DATA SOCIETY®

Advanced visualization with R - Part 2

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

Warm up

Before we start the class, check out this list of exciting data visualizations created in 2020:
https://www.cyfe.com/blog/best-data-visualizations-of-2020/

Recap

- In the last module, we:
  - defined exploratory data analysis
  - created **basic static univariate, bivariate, and multivariate plots**
  - introduced the `ggplot2` package
- In this module, we will learn how to transform data and **build more complex static plots** using ggplot2

Module completion checklist

Objective	Complete
Create a scatterplot with ggplot2	
Transform data using tidyverse to prepare for compound visualizations	
Visualize a boxplot of transformed data with ggplot2	
Visualize a scatterplot of transformed data with ggplot2	
Save plots in R	

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 `skillsoft` folder

```
# Set `main_dir` to the location of your `skillsoft` folder (for Mac/Linux).
main_dir = "~/Desktop/skillsoft"
# Set `main_dir` to the location of your `skillsoft` folder (for Windows).
main_dir = "C:/Users/[username]/Desktop/skillsoft"

# 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")

# Set directory to data_dir.
setwd(data_dir)
```

Loading the data

- We will use the `CMP_subset` data from the last module to create more visualizations
- It is saved as `CMP_subset.csv` in your `data` directory

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

# Read the csv file into R environment.
CMP_subset = read.csv("CMP_subset.csv",
                      header = TRUE,
                      stringsAsFactors = FALSE)
```

Building a scatterplot with `geom_point`: set up

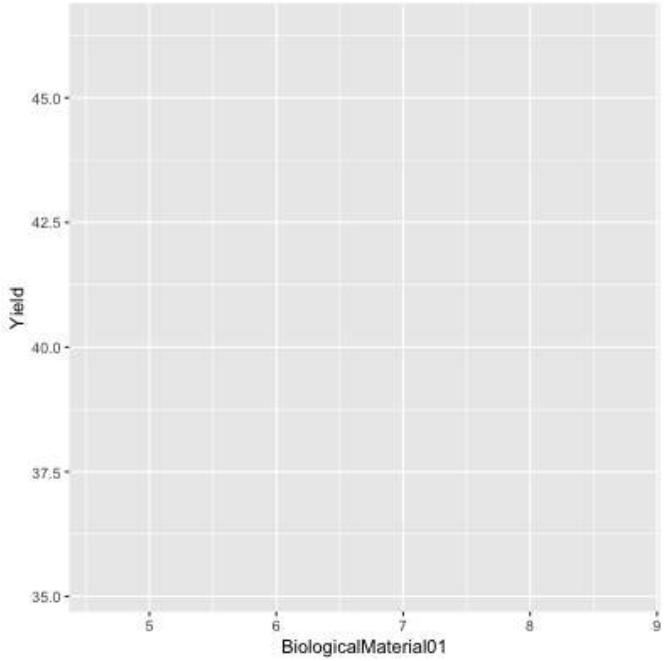
- Now that we have a basic understanding of building univariate plots, let's use `ggplot2` to create a scatterplot
- Since the scatterplot is a bivariate visualization, it requires a definition of 2 axes:
  - `x-axis` and
  - `y-axis`
- In `ggplot` terms, this means that we need to map 2 `aes` parameters (`x` and `y` respectively)
- Let's plot `BiologicalMaterial01` on `x-axis` and `Yield` on `y-axis`

Scatterplot with `geom_point`: set up (cont'd)

- First, we will create a base plot and map the `x` and `y` axes to the respective variables

```
# Load package
library(ggplot2)

ggp2 = ggplot(CMP_subset,                          #<-
specify dataframe
              aes(x = BiologicalMaterial01, #<-
map x-axis
                  y = Yield))                      #<-
map y-axis
```

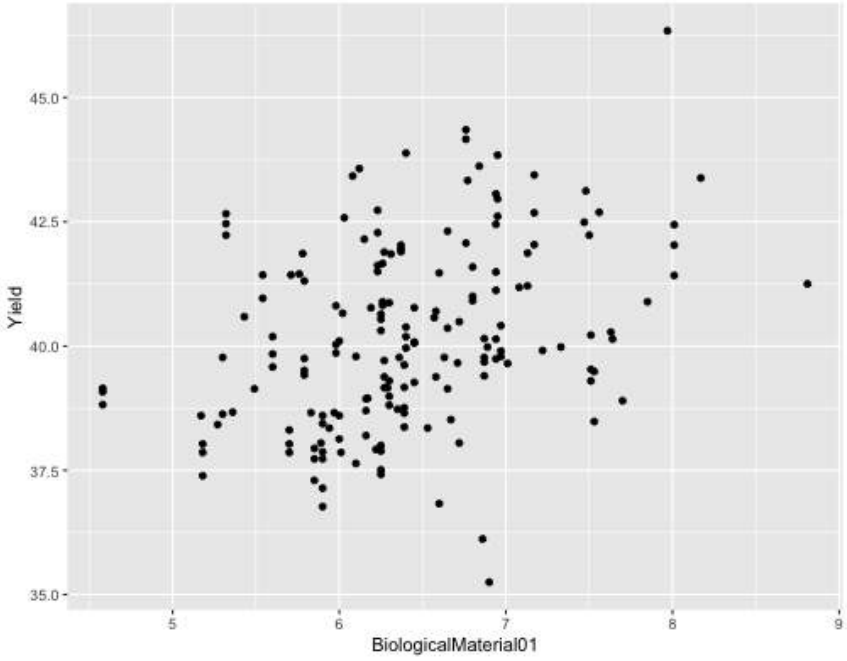
```
ggp2
```



Scatterplot with `geom_point`: set up (cont'd)

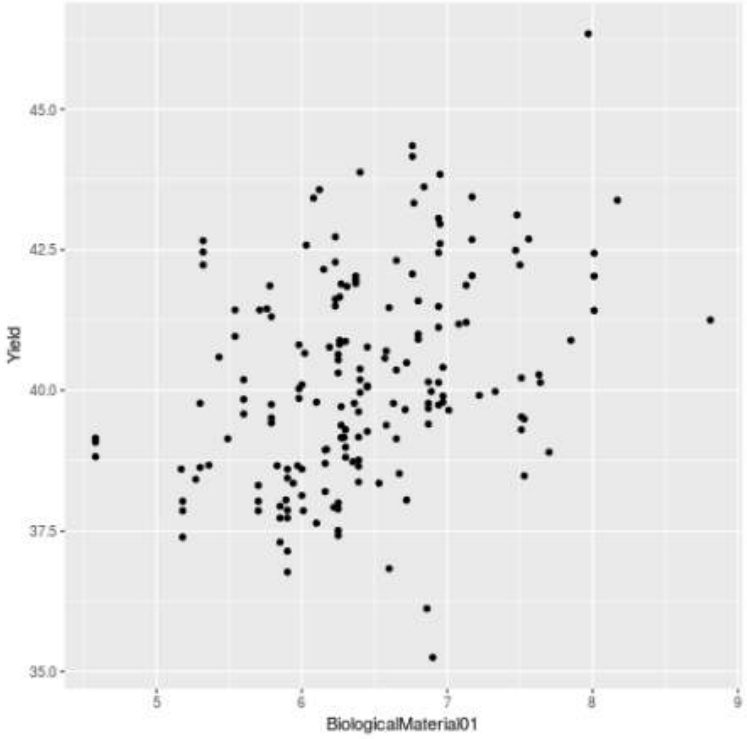
Now add the scatterplot layer using `geom_point()`

```
ggp2 = ggp2 + geom_point()
ggp2
```

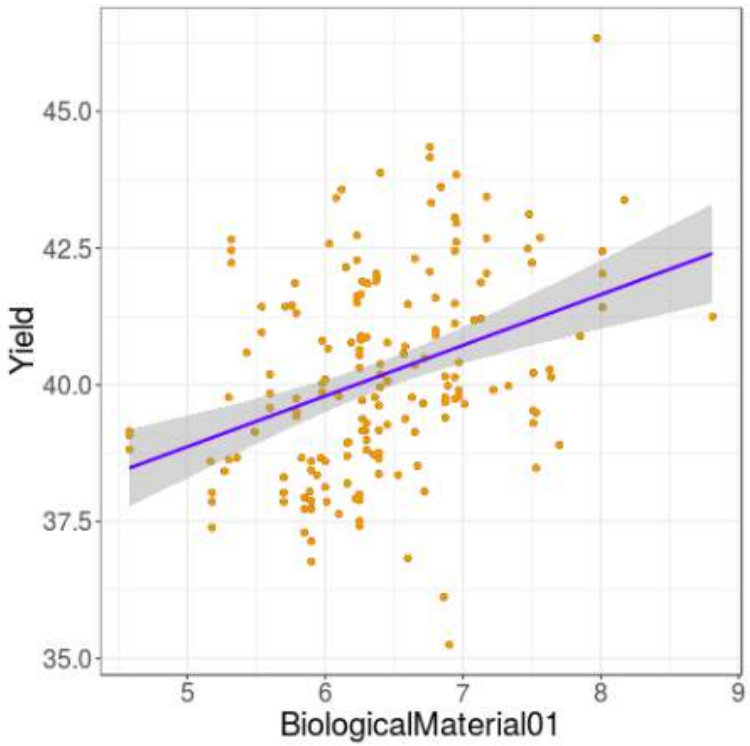


Building a scatterplot with fitted line

What we have



What we need

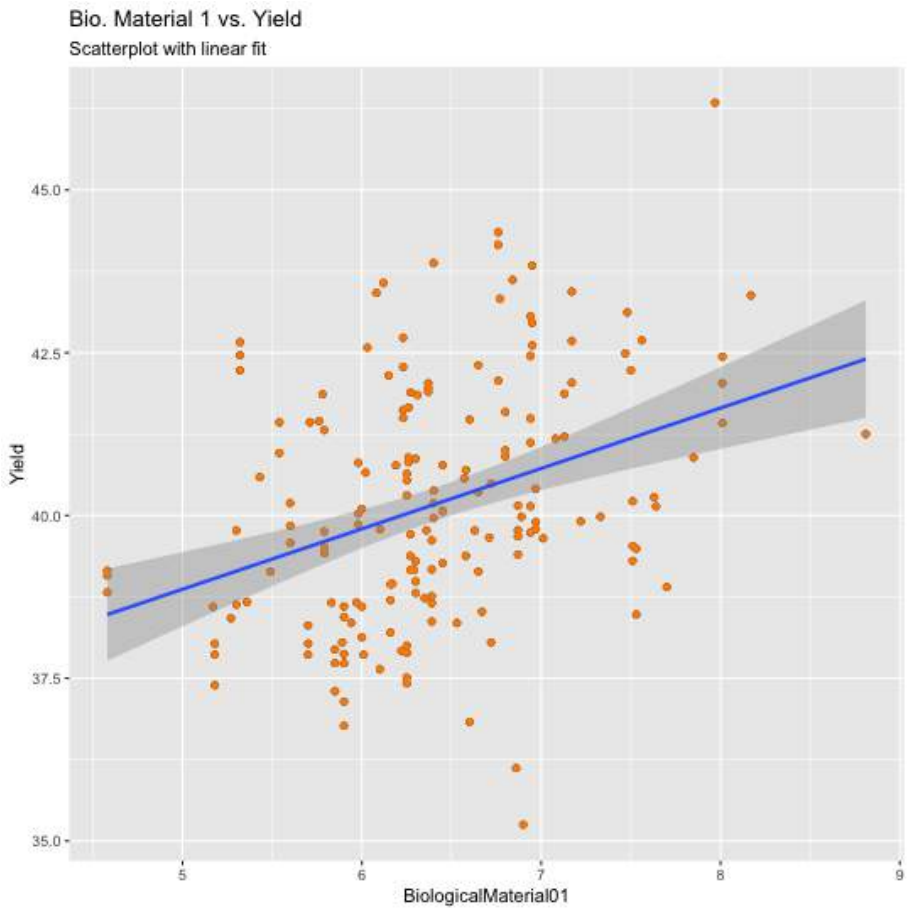


Scatterplot with `geom_point`: adjust

- Let's customize our plot as follows:
  - Change the color of data points to "darkorange"
  - Add a linear regression line using `geom_smooth()`
  - Add a title and a subtitle

```
ggp2 = ggp2 +

  # Adjust the color of the points.
  geom_point(color = "darkorange") +

  # Add linear regression line (`lm`).
  geom_smooth(method = lm) +

  # Add a title and a subtitle.
  labs(title = "Bio. Material 1 vs. Yield",
       subtitle = "Scatterplot with linear fit")
```

```
# View the plot.
ggp2
```

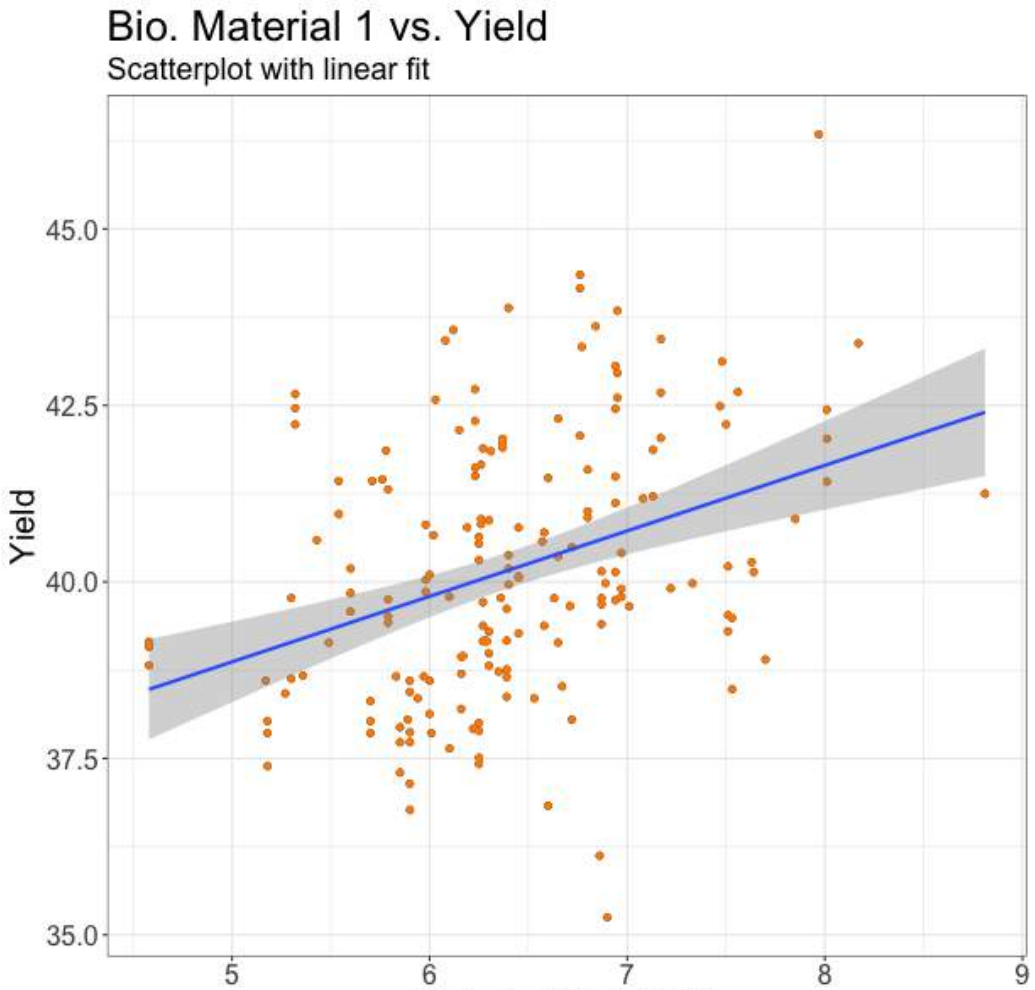


Scatterplot with `geom_point`: polish

Finally, let's polish our scatterplot by adding a theme

```
ggp2 = ggp2 +

  # Add black & white theme, adjust labels.
  theme_bw() +

  # Customize elements of the theme.
  theme(axis.title = element_text(size = 20),
        axis.text = element_text(size = 16),
        plot.title = element_text(size = 25),
        plot.subtitle = element_text(size = 18))
```

```
#View the plot
ggp2
```

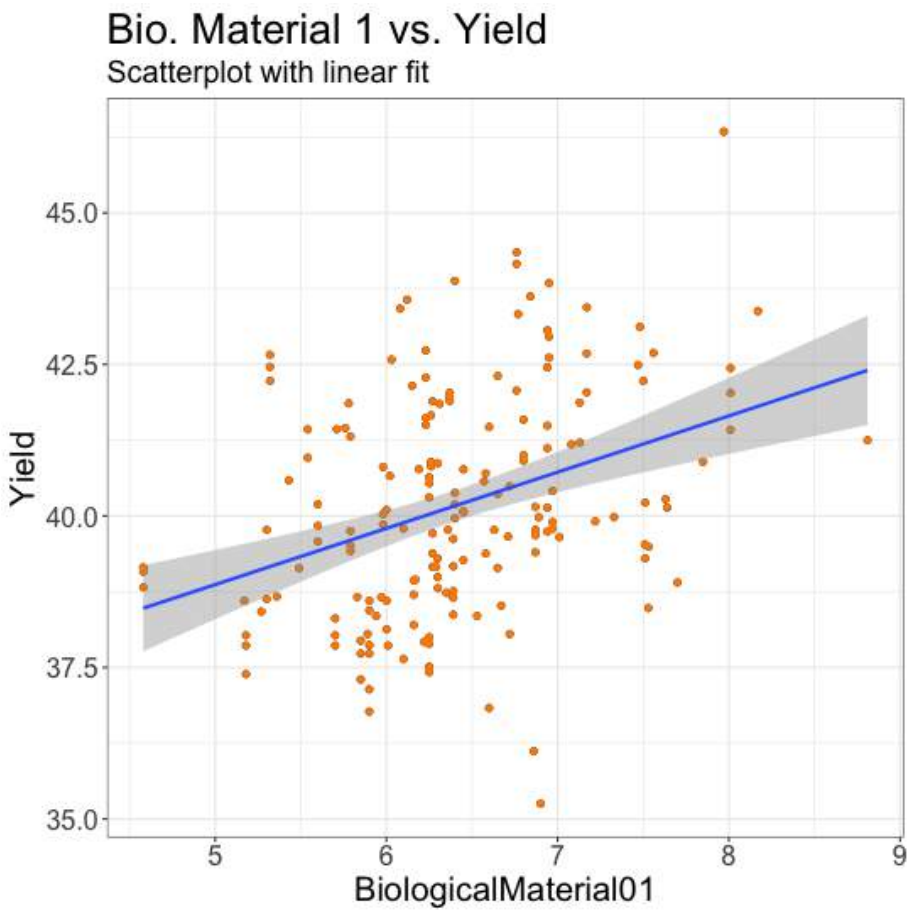


Saving a theme to a variable

- It would be a lot easier to **save the theme adjustments to a variable** and add a variable to our `ggplot`

```
my_ggtheme = theme_bw() +
  theme(axis.title = element_text(size = 20),
        axis.text = element_text(size = 16),
        plot.title = element_text(size = 25),
        plot.subtitle = element_text(size = 18))
```

```
# Add saved theme and re-save the plot.
ggp2 = ggp2 + my_ggtheme
ggp2
```



Module completion checklist

Objective	Complete
Create a scatterplot with ggplot2	✔
Transform data using tidyverse to prepare for compound visualizations	
Visualize a boxplot of transformed data with ggplot2	
Visualize a scatterplot of transformed data with ggplot2	
Save plots in R	

Data prep for `ggplot2`

As we learned in the last module, each chart in a `ggplot2` package is like a layered cake

- `ggplot2` is a part of the `tidyverse` collection of R packages
- `ggplot2` works best with **tidy data**, allowing 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

Set up: load tidyverse

- Let's load the `tidyverse` package that includes `ggplot2`

```
# Load tidyverse library
library(tidyverse)
```

- We will use the same `ggplot2` theme

```
# 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))
```

Set up: data conversion with `gather`

Let's convert our dataset from **wide** to **long** using `gather()` for easy visualization

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

The latest `tidyverse` version has `pivot_long()` which does the same work as `gather()` but it's still under development.

```
# 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
```

Set up: data cleaning with `mutate`

- Let's make a few edits to 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",
                          " "))
```

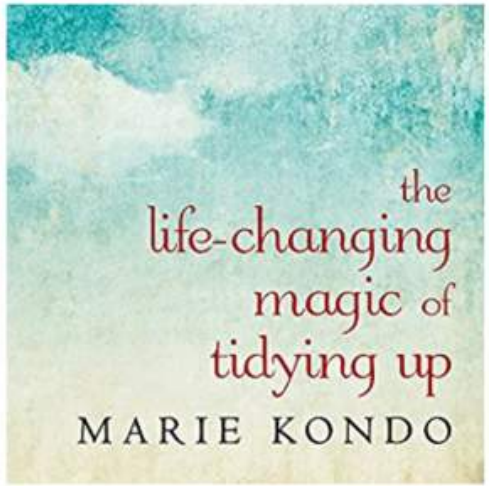
```
# Inspect few first
# entries in the data.
head(CMP_subset_long)
```

```
  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
```

Set up: tidy data ready for `ggplot`



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

NEXT STEP: Compare variables and their distributions to each other using `ggplot`.

Module completion checklist

Objective	Complete
Create a scatterplot with ggplot2	✔
Transform data using tidyverse to prepare for compound visualizations	✔
Visualize a boxplot of transformed data with ggplot2	
Visualize a scatterplot of transformed data with ggplot2	
Save plots in R	

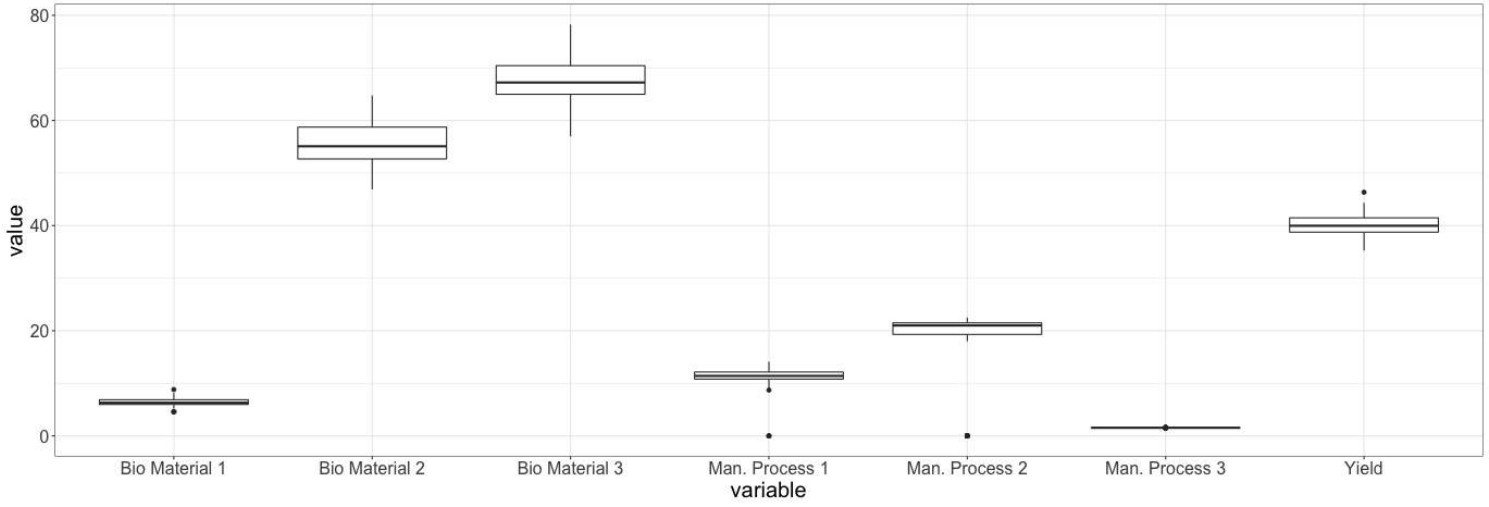
Using ggplot: assembling a layered cake

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

Set up & link data: make boxplots

Use the `geom_boxplot()` to add a boxplot layer to base plot

```
# A basic box plot 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 ✔

Set up: normalize data with group_by + mutate

- The variables are on different scales, which makes it harder for us to compare them

```
# Let's normalize the data and then create boxplots.
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
```

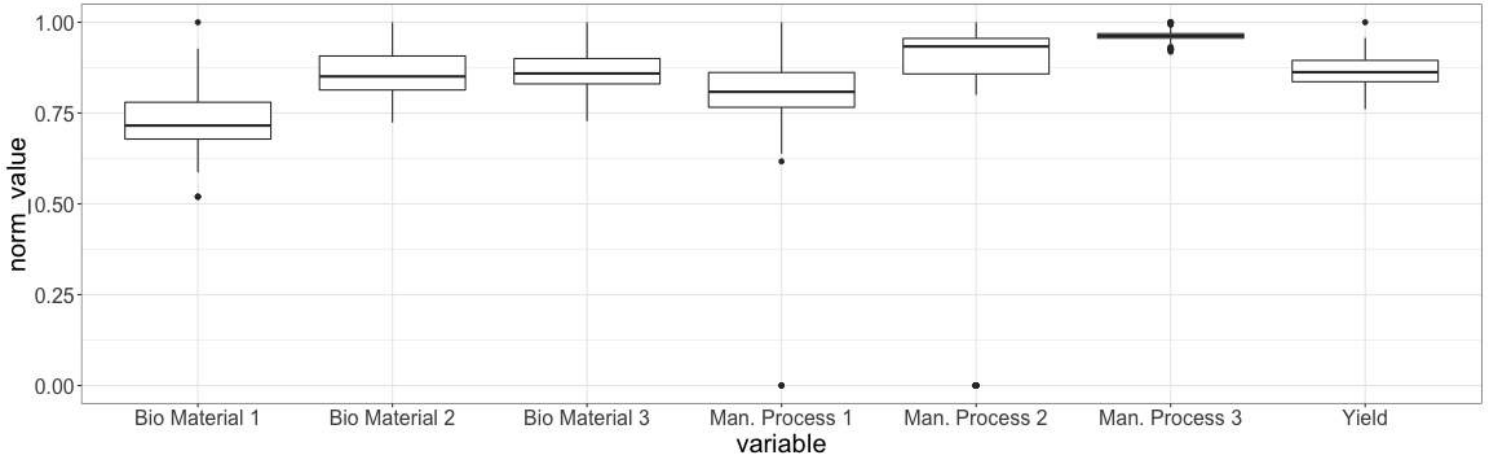
```
# A tibble: 1,232 x 3
# Groups:   variable [7]
   variable value norm_value
   <chr>    <dbl>      <dbl>
 1 Yield     38        0.820
 2 Yield     42.4      0.916
 3 Yield     42.0      0.907
 4 Yield     41.4      0.894
 5 Yield     42.5      0.917
 6 Yield     43.6      0.940
 7 Yield     43.1      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 ✔

Set up: boxplot with normalized data

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

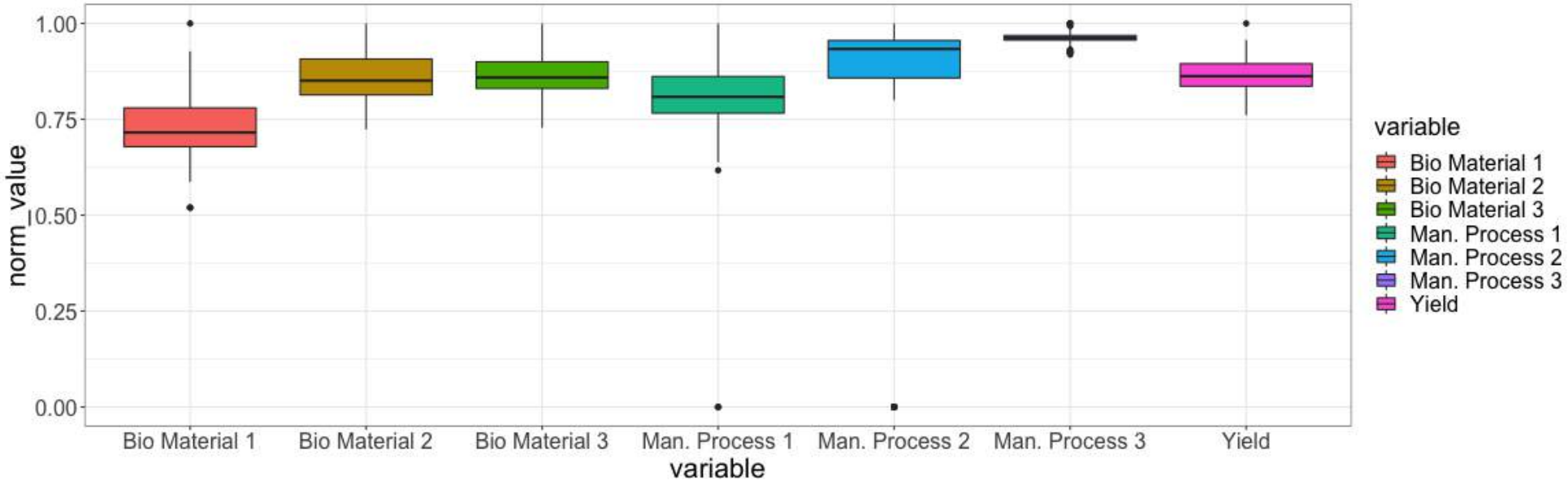
```
# Construct the base plot for normalized data.
base_norm_plot = ggplot(CMP_subset_long,          #<- set data
                        aes(x = variable,         #<- map `variable`
                            y = norm_value)) +    #<- map `norm_value`
                  geom_boxplot() +            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

# View
base_norm_plot
```



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

Adjust: normalized boxplot aesthetics

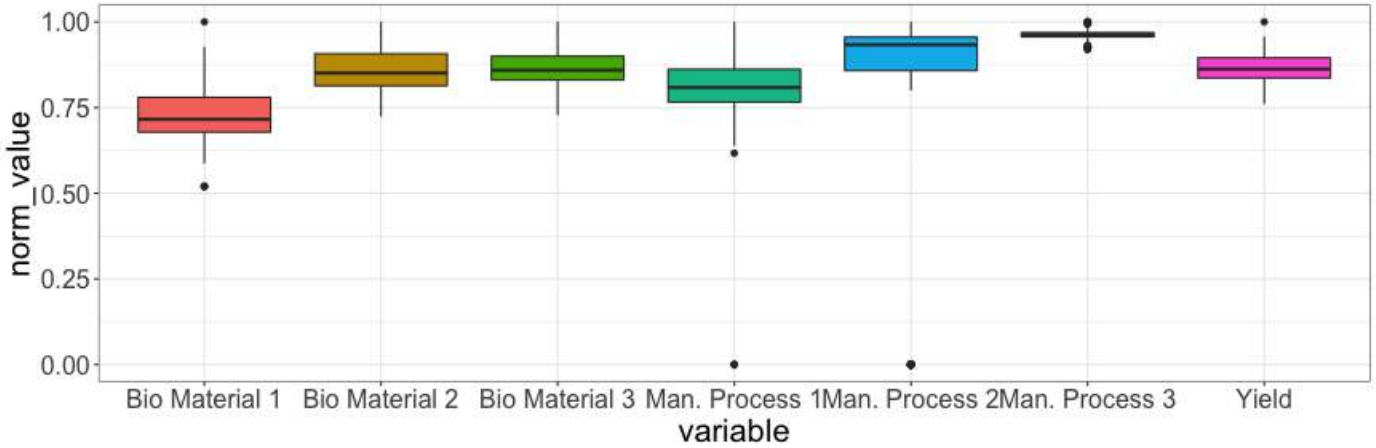
```
# Let's add a fill color to the plot.
boxplots_norm = ggplot(CMP_subset_long,          #<- set data
                       aes(x = variable,          #<- map `variable`
                           y = norm_value,        #<- map `norm_value`
                           fill = variable)) +    #<- set fill
                 geom_boxplot()  +                #<- boxplot layer
                 my_ggtheme                       #<- add theme

# View boxplot
boxplots_norm
```



Adjust: normalized boxplot legends

- To remove the legend for `fill` color, use the function `guides()` and pass `fill = FALSE` to it
- You can read more about modifying legends with `guides()` ***here***

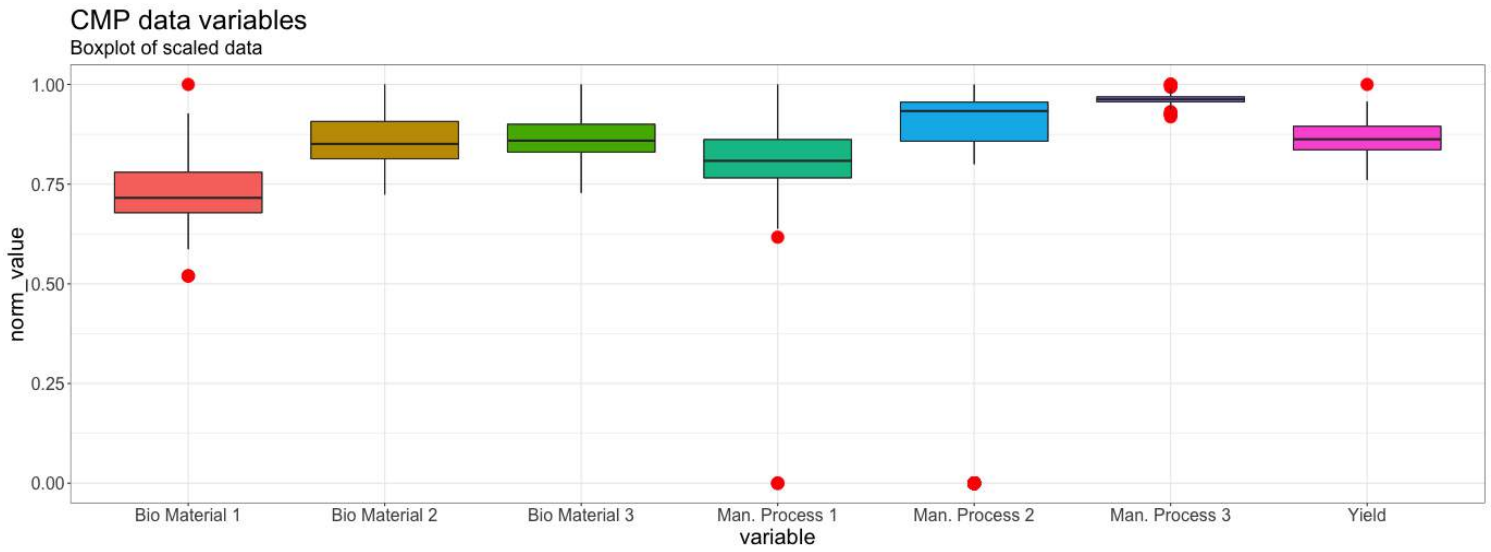
```
# Make color of fill based on variable.
boxplots_norm = boxplots_norm +      #<- set base plot
                guides(fill = FALSE) #<- remove redundant legend

# View updated plot.
boxplots_norm
```



- Followed by 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
```



- And polish until it looks just right ✔

Knowledge check 1

Exercise 1

Module completion checklist

Objective	Complete
Create a scatterplot with ggplot2	✔
Transform data using tidyverse to prepare for compound visualizations	✔
Visualize a boxplot of transformed data with ggplot2	✔
Visualize a scatterplot of transformed data with ggplot2	
Save plots in R	

Scatterplots work best with long data

Now let's build another **scatterplot** with ggplot2

- We are going to use data from materials and processes to predict `Yield` in our regression algorithms
- 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

- Sometimes differently structured ingredients are needed ✔

Set up: 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

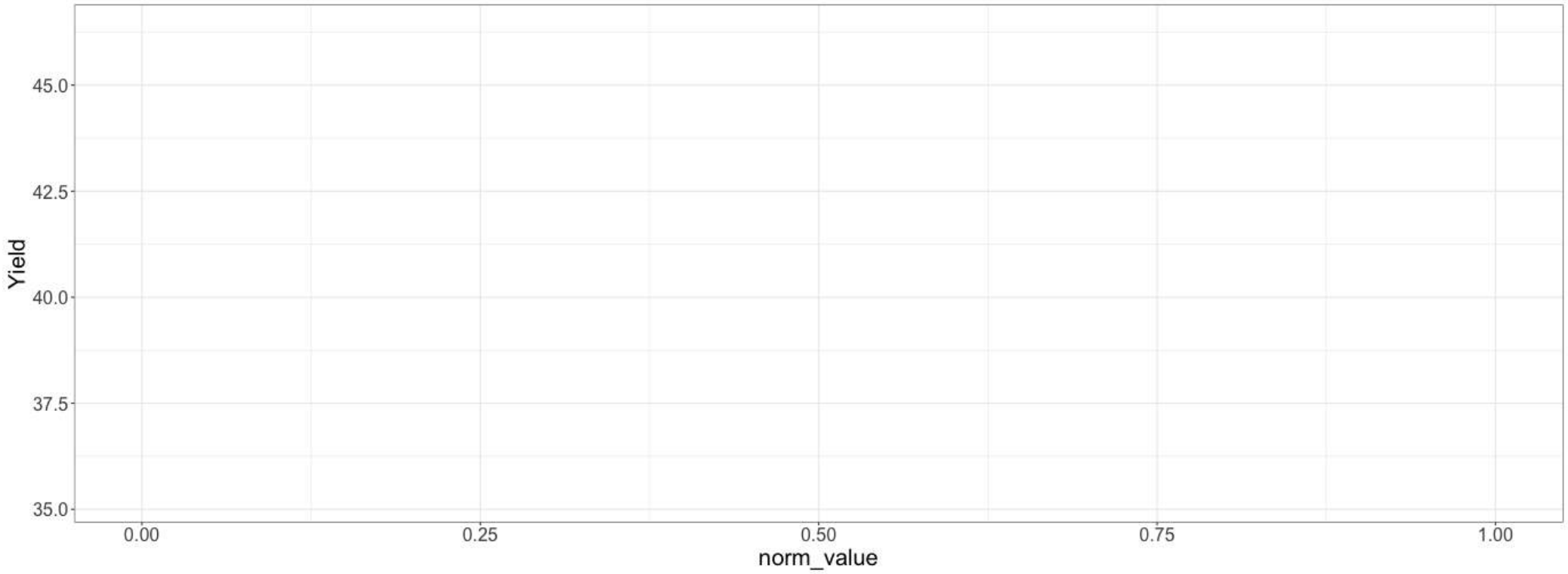
Set up: transform data for scatterplot

```
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
```

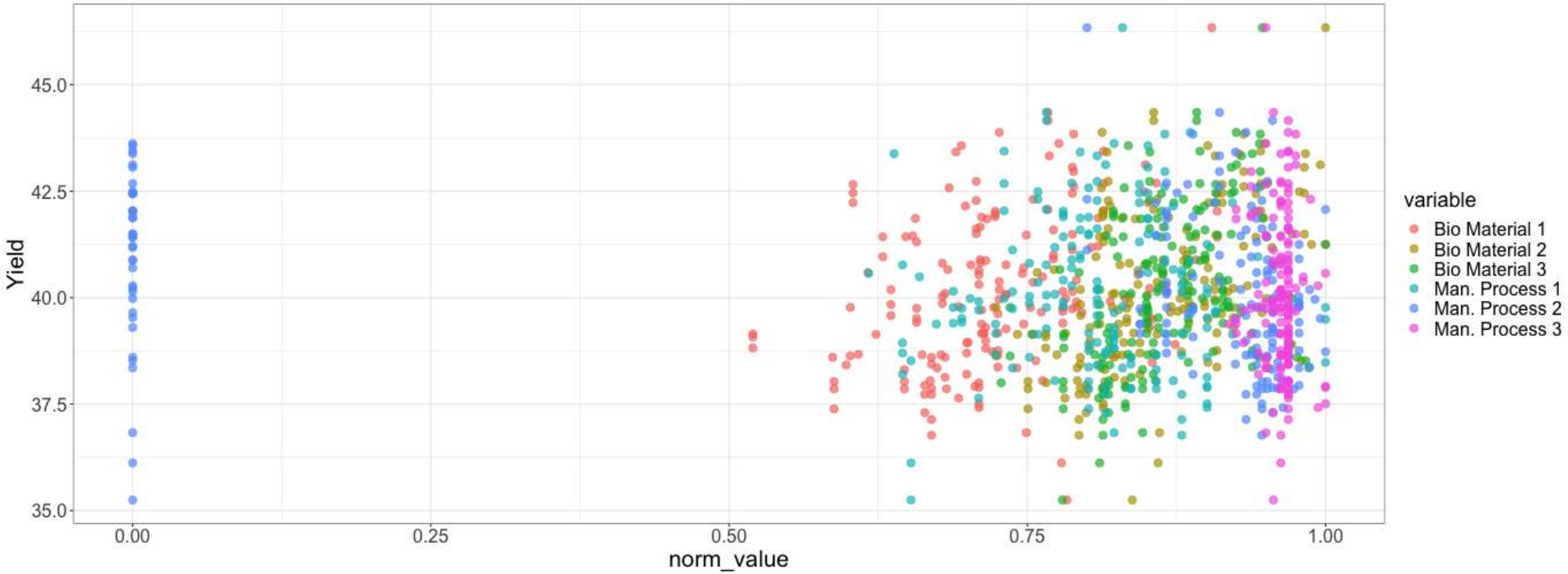
Set up & link: normalized data base plot

```
# 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

base_norm_plot
```



Set up & adjust: normalized data scatterplot

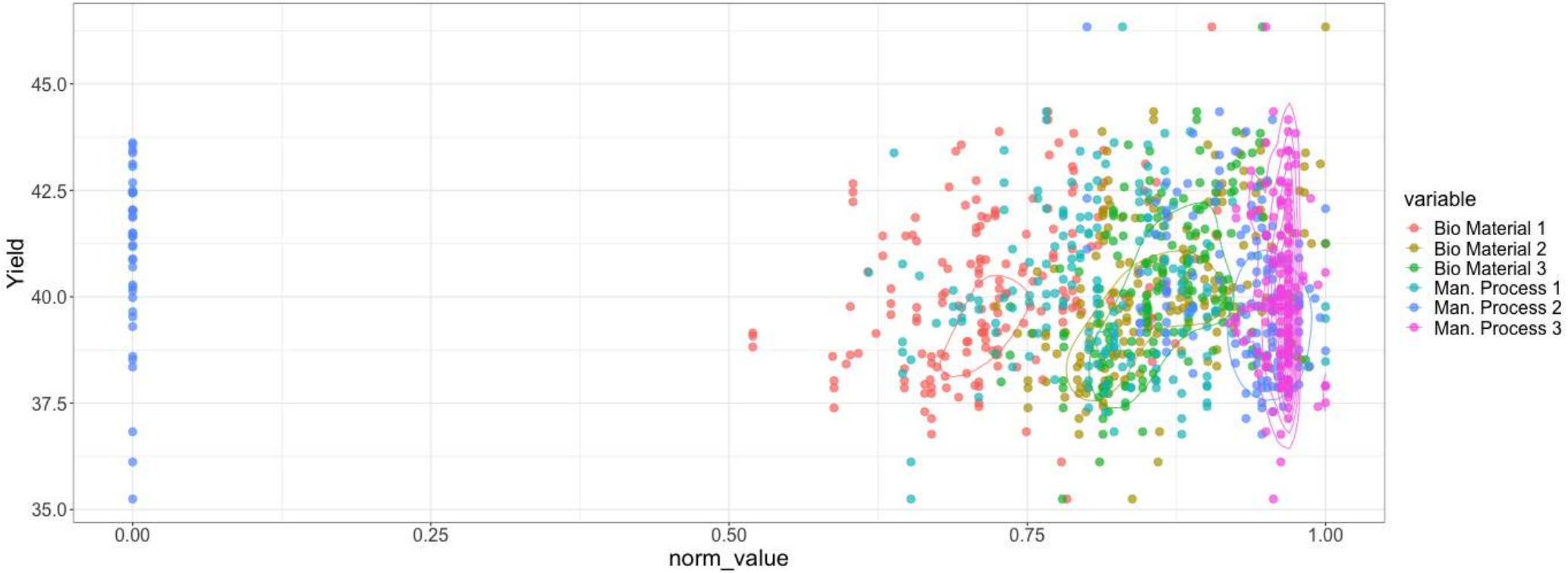
```
# Create a scatterplot.
scatter_norm = base_norm_plot +          #<- base plot
               geom_point(size = 3,      #<- add point geom with size of point = 3
                          alpha = 0.7)   #<-  make it 70% opaque

# View updated plot.
scatter_norm
```



Adjust: add density geom to scatterplot

Now add a 2D density geom to our normalized 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

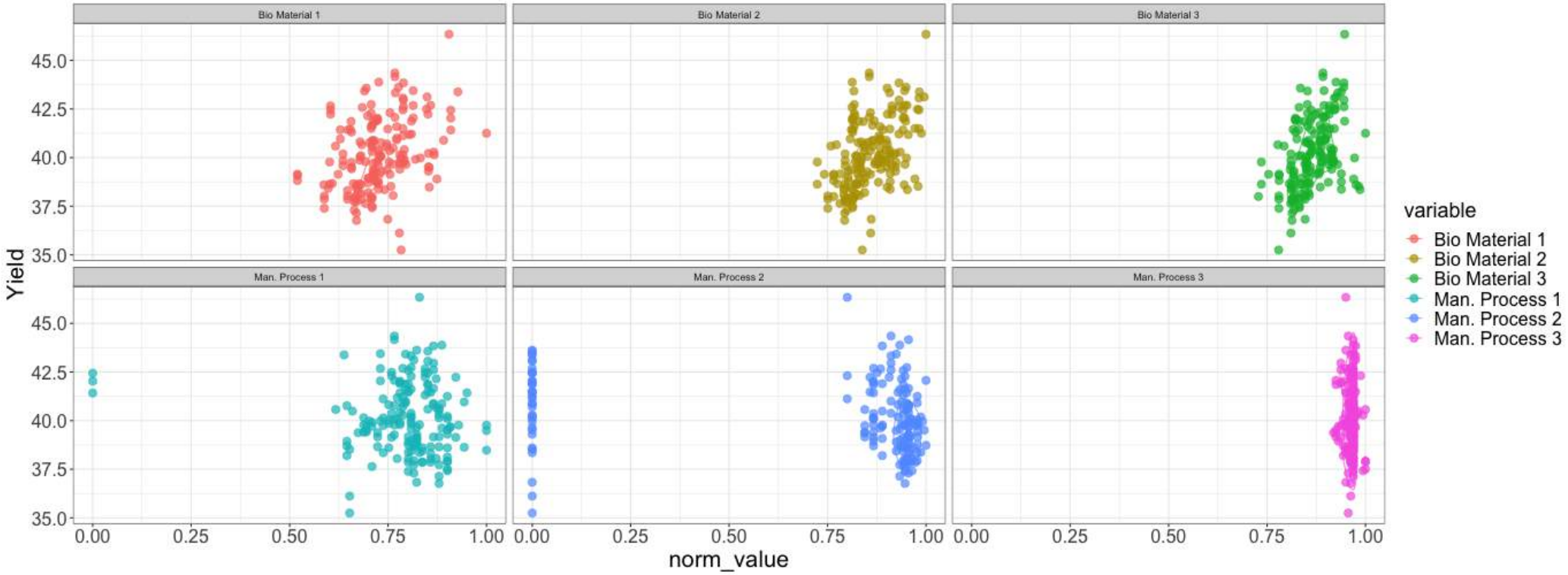
Notice that the previous plot is hard to decipher

- So, instead of overlaying densities, let's split them into individual plots called **facets**
- This can be done using ***facet_wrap()*** function which splits the data by one or more variables and plots these subsets together

```
# 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
```

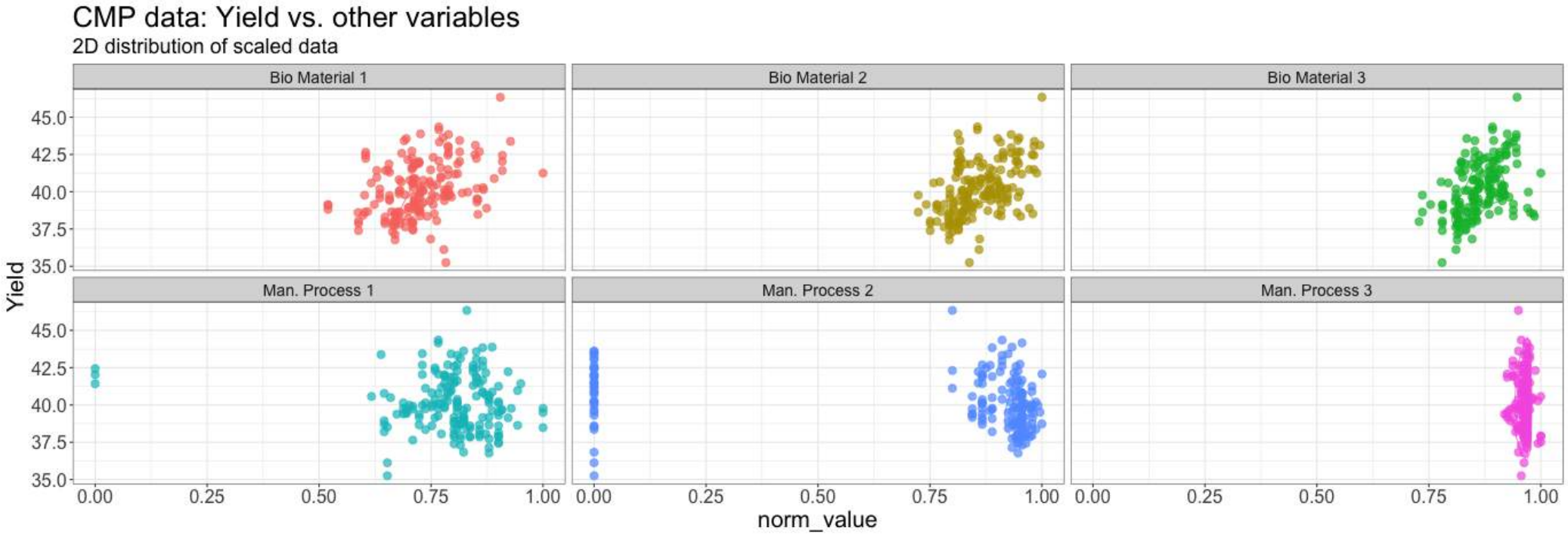
Adjust: wrap scatterplots in facets (cont'd)

```
# 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
```



Module completion checklist

Objective	Complete
Create a scatterplot with ggplot2	✔
Transform data using tidyverse to prepare for compound visualizations	✔
Visualize a boxplot of transformed data with ggplot2	✔
Visualize a scatterplot of transformed data with ggplot2	✔
Save plots in R	

Saving plots

- Once we have created a plot in R, we may want to save it to a file so that we can use it in another document
- The first step in saving plots is to decide which output format we want to use
- Some of the available file formats are: `PNG`, `JPEG`, `BMP`, and `PDF`
- We will explore the `PNG` and `PDF` formats

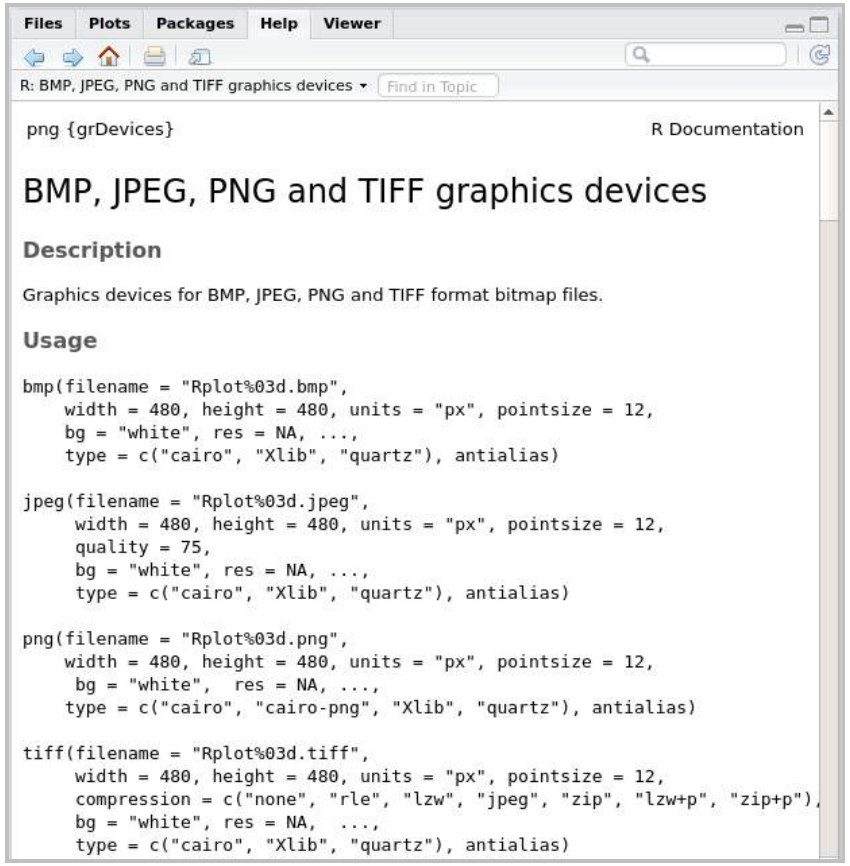
Saving plots: export to PNG

`png()` function opens R graphics device and lets us save our plots in PNG file format

- The syntax of the function is as follows:

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

plot 1   #<- call the plot object you want to export

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

- `dev.off()` command allows to clear R graphics device so that we can continue working with our plots
- `bmp`, `jpeg`, and other graphic export commands use a similar command format



Saving plots: export to PNG (cont'd)

```
# 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()
```

- When the graphics device is cleared, we should get a similar message in our console

```
RStudioGD
        2
```

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

```
RStudioGD
        2
```

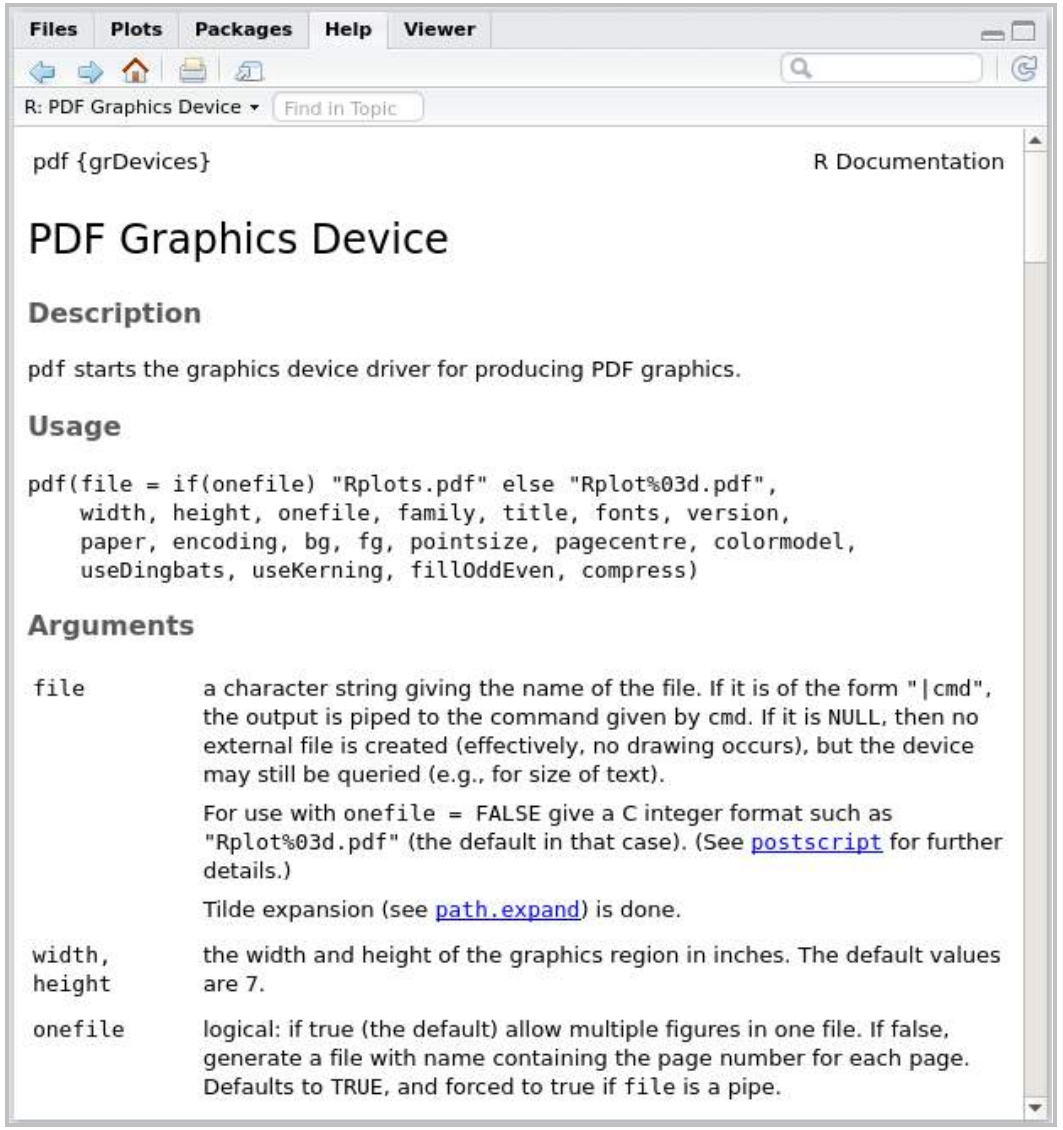
Saving plots: export to PDF

- There are a few advantages of saving 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 we zoom in!
  - We can **save multiple plots into a single PDF document** with one command

Saving plots: export to PDF (cont'd)

`pdf()` function follows the same syntax as `png()`:

```
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
```



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: export to PDF (cont'd)

Let's save more than one plot together in a pdf

```
# 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
scatter_norm         #<- plot 2

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

```
RStudioGD
        2
```

Knowledge check 2

Exercise 2

Module completion checklist

Objective	Complete
Create a scatterplot with ggplot2	✔
Transform data using tidyverse to prepare for compound visualizations	✔
Visualize a boxplot of transformed data with ggplot2	✔
Visualize a scatterplot of transformed data with ggplot2	✔
Save plots in R	✔

Summary

- In this module, we learned to:
  - Transform and tidy data using `tidyverse`
  - Use transformed data to build complex visualizations and layered plots using `ggplot2`
- In the next module, we will explore and learn to create plots using the `highcharter` package

This completes our module

Congratulations!