Data Wrangling: Day Two

AUTHOR
Brian Gural, Lorrie He, JP Flores

Case Study

Last class, we had introduced a dataset/experiment that we would work through. Let's remind ourselves of some of the details:

- We have proportions of cell types across samples
- There controls made of mostly pure cell types (`fractions`) and experimental samples (`whole`)
- There are `.csv` files for both the cell type proportion data and the sample phenotypes

There are a few things to think about when wrangling/exploring data:

- What do you want to know about this data?
- What kind of visuals would you want to make?
- How does the data need to be formatted to get those visuals?
- What are some expected features of our data?

Take a moment to talk among yourselves about this/any ideas you had since last class!

Getting familiar with the data

Proportions Data

```
# Load the data. The sample IDs were stored as the first row, so lets
cell_props <- read.csv("wrangling-files/cellProportions.csv",
                       row.names = 1)

head(cell_props)
```

```
           Cardiomyocytes Fibroblast Endothelial.Cells Macrophage
whole_2             0.652     0.0886           0.06700     0.1761
fraction_13         0.824     0.0370           0.06387     0.0692
fraction_12         0.895     0.0213           0.04436     0.0390
fraction_19         0.000     0.9983           0.00167     0.0000
fraction_18         0.000     1.0000           0.00000     0.0000
whole_16            0.820     0.0208           0.08889     0.0501
           Pericytes.SMC
whole_2          0.01672
fraction_13      0.00558
fraction_12      0.00000
fraction_19      0.00000
```

```
fraction_18       0.00000
whole_16          0.02058
```

> **Tip**
>
> Our data fits the `tidy` style, since each row is a sample (observation) and each column is a different cell type (variable).

When assessing data, it's good to *consider what features you'd expect from a given data set*. This helps you know if something has gone wrong before you've gotten your hands on it.

We're looking at the proportion of cell types in each sample, which should sum up to 1. Checking that the values in each row add to 1 would help confirm that we have what we're expecting:

```
rowSums(cell_props)
```

```
    whole_2 fraction_13 fraction_12 fraction_19 fraction_18
whole_16
          1           1           1           1           1
1
   whole_15    whole_12 fraction_14     whole_8    whole_14
whole_3
          1           1           1           1           1
1
 fraction_9 fraction_11     whole_6 fraction_16  fraction_8
whole_4
          1           1           1           1           1
1
 fraction_5  fraction_1    whole_10 fraction_10     whole_7
whole_5
          1           1           1           1           1
1
    whole_9  fraction_6  fraction_3 fraction_17     whole_1
whole_13
          1           1           1           1           1
1
 fraction_7  fraction_4    whole_11 fraction_20  fraction_2
fraction_15
          1           1           1           1           1
1
```

> **Tip**
>
> Raw RNA-seq matrices should go up to 100,000s. So if you only see small numbers in the data, it's likely been manipulated in some way.

Phenotype data

We also have the phenotypes for the samples in a separate file:

```
cell_phenos <- read.csv("wrangling-files/cellPhenotypes.csv",
                        row.names = 1)

str(cell_phenos)
```

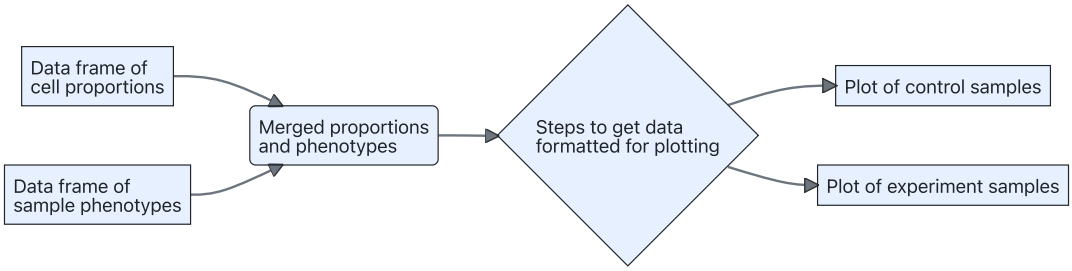
```
'data.frame':   36 obs. of  3 variables:
 $ type     : chr  "purified_cardiomyocytes"
"purified_cardiomyocytes" "purified_cardiomyocytes"
"purified_cardiomyocytes" ...
 $ genotype : chr  NA NA NA NA ...
 $ treatment: chr  NA NA NA NA ...
```

Planning the analysis

We want to know:

- If the controls look as we'd expect
- What group differences there are

To get at the question about controls, we'd need to check `cell_phenos` to see which samples are from the control or experimental groups. After, we'll plot the proportions.



Manipulating data frames

Summarizing and subsetting

Let's get more context on what's in the data. `table` is a convenient way to summarize columns and lists:

```
# What unique values and how many of each are in the "genotype" field
table(cell_phenos$genotype)
```

```
cmAKO    WT
    8     8
```

```
# Table can also compare two variables. useNA need to be added to inc
table(cell_phenos$type, cell_phenos$genotype,  useNA = "ifany")
```

```
                           cmAKO WT <NA>
  purified_cardiomyocytes      0  0    9
  purified_endothelial_cells   0  0    3
  purified_fibroblasts         0  0    8
  whole_tissue                 8  8    0
```

Seems that the purified cell types list NA for their genotype and that there are three types. Also, we have 8 knock-outs and 8 are wild type.

Combining and reordering

Data frames can be combined in a bunch of ways, but no matter the method it's essential that the order of samples match. R has two built-in methods, binds (`cbind` and `rbind`) and `merge`.

Captain Planet and the Planeteers likely combined using merge functions

Binds slap two data frames together. `cbind` adds columns, `rbind` adds rows. Binds don't consider the order of the data sets, so there's a risk of things being out of order.

`merge` is similar to `cbind`, but matches the data sets based on a common column.

`cbind`

```
# They don't, lets take a closer look
cbind(rownames(cell_phenos), rownames(cell_props)) |> head()
```

```
     [,1]         [,2]
[1,] "fraction_1" "whole_2"
[2,] "fraction_2" "fraction_13"
[3,] "fraction_3" "fraction_12"
[4,] "fraction_4" "fraction_19"
[5,] "fraction_5" "fraction_18"
[6,] "fraction_6" "whole_16"
```

Above, you can see that `cbind` would mismatch the samples. The first row would have the phenotype info for `fraction_1` but the proportion data for `whole_2`. **Always be careful when using `cbind`! It has no guardrails!**

```
# Reorder one to match the other
# This uses the cell_phenos rownames as a list to specify the order o
cell_props <- cell_props[rownames(cell_phenos),]

# They should all be TRUE now
all(rownames(cell_phenos) == rownames(cell_props))
```

```
[1] TRUE
```

```
# Now we can merge them
data_bind <- cbind(cell_phenos, cell_props)

head(data_bind)
```

```
                            type genotype treatment Cardiomyocytes
Fibroblast
fraction_1 purified_cardiomyocytes     <NA>      <NA>          0.911
0.0160
fraction_2 purified_cardiomyocytes     <NA>      <NA>          0.941
0.0118
fraction_3 purified_cardiomyocytes     <NA>      <NA>          0.898
0.0165
fraction_4 purified_cardiomyocytes     <NA>      <NA>          0.869
0.0508
fraction_5 purified_cardiomyocytes     <NA>      <NA>          0.946
0.0129
fraction_6    purified_fibroblasts     <NA>      <NA>          0.000
0.8976
           Endothelial.Cells Macrophage Pericytes.SMC
fraction_1            0.0398     0.0328       0.00000
fraction_2            0.0266     0.0202       0.00000
fraction_3            0.0446     0.0339       0.00720
fraction_4            0.0235     0.0524       0.00434
fraction_5            0.0215     0.0199       0.00000
fraction_6            0.0138     0.0184       0.07029
```

`merge`

```
# Specify row.names as the feature to merge by
data_merge <- merge(cell_phenos, cell_props, by = "row.names")
```

```
head(data_merge)
```

```
    Row.names                   type genotype treatment
Cardiomyocytes
1  fraction_1 purified_cardiomyocytes     <NA>      <NA>
0.911
2 fraction_10    purified_fibroblasts     <NA>      <NA>
0.000
3 fraction_11 purified_cardiomyocytes     <NA>      <NA>
0.772
4 fraction_12 purified_cardiomyocytes     <NA>      <NA>
0.895
5 fraction_13 purified_cardiomyocytes     <NA>      <NA>
0.824
6 fraction_14 purified_cardiomyocytes     <NA>      <NA>
0.928
  Fibroblast Endothelial.Cells Macrophage Pericytes.SMC
1    0.01600            0.0398    0.03277      0.000000
2    0.99385            0.0050    0.00114      0.000000
3    0.06261            0.0599    0.09814      0.007635
4    0.02127            0.0444    0.03896      0.000000
5    0.03702            0.0639    0.06917      0.005581
6    0.00861            0.0416    0.02198      0.000179
```

While this won't always be the case with `merge` vs. `bind`, its better to use `merge` in this scenario, since it helps keep your script *interpretable*

> **Reproducible code**
>
> If you continue with programming, you'll need to share your code or return to code you wrote months ago. Writing easy-to-understand scripts gives you less headache later!

Preparing for different visualizations

At this point, we should ask ourselves a few questions:

- What am I trying to see about the data?
- What kind of plot helps us see that?

Take a minute to talk as a group about how you would visualize the data!

What am I trying to see about the data?

Our samples have data on the proportions of many cell types. I'd want to easily compare all of these cell types at once, with samples/groups side-by-side.

What kind of plot do we want?

Pie charts are often used to visualize percents/proportions, but its difficult to see differences between two pie charts. A stacked bar plot would be a better fit, since we're trying to compare different sample groups.

What format does my data need to be to make said plot?

This stacked bar plot would have:

- Samples on the X-axis
- Cell-type proportions on the Y-axis
- Colors for each cell type in each bar

For `ggplot` to make this our data needs to have a column for each term, but the data is spread across many columns. To solve this, we first need to understand the concepts of wide and long data.

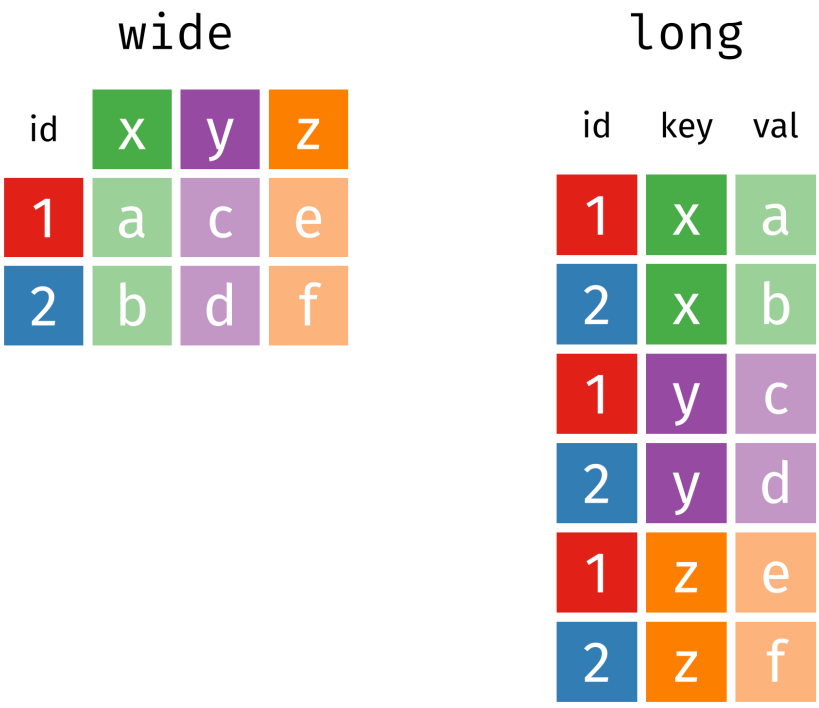
Pivoting wide and long

Data is often formatted as *wide* or *long*. Our data is in a wide format, which has a single row for each sample and a column for each variable. When wide data is pivoted into a long format columns are condensed together.

Ross understands the importance of converting wide and long data

It's easiest to understand how pivoting works in visuals:

Still images | Animated transition



Source: Garrick Aden-Buie's (@grrrck) Tidy Animated Verbs
github.com/gadenbuie/tidyexplain

As a reminder, we want to make a **column of proportions values** (val) and a **column specifying cell types** (key).

```
library(tidyverse)
```

```
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✔ dplyr     1.1.4     ✔ readr     2.1.5
✔ forcats   1.0.0     ✔ stringr   1.5.1
✔ ggplot2   3.5.0     ✔ tibble    3.2.1
✔ lubridate 1.9.3     ✔ tidyr     1.3.1
✔ purrr     1.0.2
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
# cell types are specified with cols = and name the new column with n
# values originally in those columns are going to move to a new value
data_long <- pivot_longer(data_merge,
                          cols = c(Cardiomyocytes, Fibroblast, Endoth
                          names_to = "cell.type", values_to = "propor

str(data_long)
```

```
tibble [180 × 6] (S3: tbl_df/tbl/data.frame)
 $ Row.names : 'AsIs' chr [1:180] "fraction_1" "fraction_1"
"fraction_1" "fraction_1" ...
 $ type      : chr [1:180] "purified_cardiomyocytes"
"purified_cardiomyocytes" "purified_cardiomyocytes"
"purified_cardiomyocytes" ...
 $ genotype  : chr [1:180] NA NA NA NA ...
 $ treatment : chr [1:180] NA NA NA NA ...
 $ cell.type : chr [1:180] "Cardiomyocytes" "Fibroblast"
"Endothelial.Cells" "Macrophage" ...
 $ proportion: num [1:180] 0.9115 0.016 0.0398 0.0328 0 ...
```

We have a couple of changes:

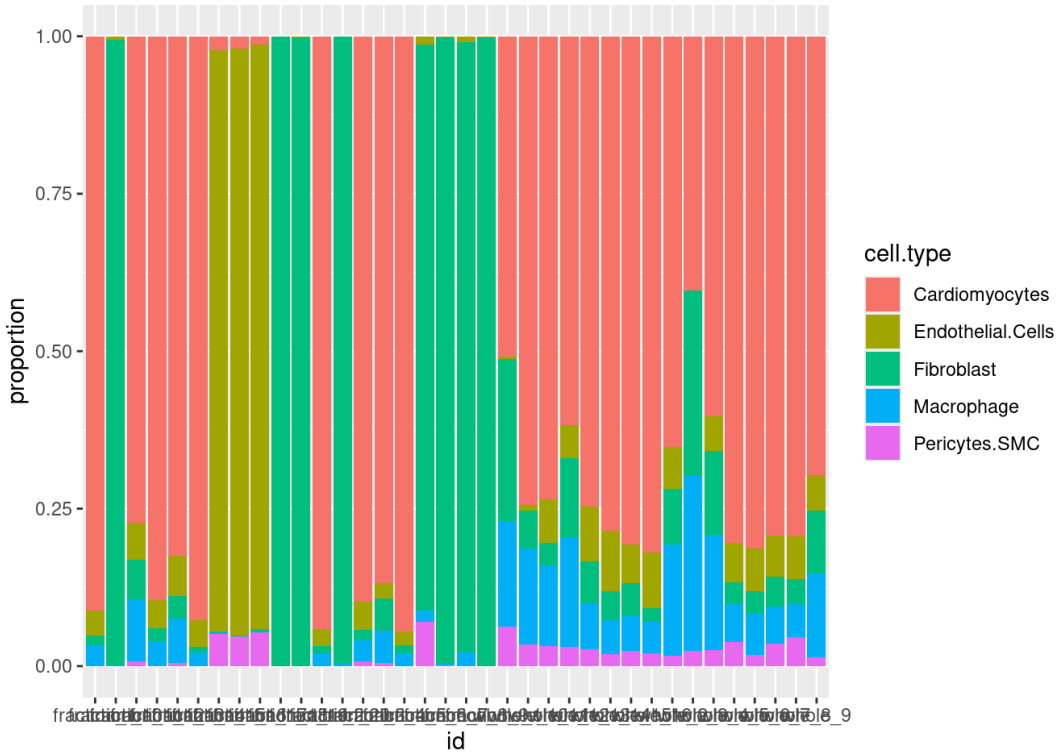
- There are two new columns, `cell.type` and `proportion`
- We have A LOT more rows than we did originally
- The sample IDs were coerced to a column “Row.names” that is an ‘AsIs’ character. We’ll need to correct that before we plot the data

Wrangling for plotting

Pure cell-type fraction controls

With our data in this format, we can make a lot of cool plots. Lets start with the bar plot we had planned.

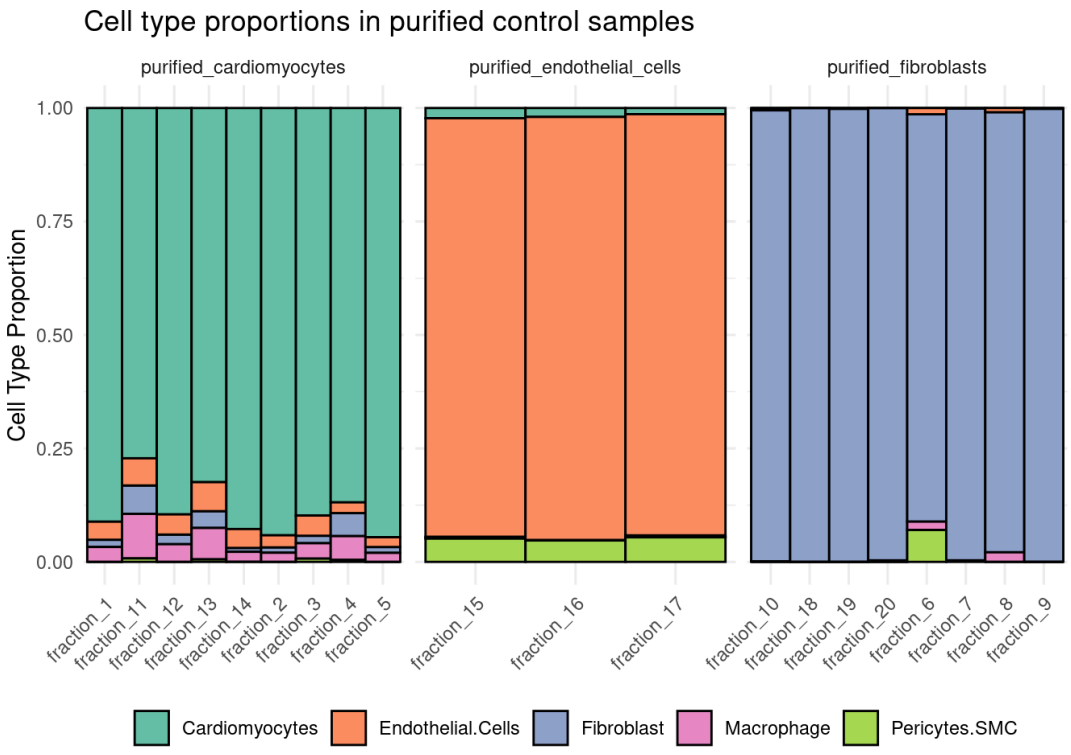
```
data_long |>
  mutate(id = as.character(Row.names)) |> # fix the AsIs type
  ggplot(aes(x = id, y = proportion, fill = cell.type))+
  geom_bar(position="fill", stat="identity")
```



It worked, but it looks... less than pleasing. Lets remind ourselves of what we wanted to see in the plot: groups side-by-side.

I'd like to start by making a plot just for the controls for now. `filter` from the `dplyr` package will help separate the groups. Also, I'll make aesthetic changes to make it easier to compare groups and nicer to look at.

```
data_long |>
  filter(type != "whole_tissue") |>
  mutate(id = as.character(Row.names)) |>
  ggplot(aes(x = id, y = proportion, fill = cell.type))+
  geom_bar(position="fill", stat="identity", color = "black", width =
  facet_grid(cols=vars(type), scales = "free") +
  scale_fill_manual(values = c("#66C2A5","#FC8D62", "#8DA0CB", "#E78A
  theme_minimal() +
  theme(
    axis.title.x = element_blank(),
    legend.title = element_blank(),
    legend.position = "bottom"
  ) +
  guides(x = guide_axis(angle = 45)) +
  labs(title = "Cell type proportions in purified control samples",
       y = "Cell Type Proportion")
```



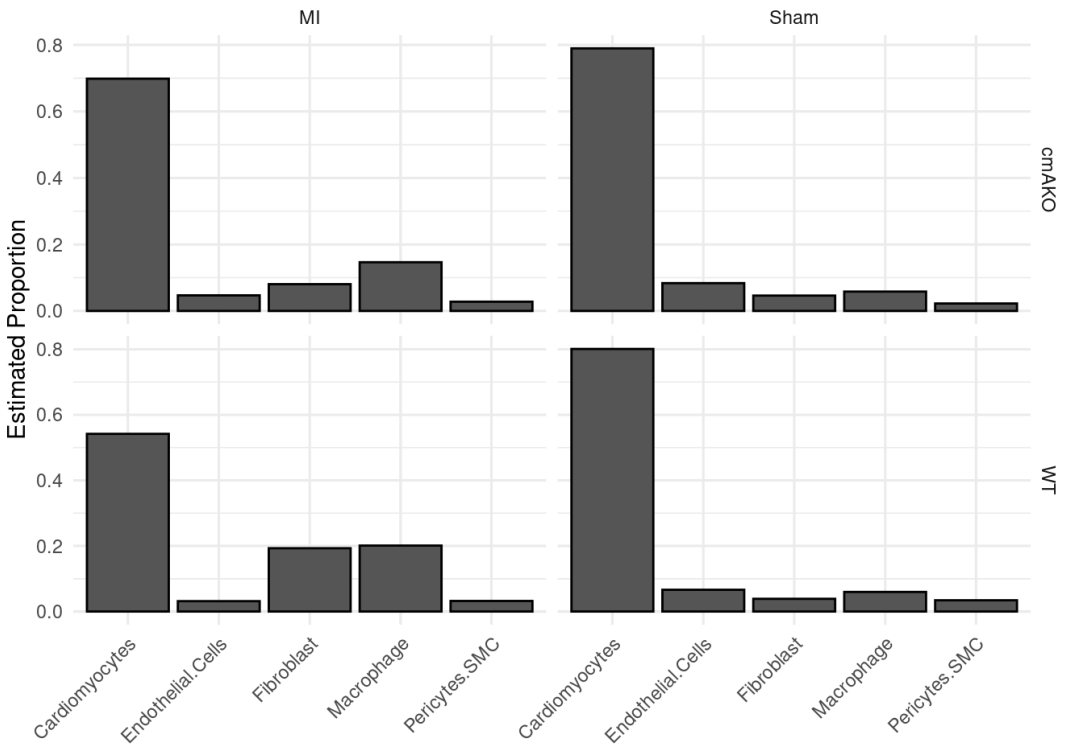
This looks good! We can see what we expected of our control samples. Each of the fractions are made up of a single cell type. Let's move onto the experimental samples.

Experimental Samples

There are two things we should consider before we visualize differences between our experimental groups:

- It would be easier to compare shifts in specific cell types if we break up the stacked bar chart so that the cell types are spread across the x-axis.
- In our last plot, we compared samples across a single phenotypic factor: `type`. This time, it's more complicated because we want to we want to compare both `genotype` and `treatment`.

```
data_long   |>
  filter(type == "whole_tissue") |>
  ggplot(aes(x = cell.type, y = proportion)) +
  geom_bar(stat = "summary", fun = mean, width = 0.9,  color = "black
  facet_grid(genotype ~ treatment, scales = "free") +
  theme_minimal() +
  theme(
    axis.title.x = element_blank(),
    legend.title = element_blank(),
    legend.position = "bottom"
  ) +
  labs(y = "Estimated Proportion") +
  scale_fill_manual(values = c("#66C2A5","#FC8D62", "#8DA0CB", "#E78A
  guides(x = guide_axis(angle = 45))
```



We just made three major changes:

- `cell.type` is on the x-axis, not sample `id`s
- We're plotting the **mean** of each cell type across many samples in each group. `geom_bar` can do this automatically with `stat = "summary", fun = mean,`
- We're showing four plots at once by having `facet_grid` contrast them with `genotype ~ treatment`

However, I think it's still tough to compare across the groups. Also, only showing the mean masks any variation within groups. Lets make two more major changes to fix that:

- Put all of the groups into a single plot
- Add dots for each sample onto each bar

And to make it easier to read, lets reorder the X-axis by most to least abundant cell types.

Reorder cell types

We can take advantage of `factors` to reorder, since `ggplot` references the order of factors when plotting.

```
# Find the most-to-least abundant cell types
cell.type.order <- data_long |>
  filter(type == "whole_tissue") |>
  group_by(cell.type) |> # Manipulate the data within cell-type group
  mutate(mean = mean(proportion)) |> # make a new column that is the
  arrange(desc(mean)) |> # arrange by mean proportion
  pull(cell.type) |> # pull out the cell type column as a list
  unique() # remove duplicated values
cell.type.order
```

```
[1] "Cardiomyocytes"    "Macrophage"        "Fibroblast"
[4] "Endothelial.Cells" "Pericytes.SMC"
```

Put all groups on a single plot

If we combine `genotype` and `treatment` into a single variable, we can condense down to a single plot. While we're at it, we can apply `cell.type.order` to make the `data_long$cell.type` into a factor-level column:

```
data_long <- data_long |>
  mutate(cell.type = factor(cell.type, levels = cell.type.order),
         Genotype_Treatment = factor(paste(genotype, "-", treatment),
```

Plot

```
# Generate boxplot
data_long  |>
  filter(type == "whole_tissue") |>
  ggplot(aes(x = cell.type, y = proportion, fill = Genotype_Treatment
  geom_bar(stat = "summary", fun = mean, width = 0.9,  color = "black
           position = position_dodge(0.9)) +
  geom_jitter(inherit.aes = T,
              position = position_dodge(0.9),
              size = 2, alpha = 0.3) +
  labs(y = "Estimated Proportion",
       fill = "Treatment") +
  theme(
```

```
    axis.title.x = element_blank(),
    legend.title = element_blank(),
    legend.position = "bottom"
  ) +
  scale_fill_manual(values = c("#A6CEE3", "#1F78B4", "#FDBF6F", "#FF7
  guides(x = guide_axis(angle = 45))
```