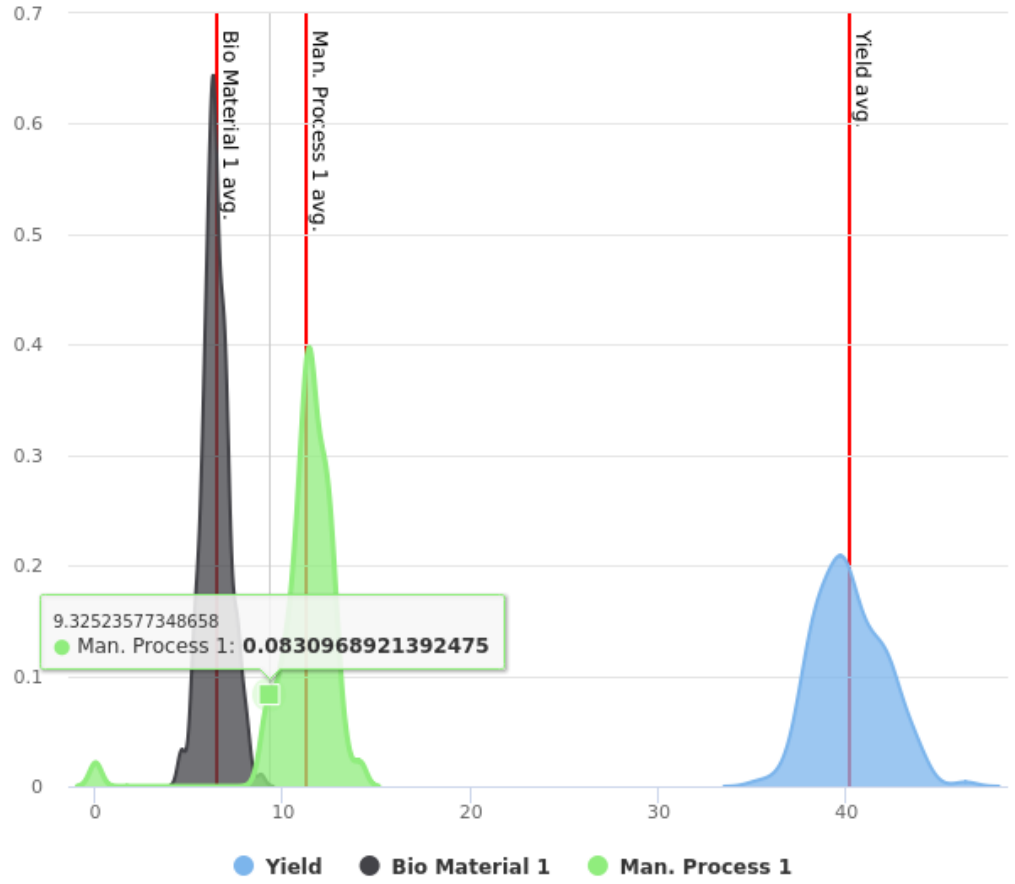
```
layered_density_interactive = highchart() %>%
  hc_chart(type = "area") %>%
  hc_add_series(data = density(CMP_subset$Yield),
                name = "Yield") %>%
  hc_add_series(data = density(CMP_subset$BiologicalMaterial01),
                name = "Bio Material 1") %>%
  hc_add_series(data = density(CMP_subset$ManufacturingProcess01, na.rm = TRUE),
                name = "Man. Process 1") %>%
  hc_xAxis(plotLines = list(
             list(label = list(text = "Yield avg."),
                  width = 2,
                  color = "red",
                  value = mean(CMP_subset$Yield)),
             list(label = list(text = "Bio Material 1 avg."),
                  width = 2,
                  color = "red",
                  value = mean(CMP_subset$BiologicalMaterial01)),
             list(label = list(text = "Man. Process 1 avg."),
                  width = 2,
                  color = "red",
                  value = mean(CMP_subset$ManufacturingProcess01, na.rm = TRUE)))) %>%
  hc_tooltip(crosshairs = TRUE) %>%
  hc_title(text = "CMP data: density and average of select variables")
```

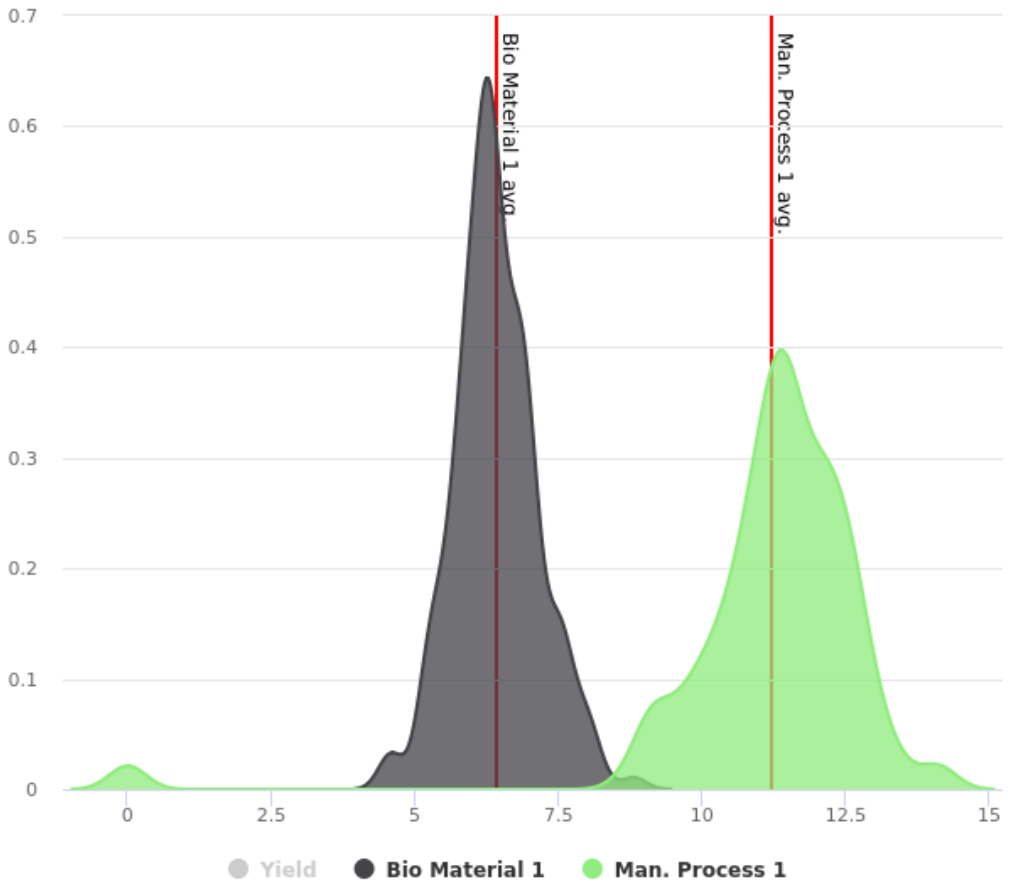
Compound plots: highchart with layers

```
layered_density_interactive
```



Layered plots





Knowledge check 4

Exercise 4

Module completion checklist

Objective	Complete
Transform data using dplyr, tidyr to prepare for compound visualizations with ggplot2	✔
Visualize transformed data using compound univariate visualizations	✔
Visualize transformed data using compound bivariate visualizations	✔
Explain the integration of the highcharter package	✔
Create basic interactive visualizations with transformed data	✔
Create interactive visualizations with transformed summary data	✔
Create complex interactive visualizations with multiple metrics and variables	✔

Workshop!

- Workshops are to be completed in the afternoons and can be reviewed in the mornings
- Make sure to annotate your code and comment so that it is easy for others to understand what you are doing
- This is an exploratory exercise to get you comfortable with the content we discussed today

Today, you will:

- transform data using `dplyr` and `tidyr`
- transform data and visualize it using `ggplot2`
- use the `highcharter` package to create interactive visualizations

This completes our module

Congratulations!