Data Visualisation

with ggplot2

QuantArch Week 3 | 21-02-2022

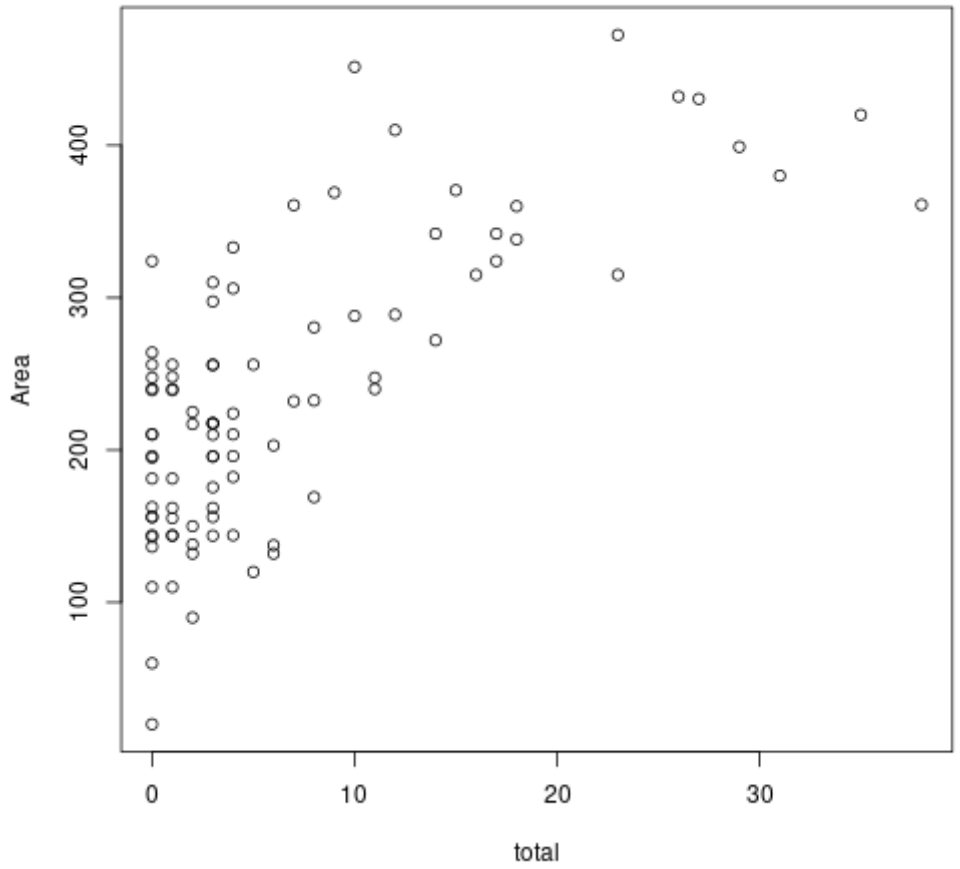
Universiteit Leiden

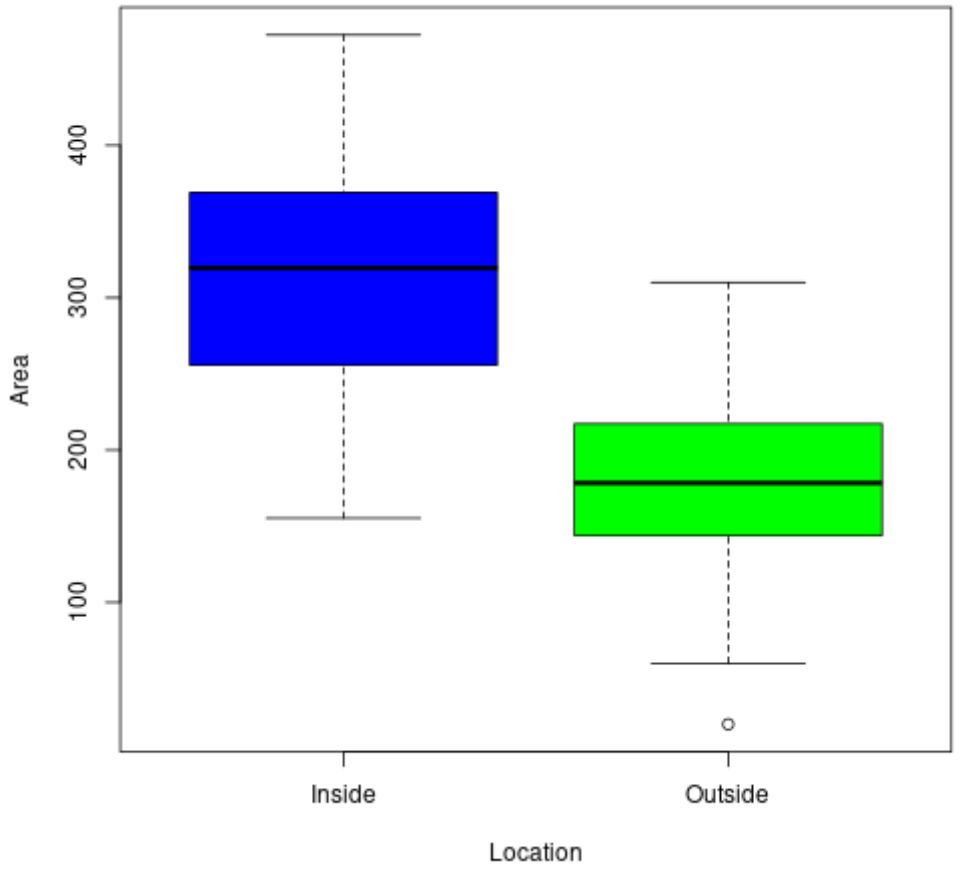
In case you missed last week's lecture

```
pits_data_raw <- readr::read_csv(here("data/house_pits_missouri.csv"), na = "N/a")

pits_data <- pits_data_raw %>%
  mutate(Segment = as_factor(Segment),
         Inside = as_factor(Inside)) %>%
  rename(Location = Inside) %>%
  rowwise() %>%
  mutate(total = sum(
    c_across(Points:Ceramics),
    na.rm = T))

pits_data_long <- pits_data %>%
  pivot_longer(cols = Points:total, names_to = "artifact", values_to = "count")
```

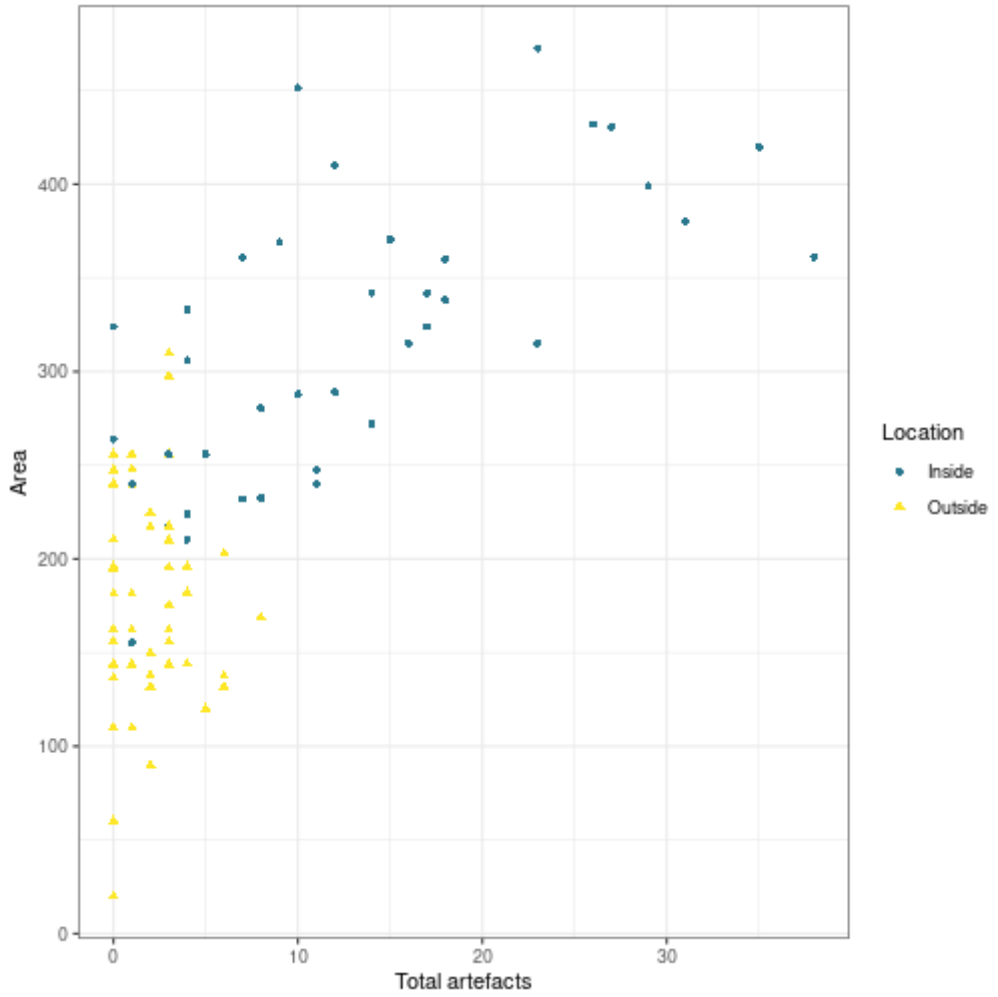
Why ggplot2?

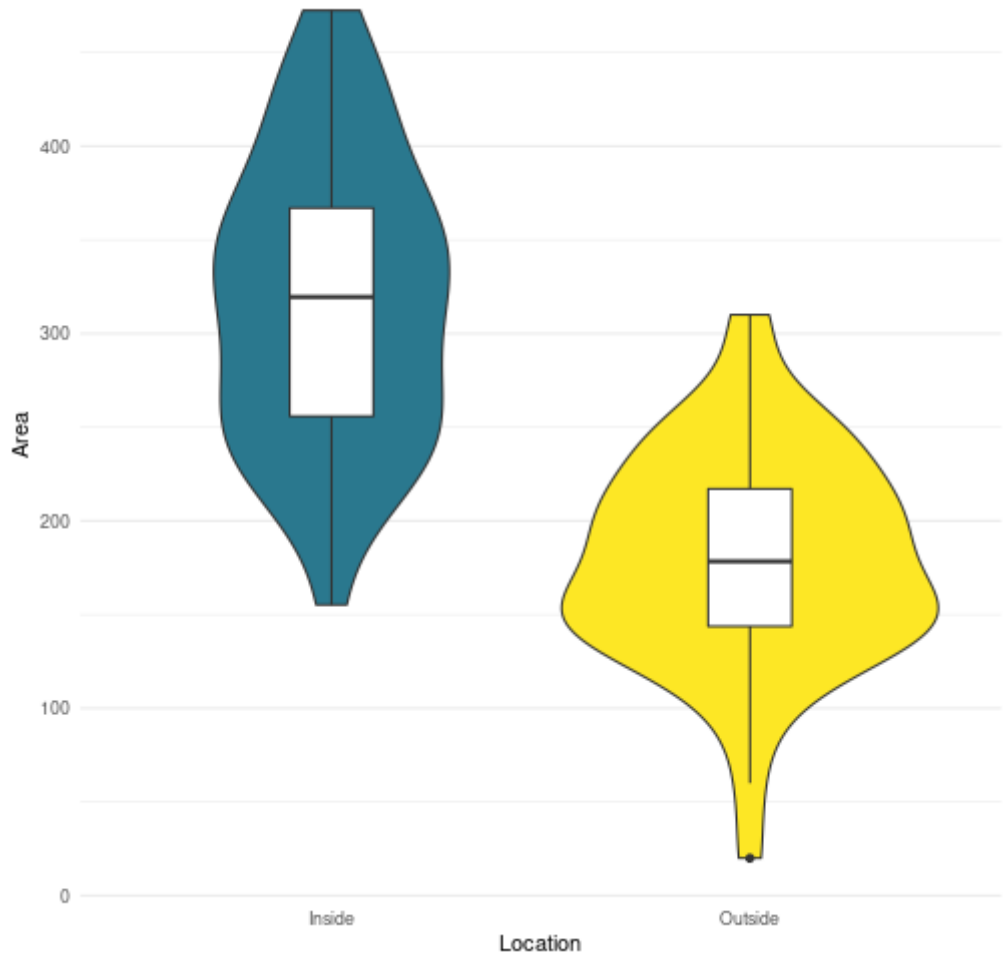
...because these are 'base' plots





…and these are ggplots 😎





ggplot2

ggplot2 is a package (included in **tidyverse**) for creating highly customisable plots that are built step-by-step by adding layers.

The separation of a plot into layers allows a high degree of flexibility with minimal effort.

Anatomy of a ggplot

```
<DATA> %>%
    ggplot(aes(<MAPPINGS>)) +
    <GEOM_FUNCTION>() +
    <CUSTOMISATION>
```

Image credit: Allison Horst

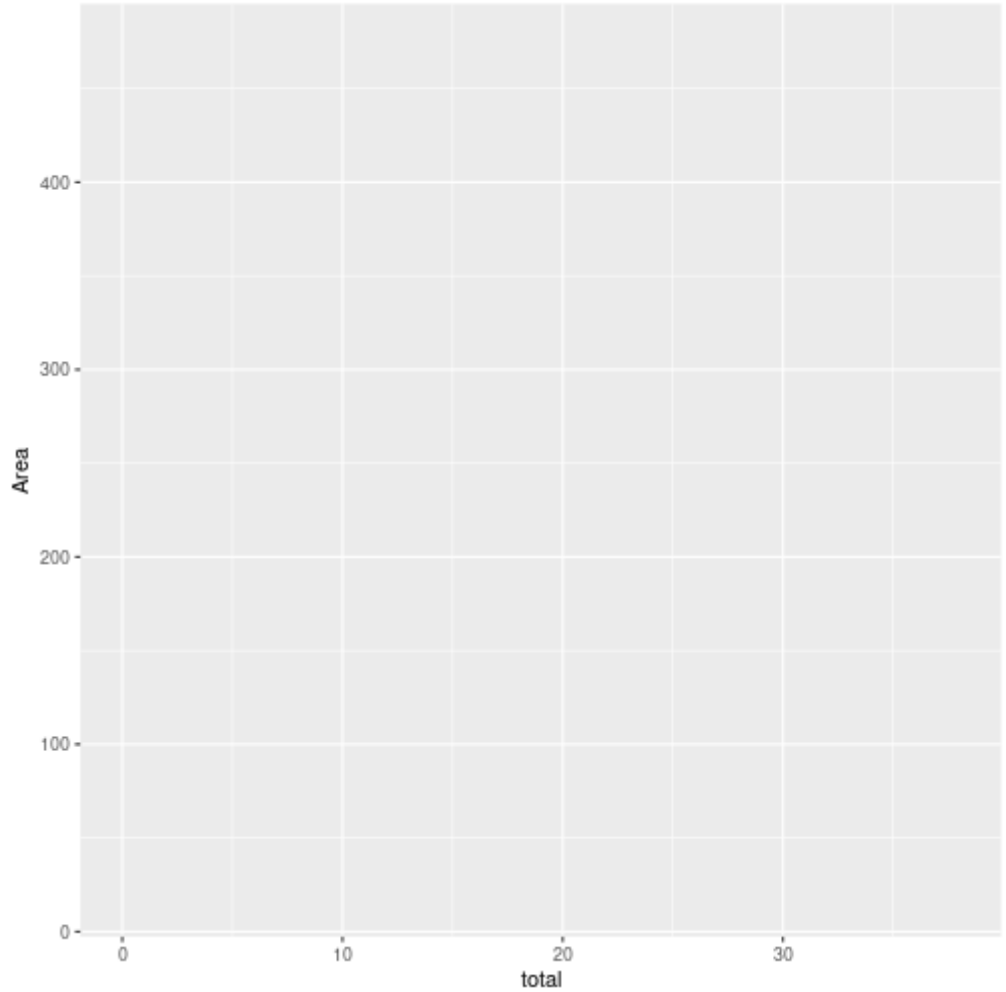
Anatomy of a ggplot

```
pits_data # <DATA>
```

```
## # A tibble: 91 × 14
##     East South Length Width Segment Location  Are
##    <dbl> <dbl>  <dbl> <dbl> <fct>   <chr>    <dbl
##  1  901.  75.1   12    12   2       Outside   144
##  2  973.  81.3   16    16   2       Outside   256
##  3  890. 163.    17    18   1       Inside    306
##  4  924. 193.    21    21.5 1       Inside    452
##  5  912. 217.    20.5  20   1       Inside    410
##  6  940. 251.    16.5  16   1       Inside    264
##  7  948. 229.    18    19   1       Inside    342
##  8  962. 212.    21    19   1       Inside    399
##  9  979. 194.     7.5   8   2       Outside    60
## 10  992. 153.    19    15.5 2       Outside   217
## # … with 81 more rows, and 4 more variables: Earp
## #   Ceramics <dbl>, total <dbl>
```

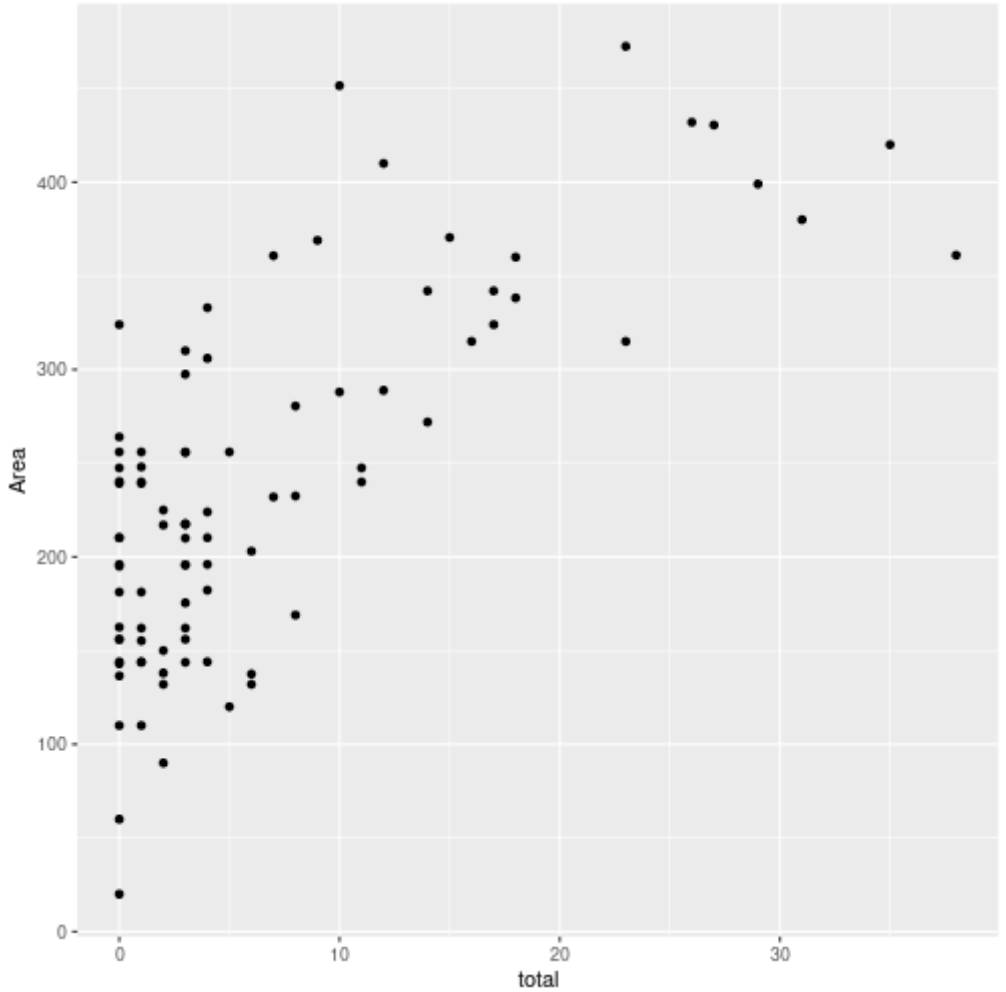
Anatomy of a ggplot

```
pits_data %>% # <DATA>
  ggplot(aes(x = total, y = Area)) # <MAPPINGS
```



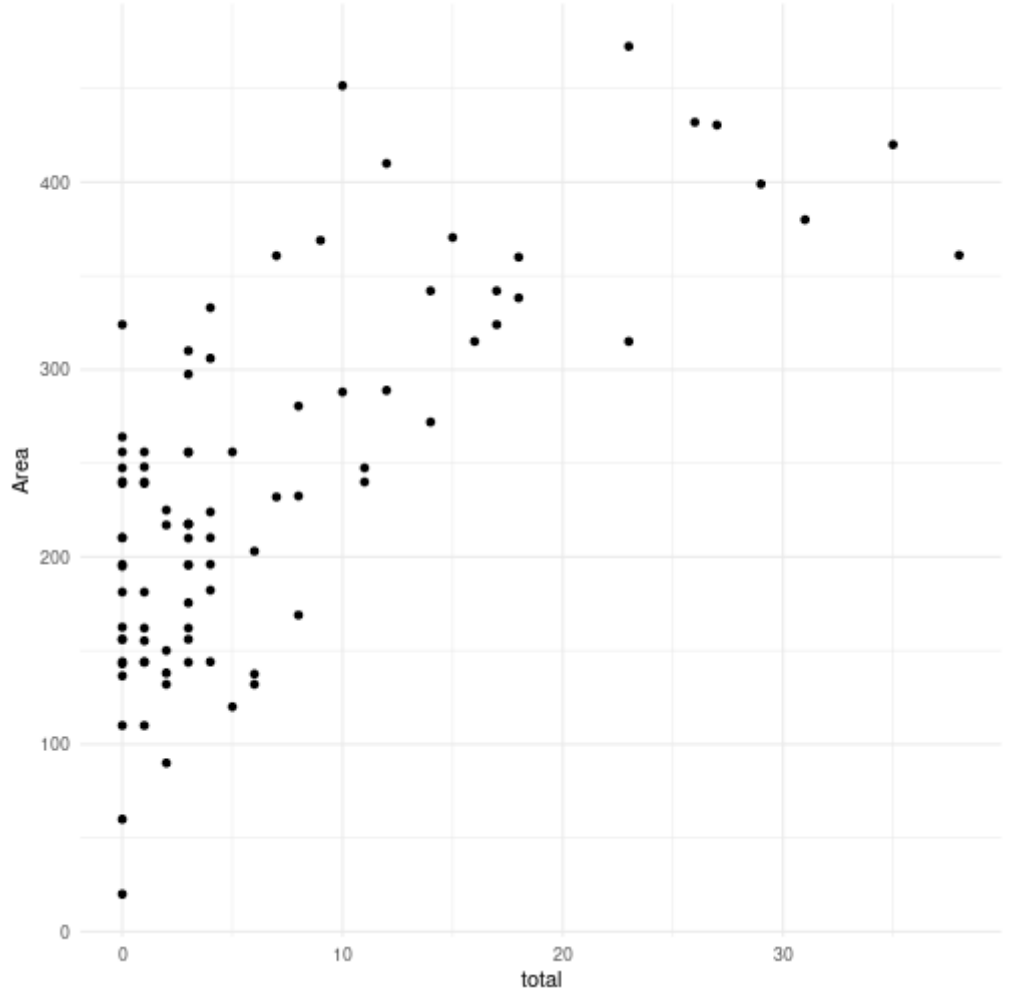
Anatomy of a ggplot

```
pits_data %>% # <DATA>
  ggplot(aes(x = total, y = Area)) + # <MAPPI
    geom_point() # <GEOM_FUNTION>
```



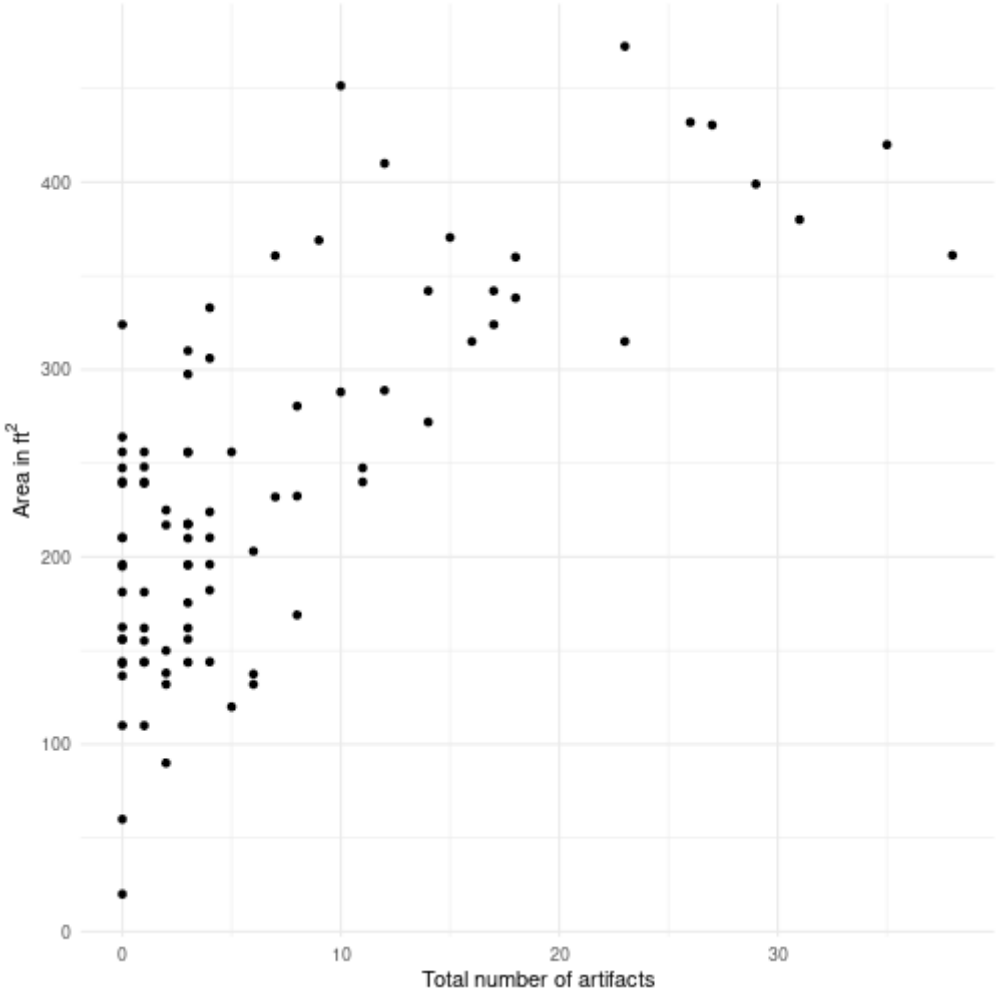
Anatomy of a ggplot

```
pits_data %>% # <DATA>
  ggplot(aes(x = total, y = Area)) + # <MAPPI
    geom_point() + # <GEOM_FUNTION>
    theme_minimal() # <CUSTOMISATION>
```



Anatomy of a ggplot

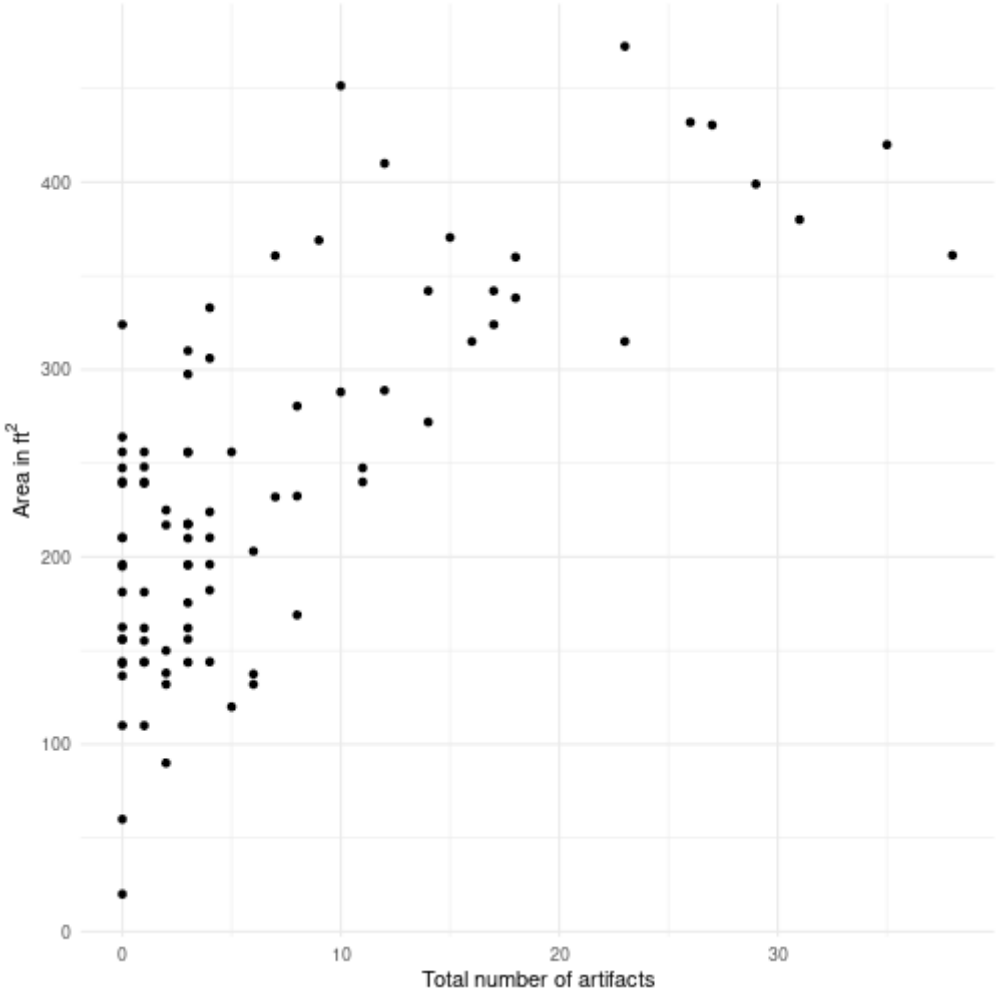
```
pits_data %>% # <DATA>
  ggplot(aes(x = total, y = Area)) + # <MAPPI
    geom_point() + # <GEOM_FUNTION>
    theme_minimal() + # <CUSTOMISATION>
    labs(x = "Total number of artifacts",
         y = bquote('Area in ft'^2)) # <CUSTOI
```



Anatomy of a ggplot

```
pits_data %>% # <DATA>
  ggplot(aes(x = total, y = Area)) + # <MAPPI
    geom_point() + # <GEOM_FUNTION>
    theme_minimal() + # <CUSTOMISATION>
    labs(x = "Total number of artifacts",
         y = bquote('Area in ft'^2)) # <CUSTOI
```

etc...



Exercise

Create a boxplot of the `Area` and `Location` variables with `geom_box`.

Solution

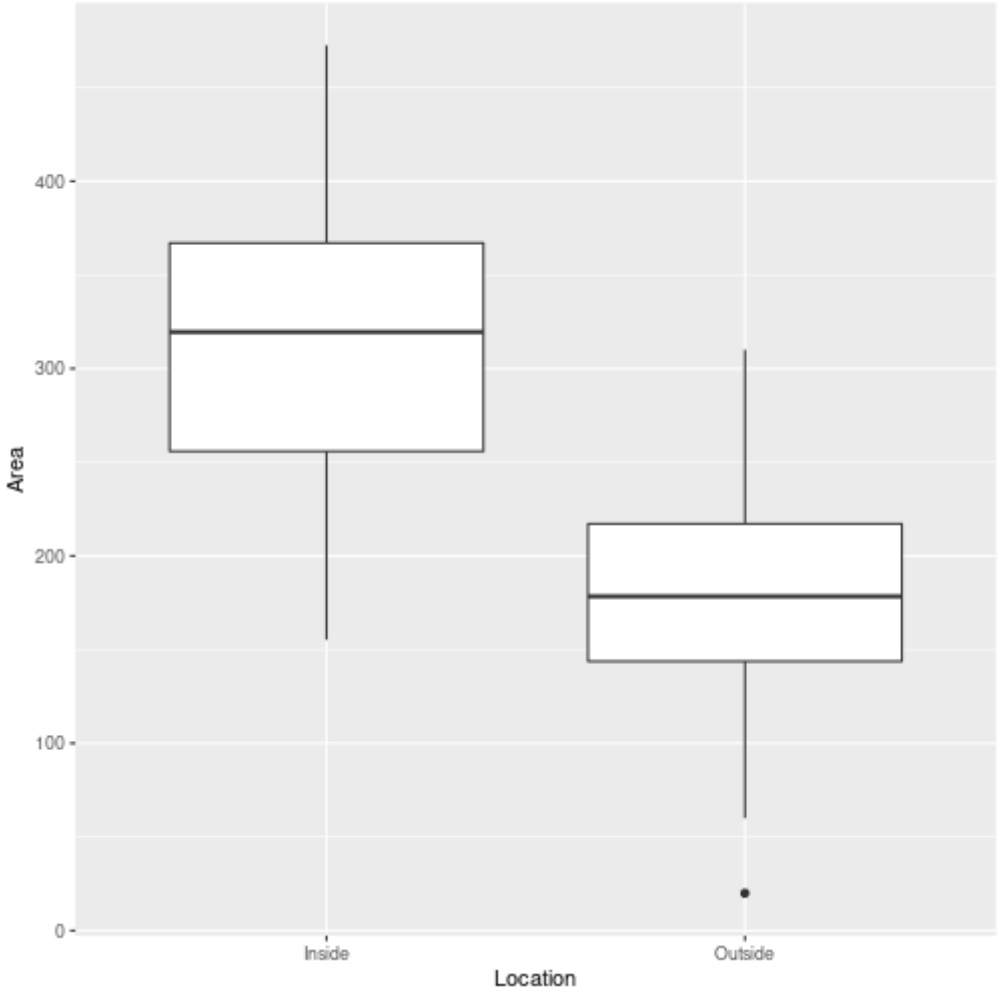
```
pits_data %>%
  ggplot(aes(x = Location, y = Area))
    geom_boxplot()
```

```
pits_data %>%
  ggplot(aes(x = Location, y = Area)) +
    geom_boxplot()
```

It's a nice overview of house pits

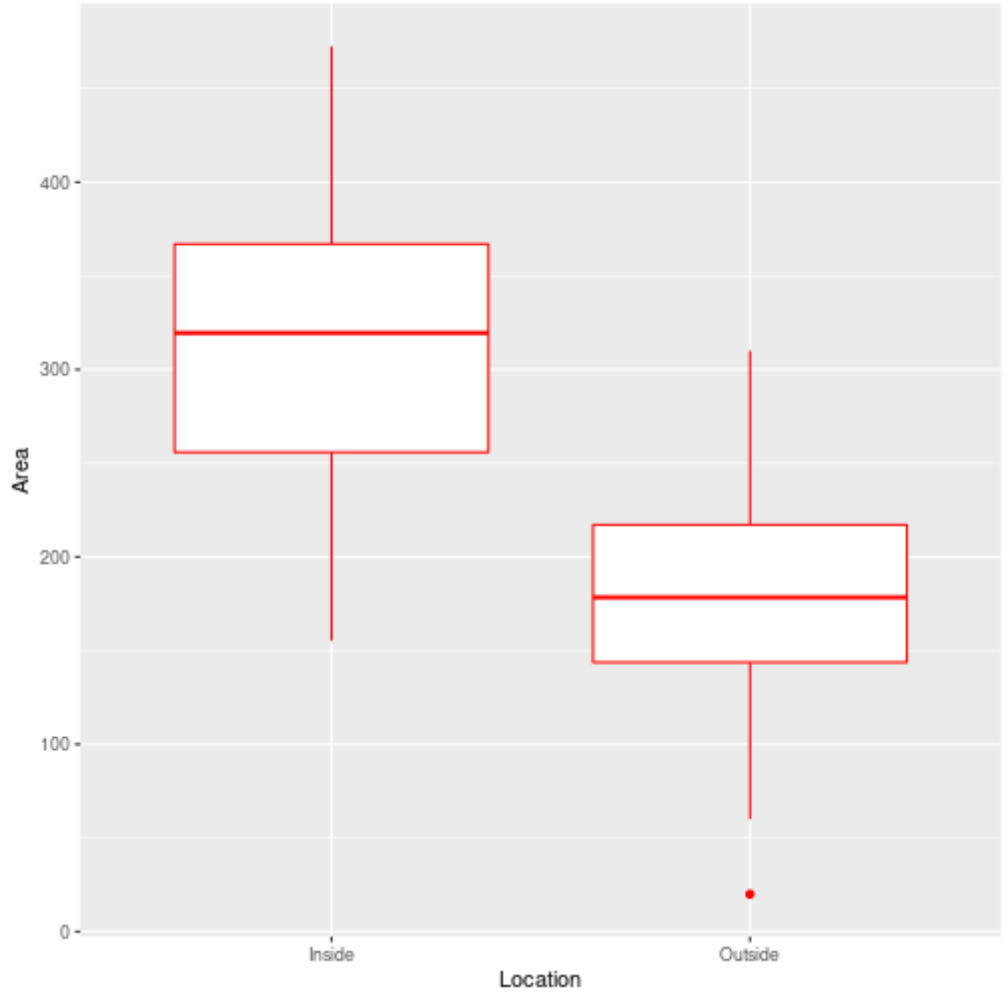
but it's not pretty

We need to add a little...



```
pits_data %>%
  ggplot(aes(x = Location, y = Area)) +
    geom_boxplot(col = "red")
```

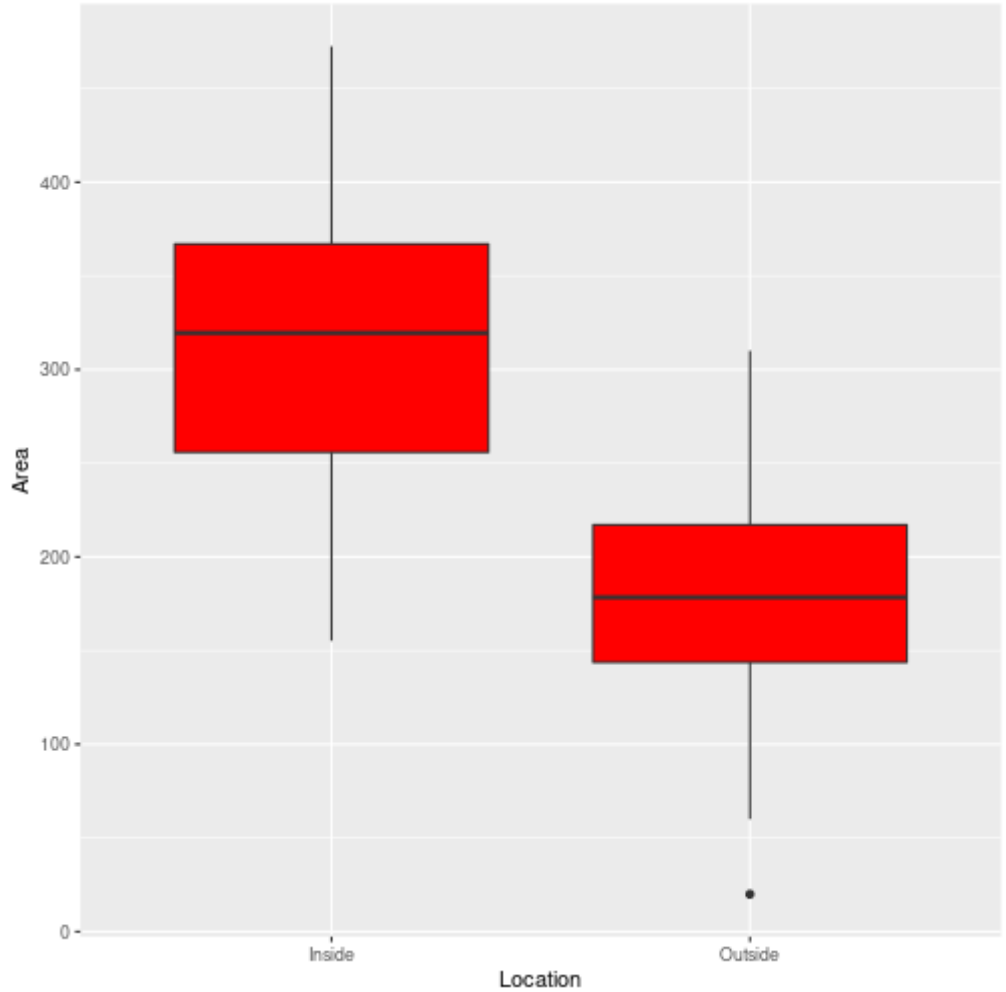
We can add a little colour 🌈



```
pits_data %>%
  ggplot(aes(x = Location, y = Area)) +
    geom_boxplot(fill = "red")
```

We can add a little colour 🌈

or fill 🎨

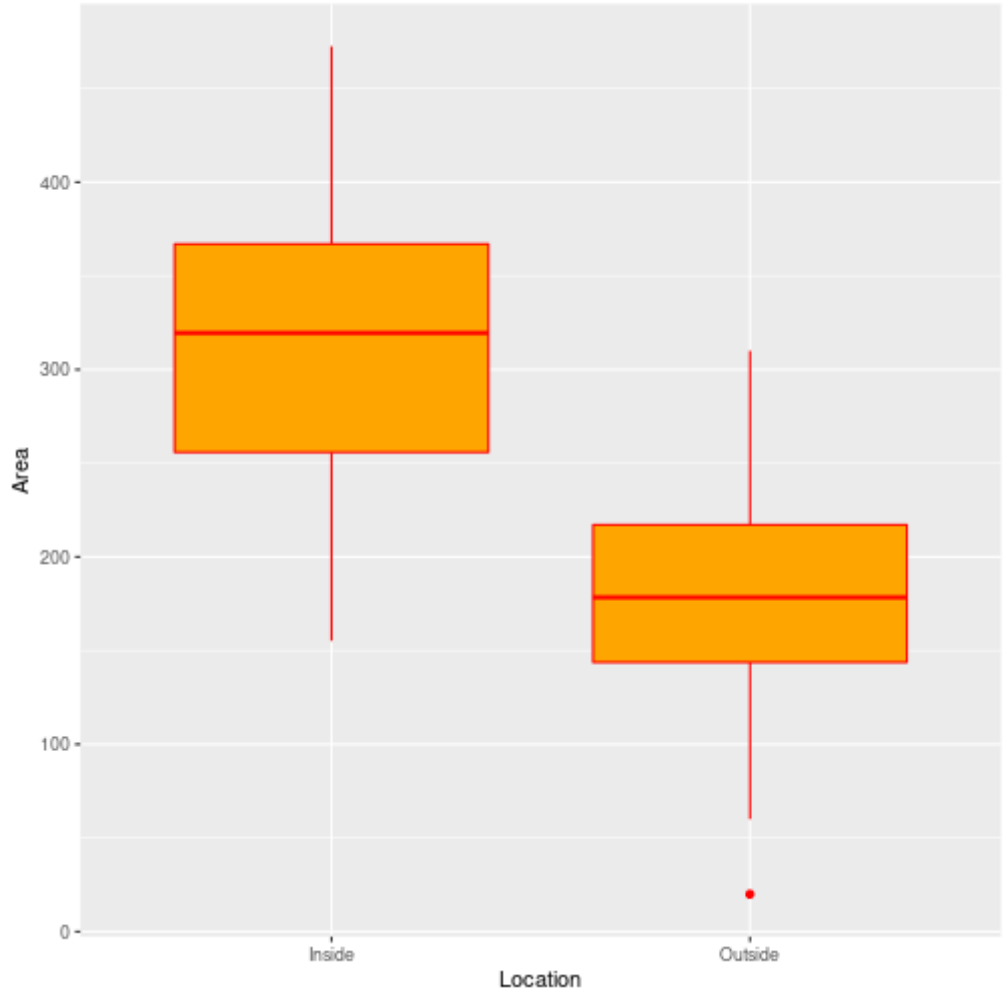


```
pits_data %>%
  ggplot(aes(x = Location, y = Area)) +
    geom_boxplot(col = "red",
                 fill = "orange")
```

We can add a little colour 🍒

or fill 🍊

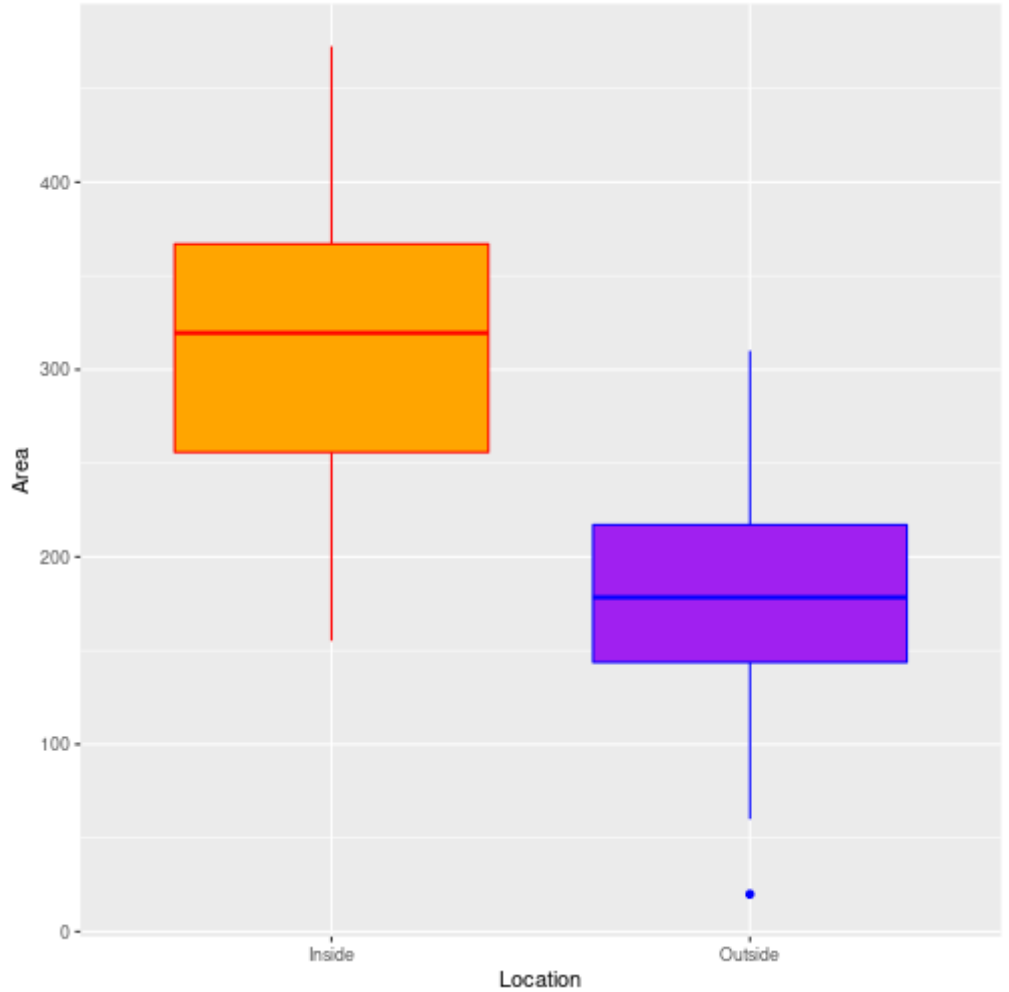
or both



```
pits_data %>%
  ggplot(aes(x = Location, y = Area)) +
    geom_boxplot(col = c("red", "blue"),
                 fill = c("orange", "purple"))
```

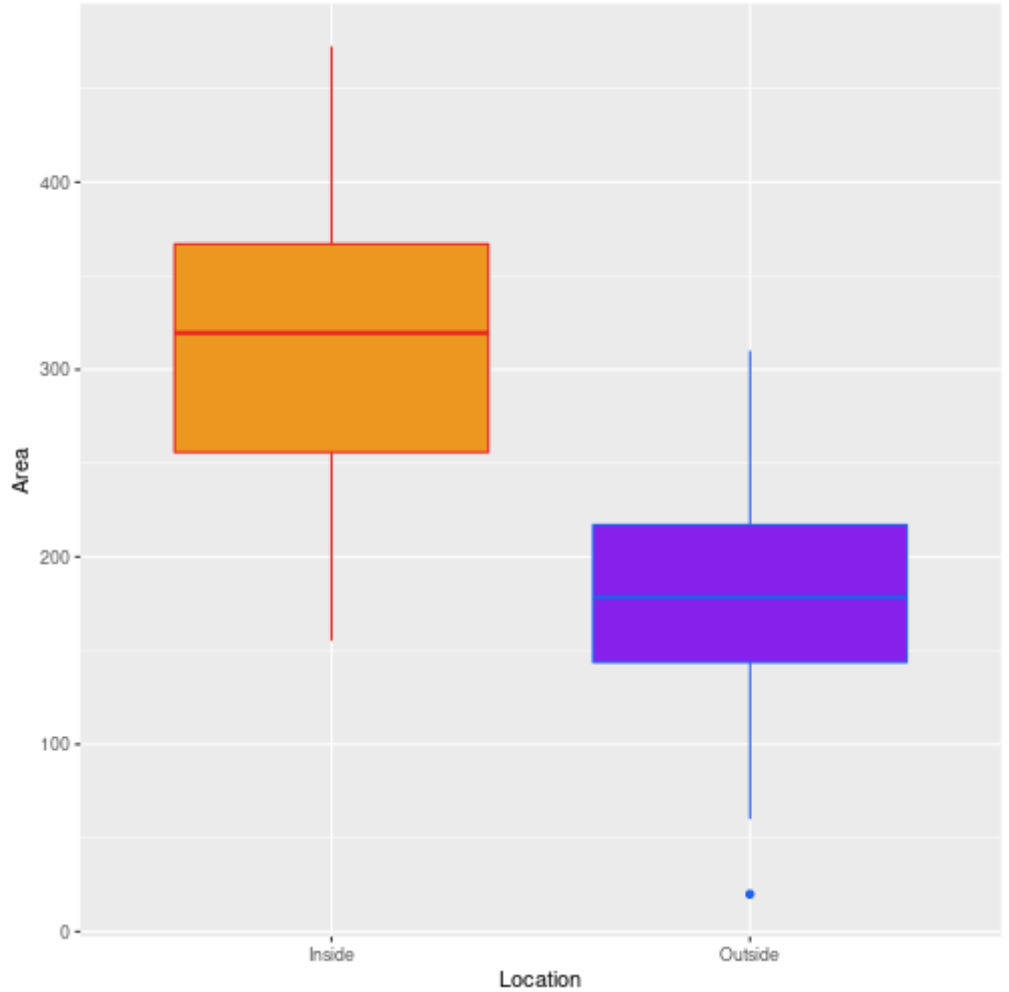
We can add multiple colours 🌈

and multiple fills 🎨



```
pits_data %>%
  ggplot(aes(x = Location, y = Area)) +
    geom_boxplot(col = c("eb2020", "2061eb"),
                 fill = c("eb9720", "8720eb"))
```

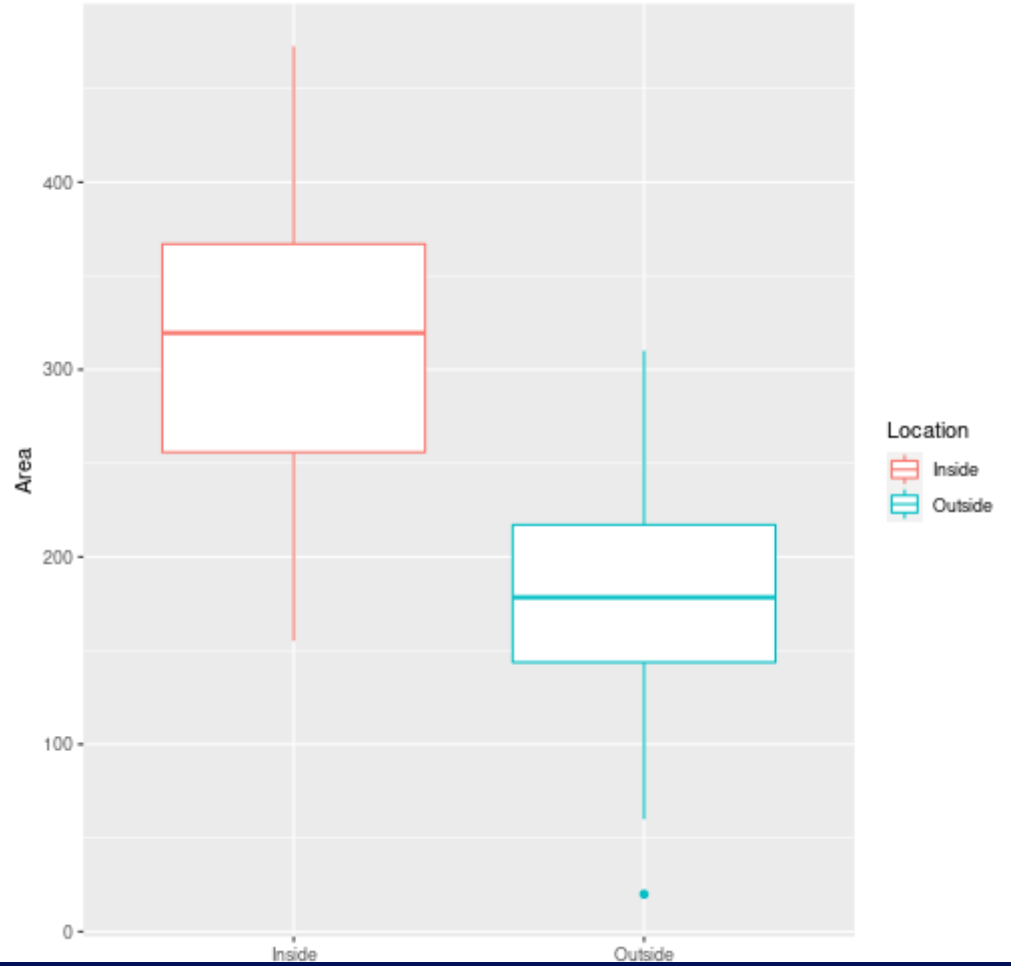
It also accepts hex colours



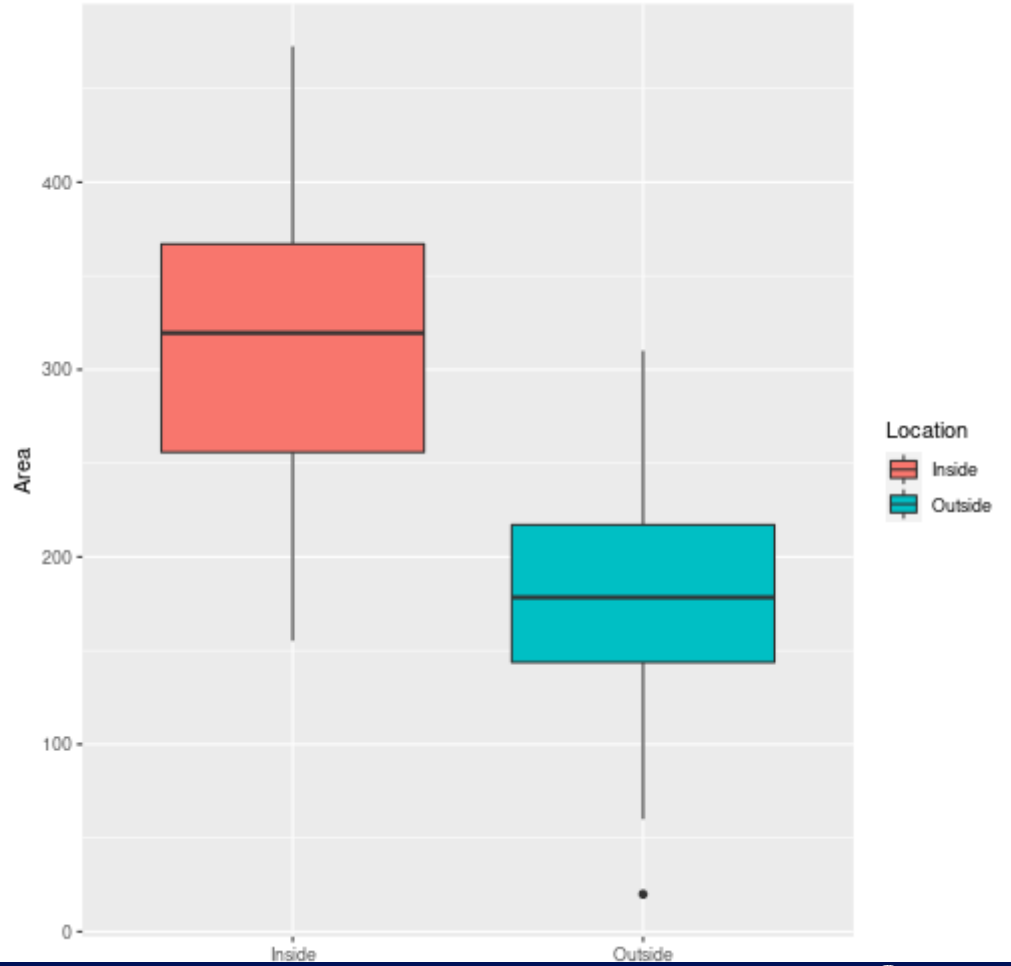
An easier way to add different colours to the variables, is to add it to `aes`

This will map the variable to a colour palette (which can of course be customised)

```
pits_data %>%
  ggplot(aes(x = Location, y = Area,
             col = Location)) +
    geom_boxplot()
```




```
pits_data %>%
  ggplot(aes(x = Location, y = Area,
             fill = Location)) +
    geom_boxplot()
```



Exercise

Make a boxplot with the `Segment` and `Area` variables

Add a `colour` or `fill` to each level of `Segment`

Solution

```
pits_data %>%
  ggplot(aes(x = Segment, y = Area,
             fill = Segment)) + # or
    geom_boxplot()
```

Adding layers

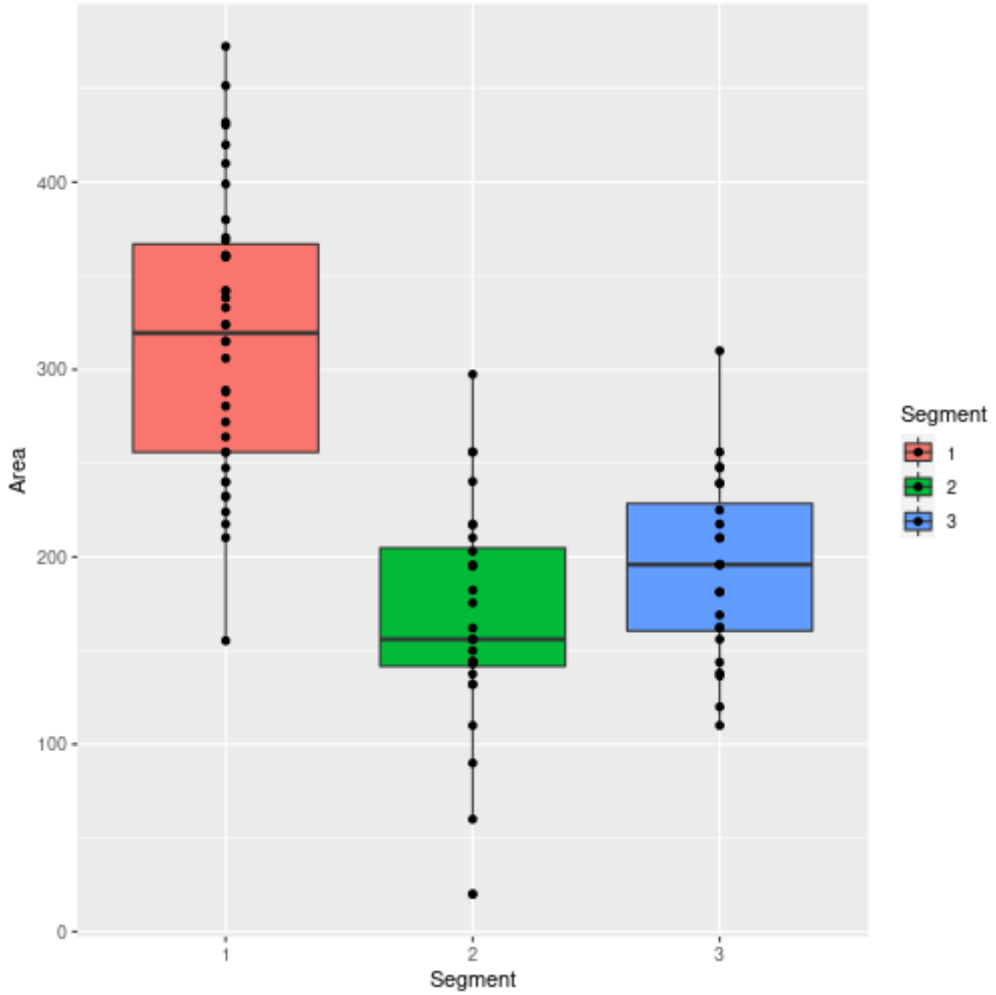
Boxplots are informative, but the information is limited

We can add more layers to the plot to add more information

Adding layers

like individual points with `geom_point`

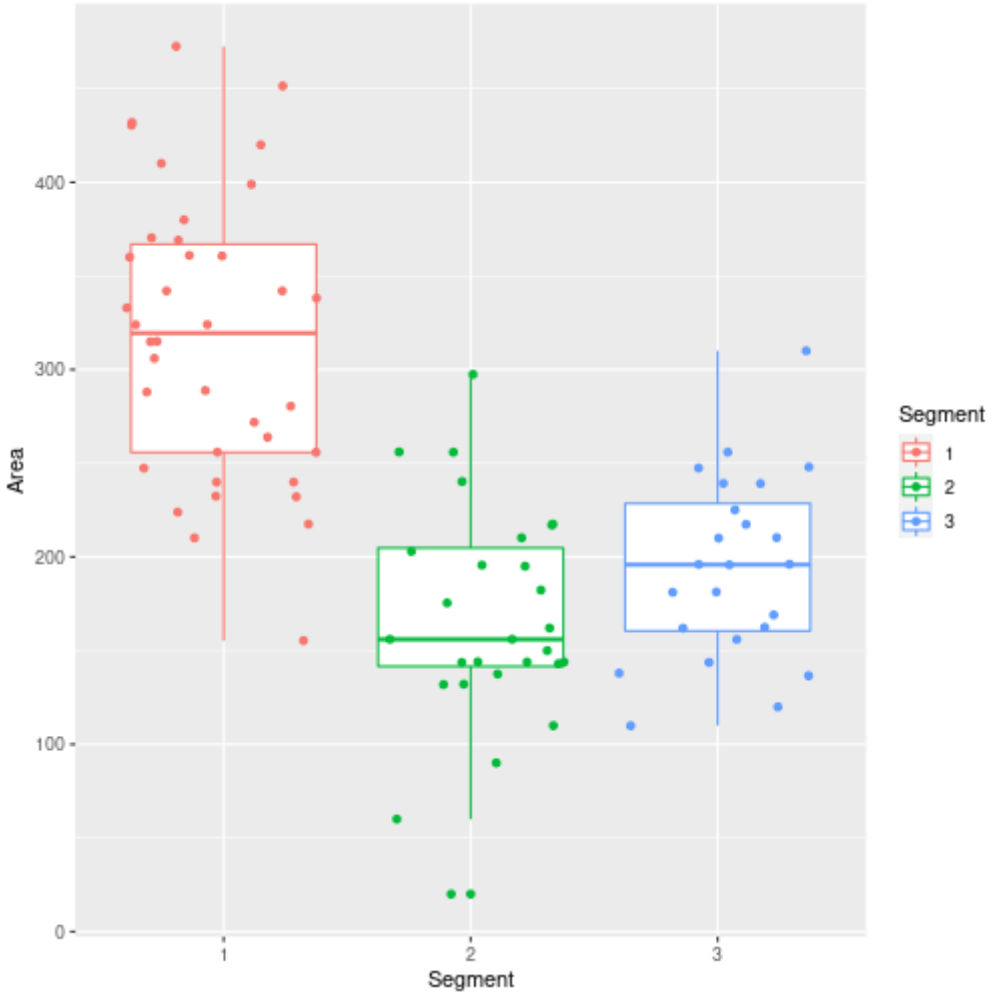
```
pits_data %>%
  ggplot(aes(x = Segment, y = Area,
             fill = Segment)) + # or col = Seg
    geom_boxplot() +
    geom_point()
```

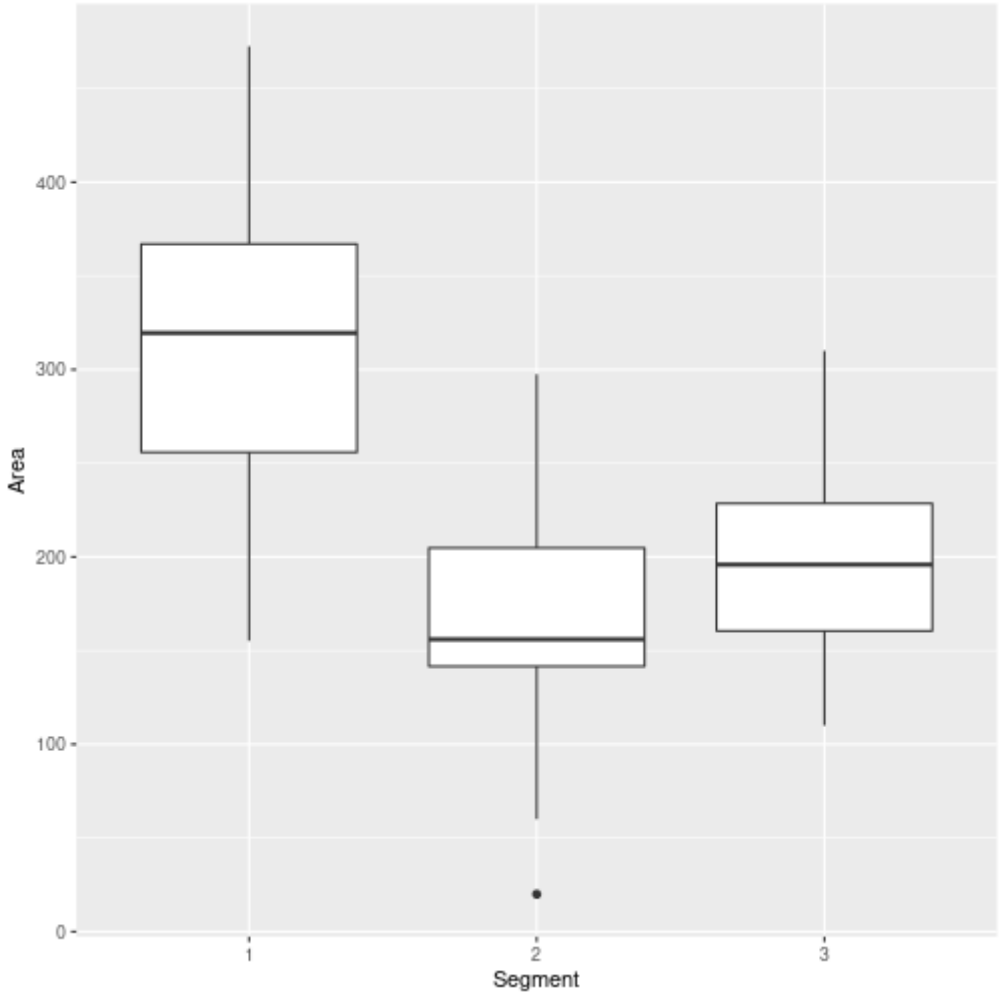


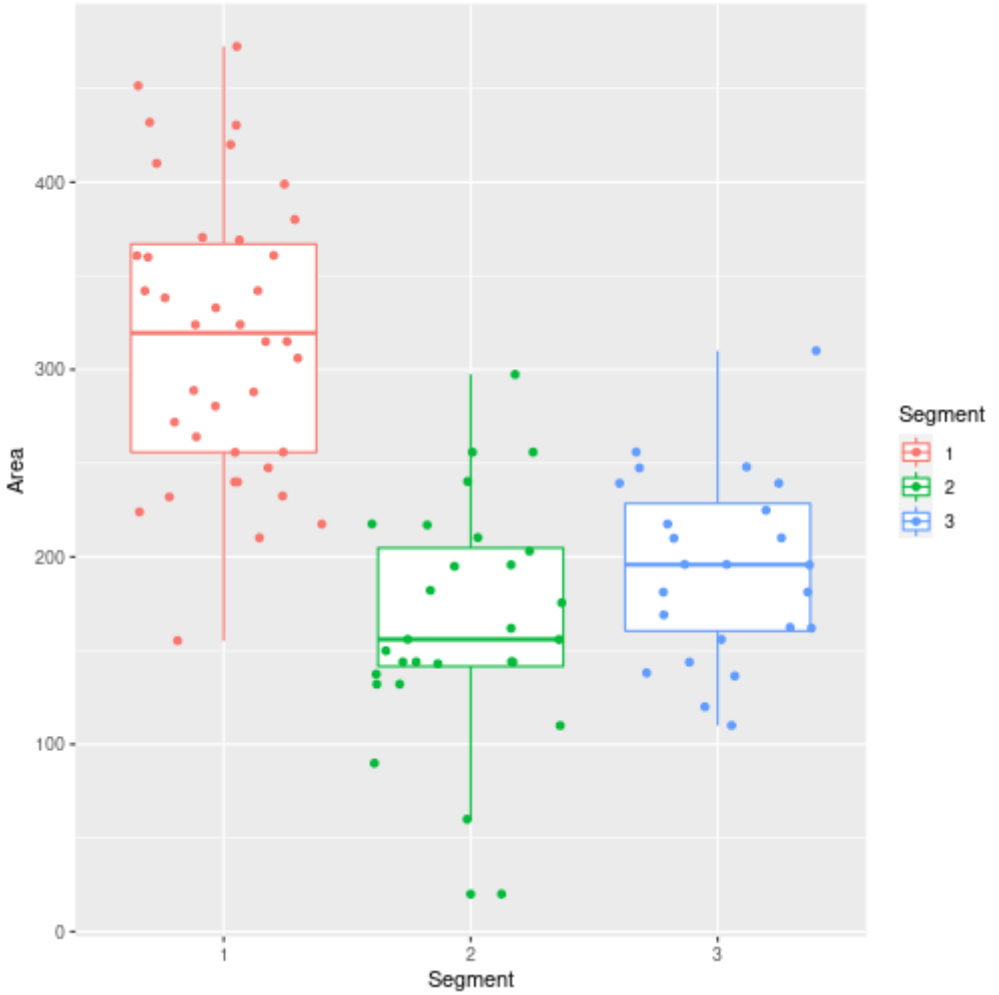
Adding layers

We can instead use `geom_jitter` to add random noise, so we can see all the points.

```
pits_data %>%
  ggplot(aes(x = Segment, y = Area,
             col = Segment)) + # change to co
    geom_boxplot() +
    geom_jitter()
```



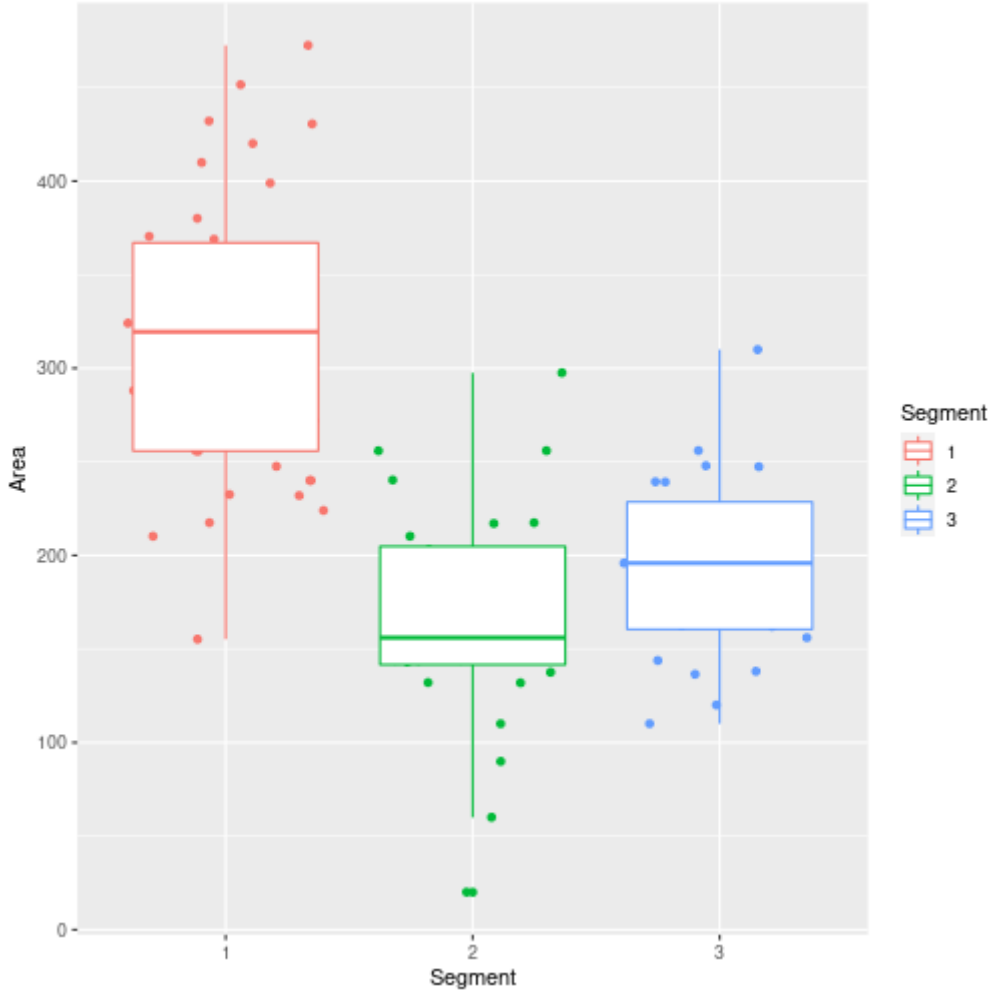

Area
400
300
200
100
0
1
2
3
Segment


Area
400
300
200
100
0
1
2
3
Segment
Segment
1
2
3

Adding layers

☠️ Make sure you add layers in an order that makes sense...

```
pits_data %>%
  ggplot(aes(x = Segment, y = Area,
             col = Segment)) + # or c
    geom_jitter() +
    geom_boxplot()
```

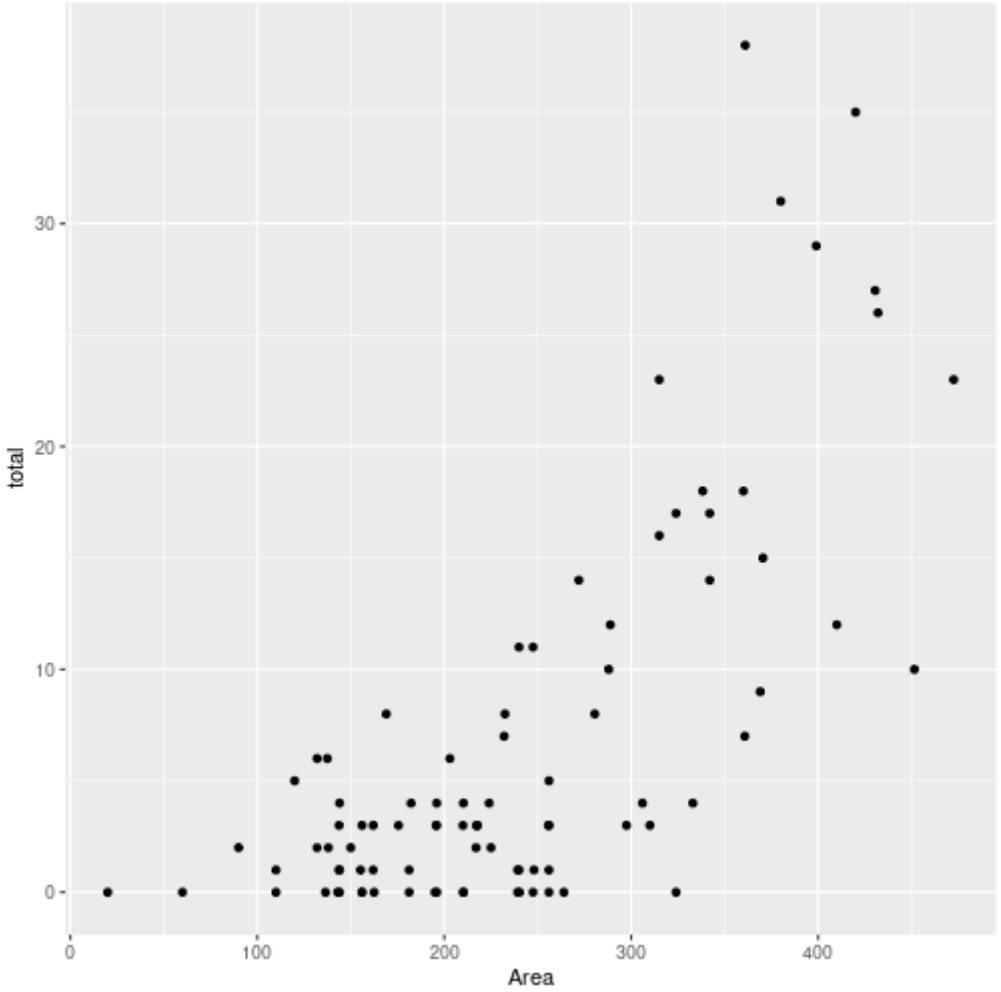


Scatter plots

Good for relationships between numeric variables,

like `total` number of artifacts by house pit `Area`:

```
pits_data %>%
  ggplot(aes(x = Area, y = total)) +
    geom_point() # scatter plot
```



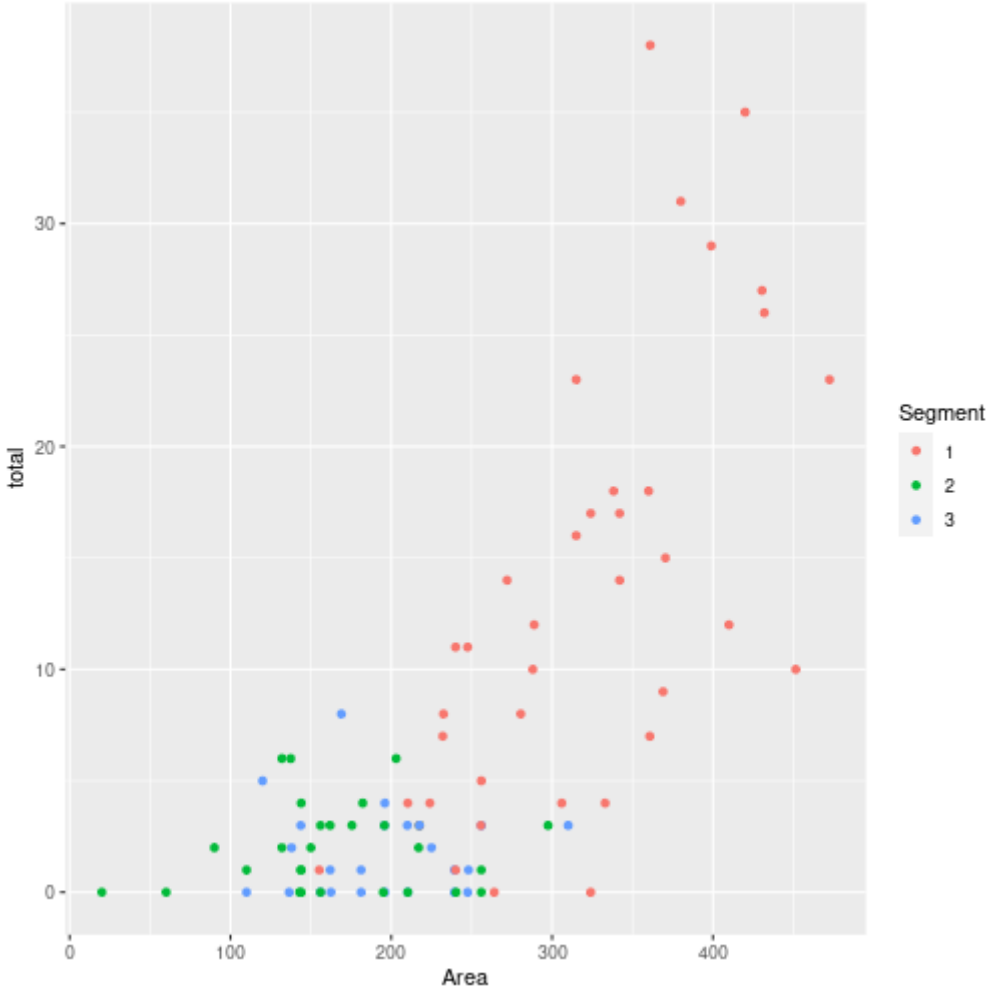
Scatter plots

Good for relationships between numeric variables,

like `total` number of artifacts by house pit `Area`:

Add groups?

```
pits_data %>%
  ggplot(aes(x = Area, y = total,
             col = Segment)) +
    geom_point() # scatter plot
```



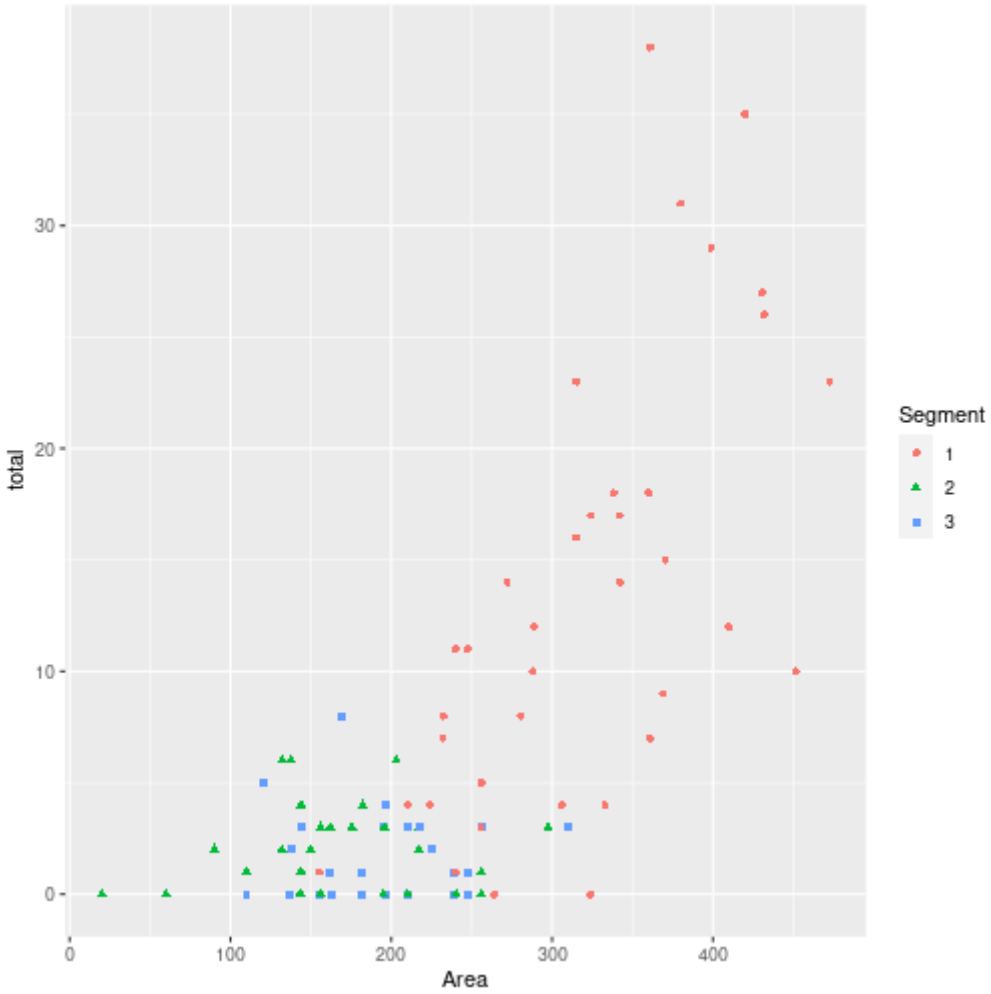
Scatter plots

Printing in black and white?

Colour deficient vision?

```
pits_data %>%
  ggplot(aes(x = Area, y = total,
             col = Segment,
             shape = Segment)) +
    geom_point() # scatter plot
```

Add shape to aes



Scatter plots

Printing in black and white?

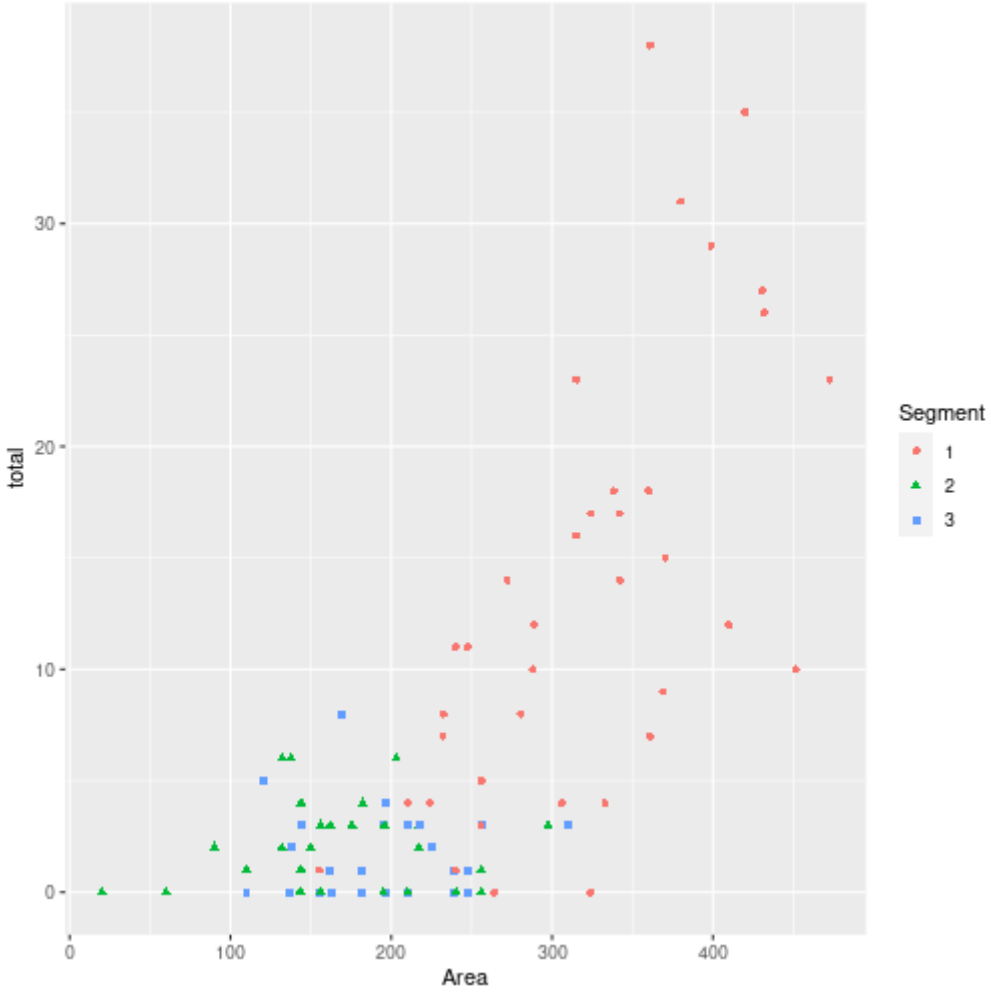
Colour deficient vision?

```
pits_data %>%
  ggplot(aes(x = Area, y = total,
             col = Segment,
             shape = Segment)) +
    geom_point(size = 1.5)
```

Add `shape` to `aes`

and you can increase the `size`

`size` is outside `aes()` because we're not mapping a variable to it



Bar plots

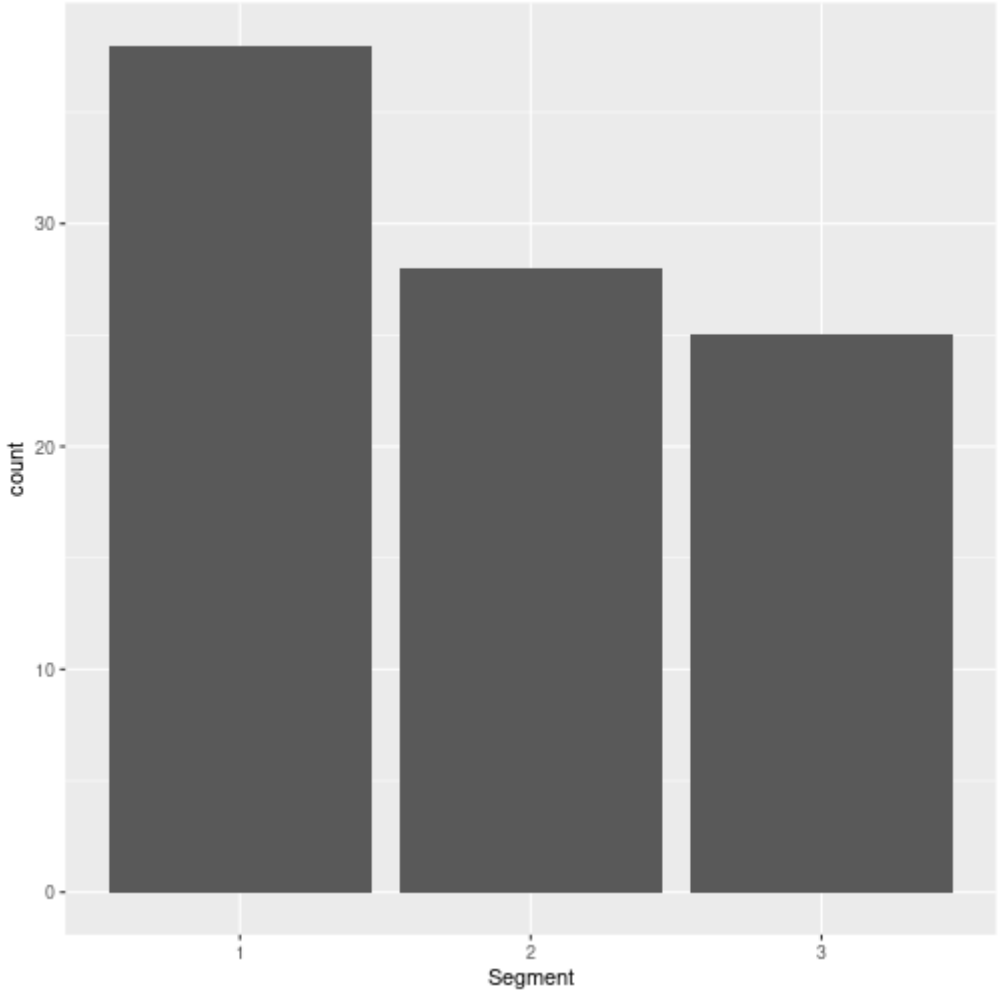
Good for categorical data

```
pits_data %>%
  ggplot(aes(x = Segment)) + # no y-a
  geom_bar()
```

`geom_bar` counts the number of elements within the `Segment` variable,

so here we're seeing number of house pits per segment.

Not particularly interesting...



Bar plots

What about number of `Points` per `Segment`?

We can just add `Points` as the y-axis, right?

```
pits_data %>%
  ggplot(aes(x = Segment, y = Points)
    geom_bar()
```

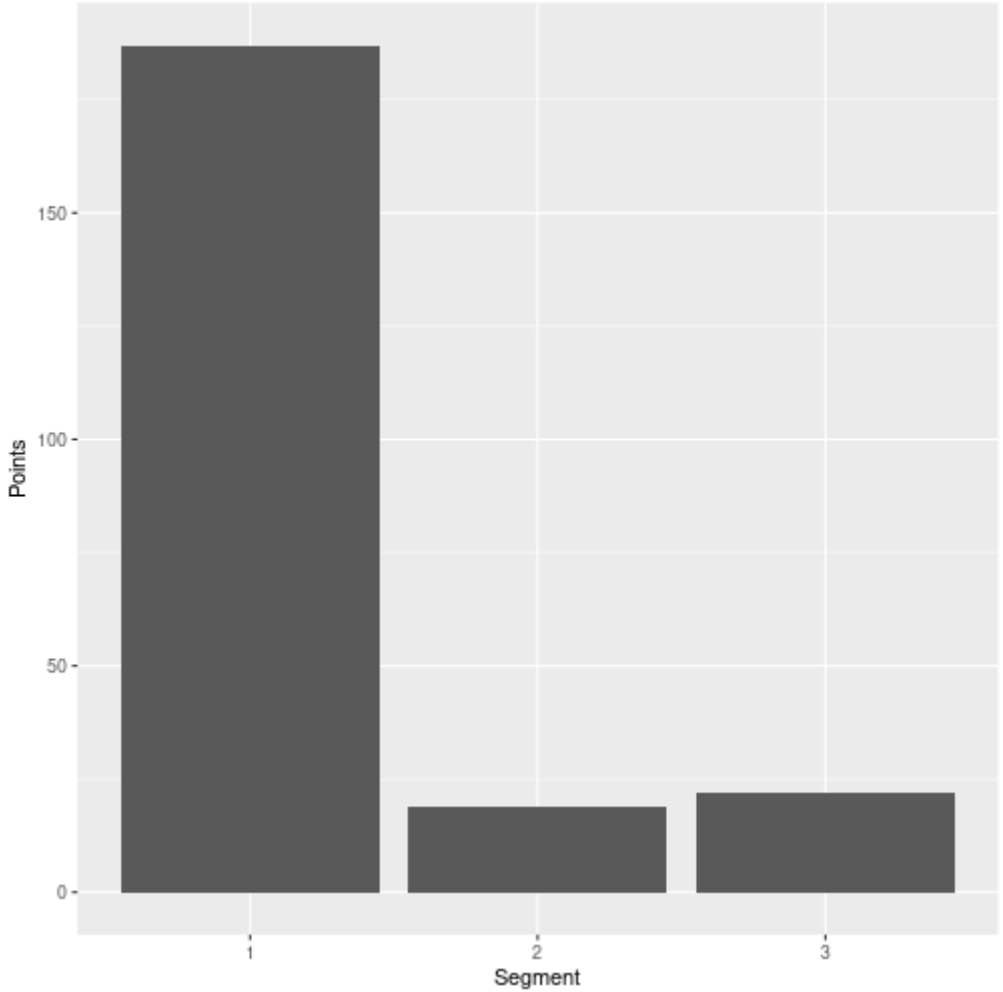
```
## <error: That was of course a leading question>
```

Bar plots

What about number of `Points` per `Segment`?

We need to use `geom_col` if we have a variable on the y-axis.

```
pits_data %>%
  ggplot(aes(x = Segment, y = Points)
    geom_col()
```



Bar plots

What about number of `Ceramics` per segment?

And `Abraders`, and `Discs`, and `Earplugs`, etc.

We need a way to easily separate by artifacts.

Luckily we already have a data frame that could be useful:

```
pits_data_long
```

```
## # A tibble: 637 × 9
##     East South Length Width Segment Location  Area artifact count
##    <dbl> <dbl>  <dbl> <dbl>   <dbl> <chr>    <dbl> <chr>    <dbl>
##  1  901.  75.1     12    12       2 Outside    144 Points       0
##  2  901.  75.1     12    12       2 Outside    144 Abraders     1
##  3  901.  75.1     12    12       2 Outside    144 Discs        0
##  4  901.  75.1     12    12       2 Outside    144 Earplugs     0
##  5  901.  75.1     12    12       2 Outside    144 Effigies     0
##  6  901.  75.1     12    12       2 Outside    144 Ceramics     0
##  7  901.  75.1     12    12       2 Outside    144 total        1
##  8  973.  81.3     16    16       2 Outside    256 Points       0
##  9  973.  81.3     16    16       2 Outside    256 Abraders     0
## 10  973.  81.3     16    16       2 Outside    256 Discs        0
## # … with 627 more rows
```

We just need to get rid of the `total` values:

```
pits_data_long <- pits_data_long %>%
  filter(artifact != "total")
```

Because we don't want to include them in the `count`

(think about what would happen if we used `sum` on the `count` variable)

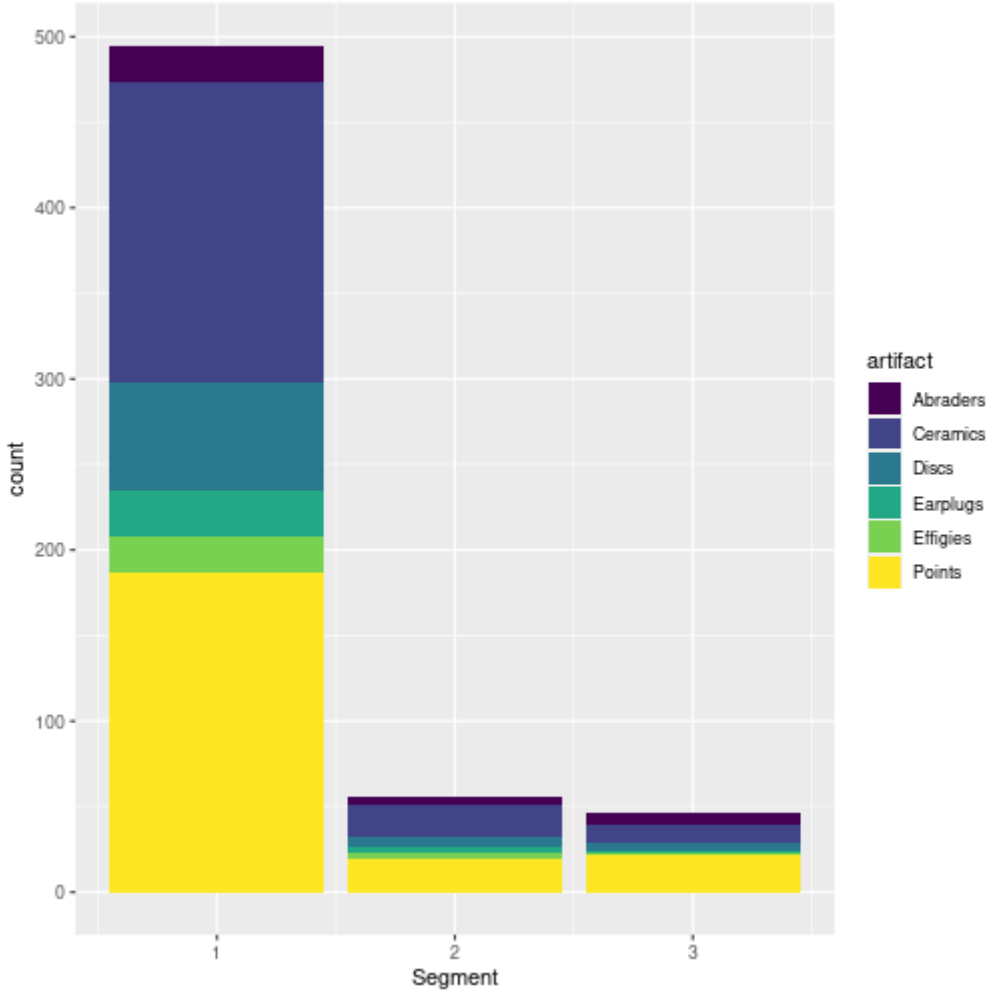
Visualising with long data

Total artifacts per `Segment`.

```
pits_data_long %>%
  ggplot(aes(x = Segment, y = count,
             fill = artifact)) +
    geom_col() +
    scale_fill_viridis_d()
```

Great! But we already know Segment 1 has more artifacts,

and it's hard to see what's going on in 2 and 3.



Visualising with long data

Total artifacts per `Segment`.

```
pits_data_long %>%
  ggplot(aes(x = Segment, y = count,
             fill = artifact)) +
    geom_col() +
    scale_fill_viridis_d()
```

Great! But we already know Segment 1 has more artifacts,

and it's hard to see what's going on in 2 and 3.

Relative count bar plot

To solve this issue,

we can calculate relative counts:

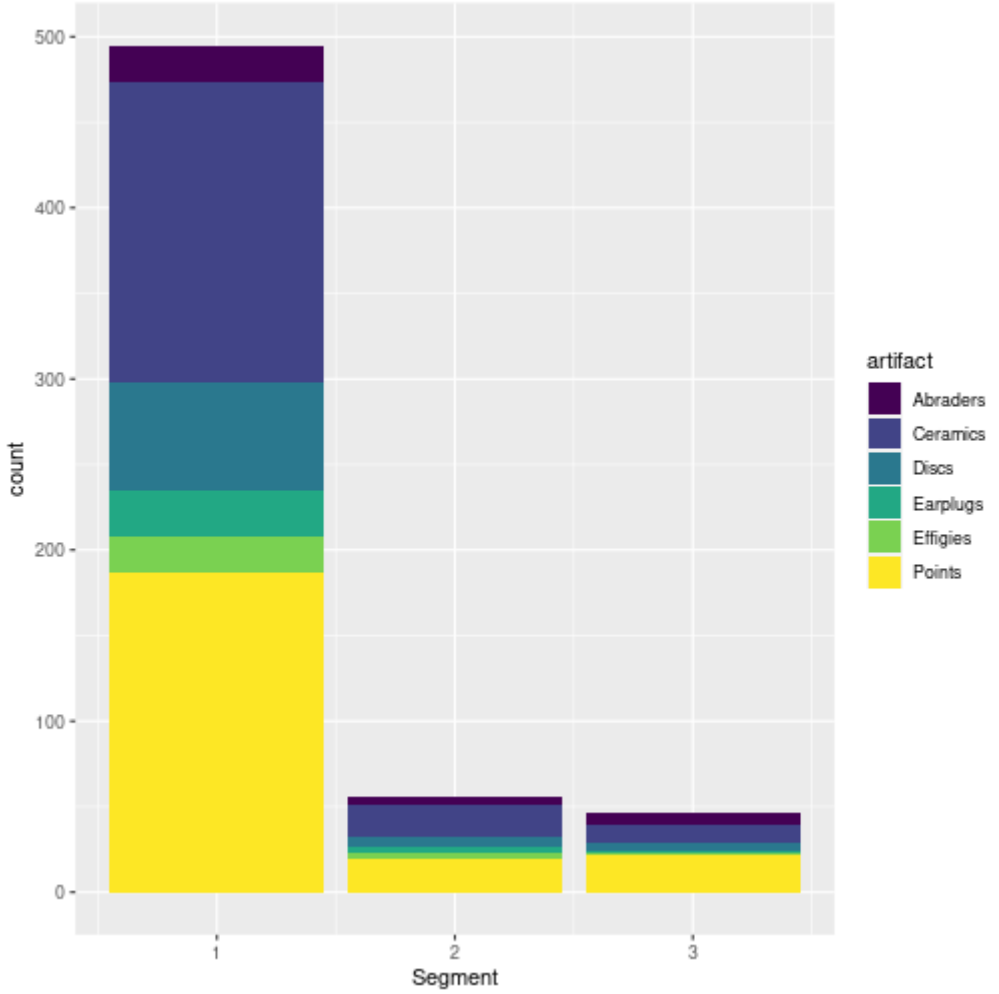
```
pits_data_long %>%
  group_by(Segment, artifact) %>%
    summarise(count = sum(count, na.rm = T)) %
    mutate(percent = count / sum(count) * 100)
```

```
## # A tibble: 18 × 4
## # Groups:   Segment [3]
##    Segment artifact count percent
##      <dbl> <chr>    <dbl>   <dbl>
##  1       1 Abraders    21    4.24
##  2       1 Ceramics   176   35.6
##  3       1 Discs       63   12.7
##  4       1 Earplugs    27    5.45
##  5       1 Effigies    21    4.24
##  6       1 Points     187   37.8
##  7       2 Abraders     5    8.93
##  8       2 Ceramics    19   33.9
##  9       2 Discs        5    8.93
## 10       2 Earplugs     4    7.14
## 11       2 Effigies     4    7.14
## 12       2 Points      19   33.9
## 13       3 Abraders     6   13.0
## 14       3 Ceramics    11   23.9
## 15       3 Discs        5   10.9
## 16       3 Earplugs     1    2.17
## 17       3 Effigies     1    2.17
## 18       3 Points      22   47.8
```

Relative count bar plot

we can calculate relative counts:

```
pits_data_long %>%
  ggplot(aes(x = Segment, y = count,
             fill = artifact)) +
    geom_col() +
    scale_fill_viridis_d()
```



Relative count bar plot

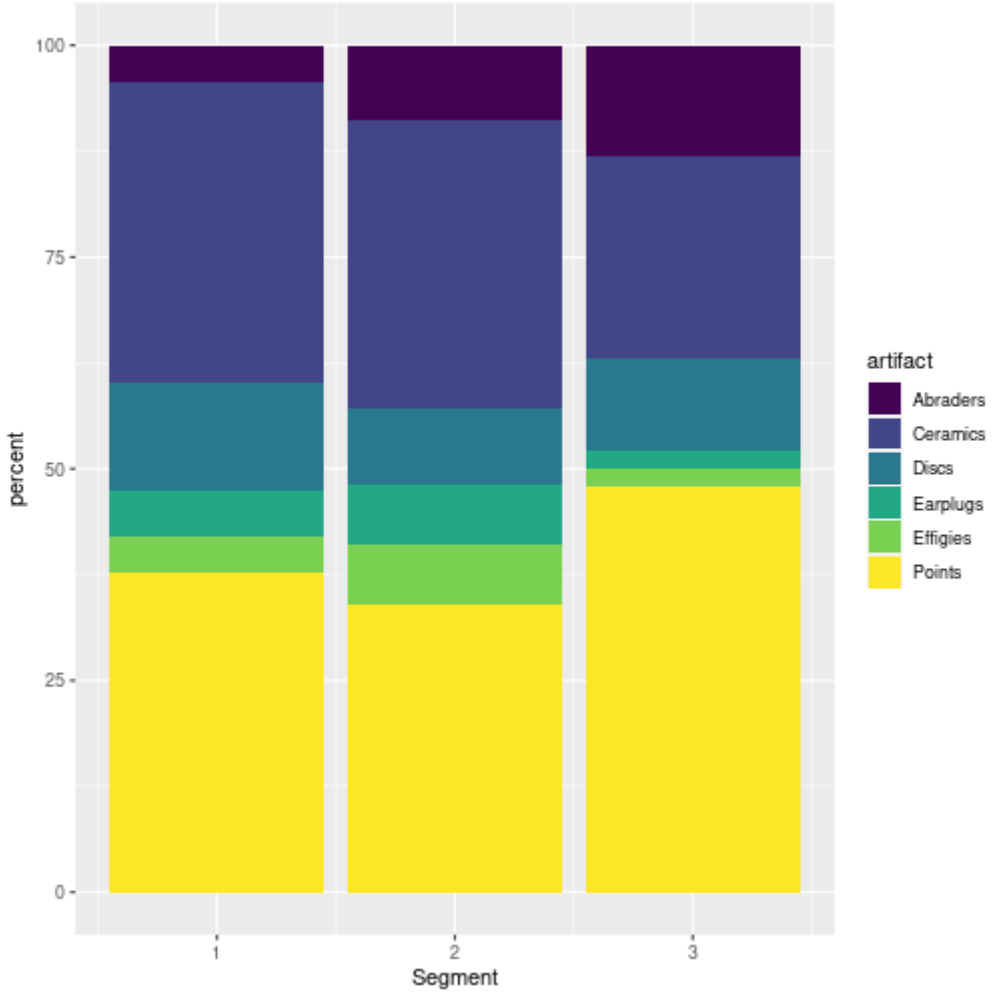
And place it before the code for our plot:

```
pits_data_long %>%
  group_by(Segment, artifact) %>%
    summarise(count = sum(count, na.r
    mutate(percent = count / sum(cour
  ggplot(aes(x = Segment, y = percent
             fill = artifact)) +
    geom_col() +
    scale_fill_viridis_d()
```

Interesting…

There are clear differences in the absolute number of artifacts between Segments,

but the relative numbers are similar.

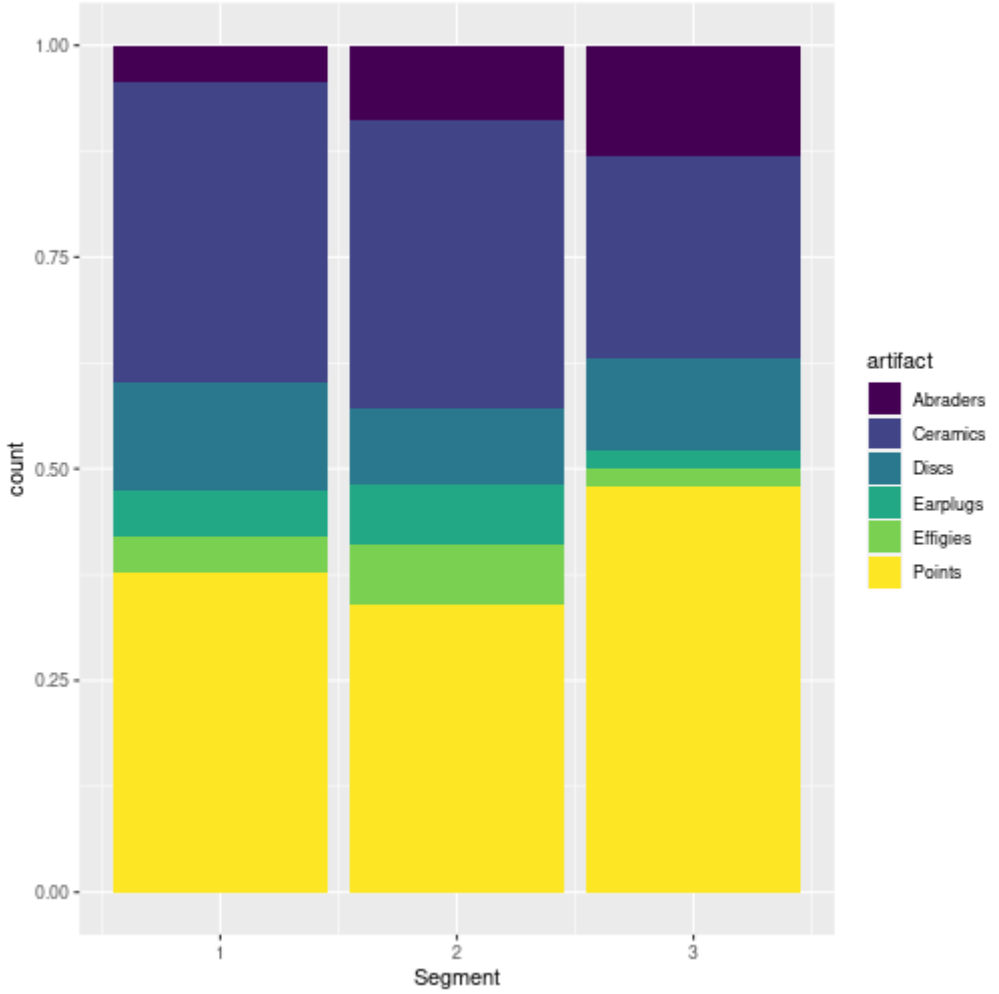


Relative count bar plot

We can also use `geom_col(position = "fill")` to get relative counts,

although with the previous method we could get a percentage.

```
pits_data_long %>%
  ggplot(aes(x = Segment, y = count,
             fill = artifact)) +
    geom_col(position = "fill") +
    scale_fill_viridis_d()
```



Exercise

Use `geom_col(position = "dodge")` instead of `"fill"`.

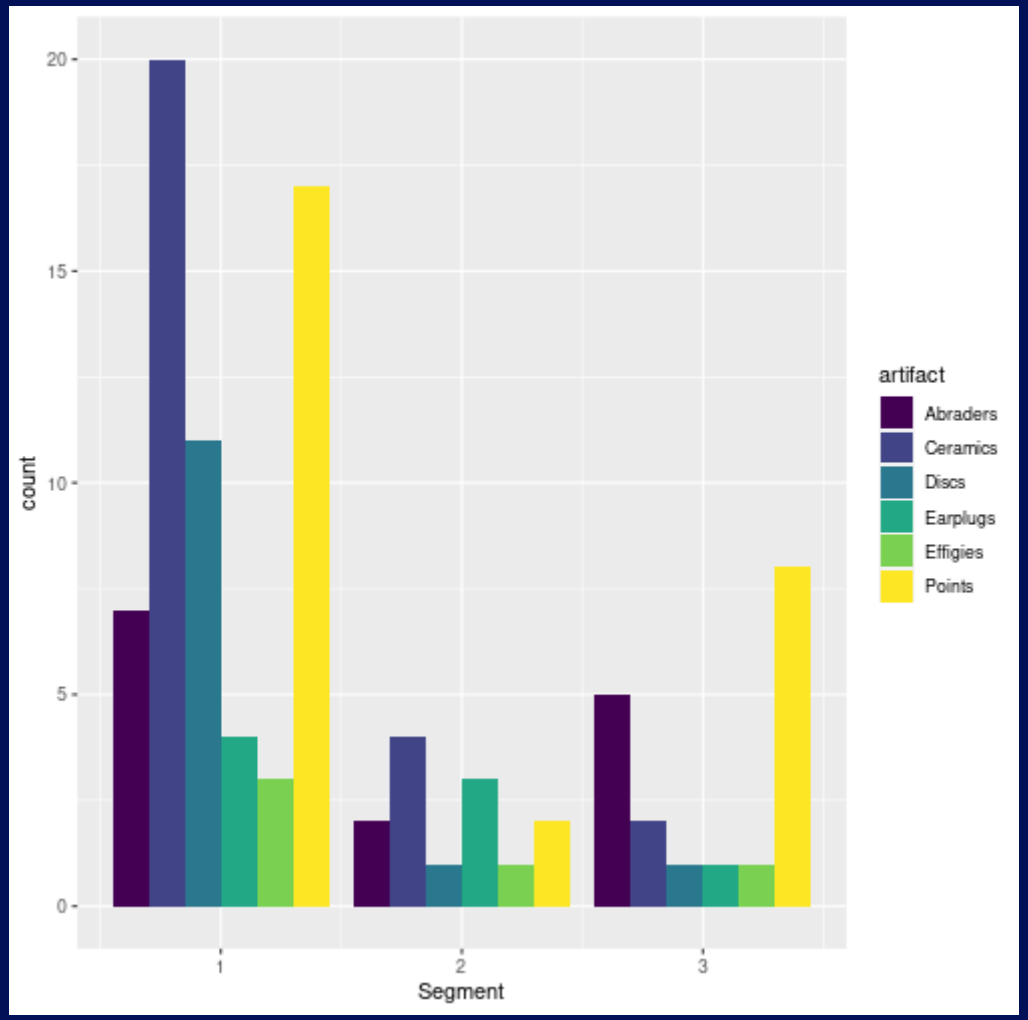
What are we seeing?
Compare to the bar with absolute counts. Is something off?

Solution

`position = "dodge"` puts `artifacts` side-by-side instead of stacking.

The problem is that it can't deal with multiple `count` values (don't ask me why...)

```
pits_data_long %>%
  ggplot(aes(x = Segment, y = count,
             fill = artifact)) +
    geom_col(position = "dodge") + #
    scale_fill_viridis_d()
```



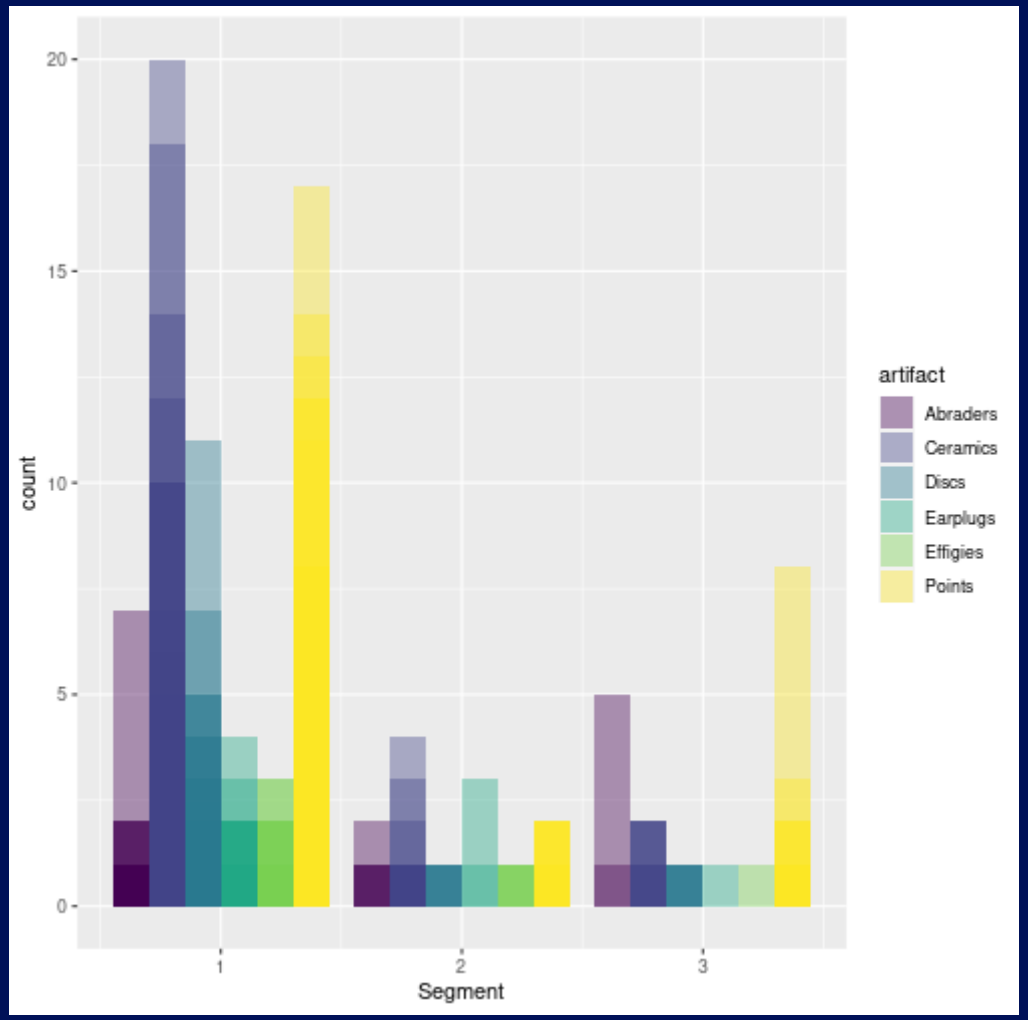
Solution

`position = "dodge"` puts `artifacts` side-by-side instead of stacking.

The problem is that it can't deal with multiple `count` values (don't ask me why...)

So it superimposes multiple columns of the same artifact types:

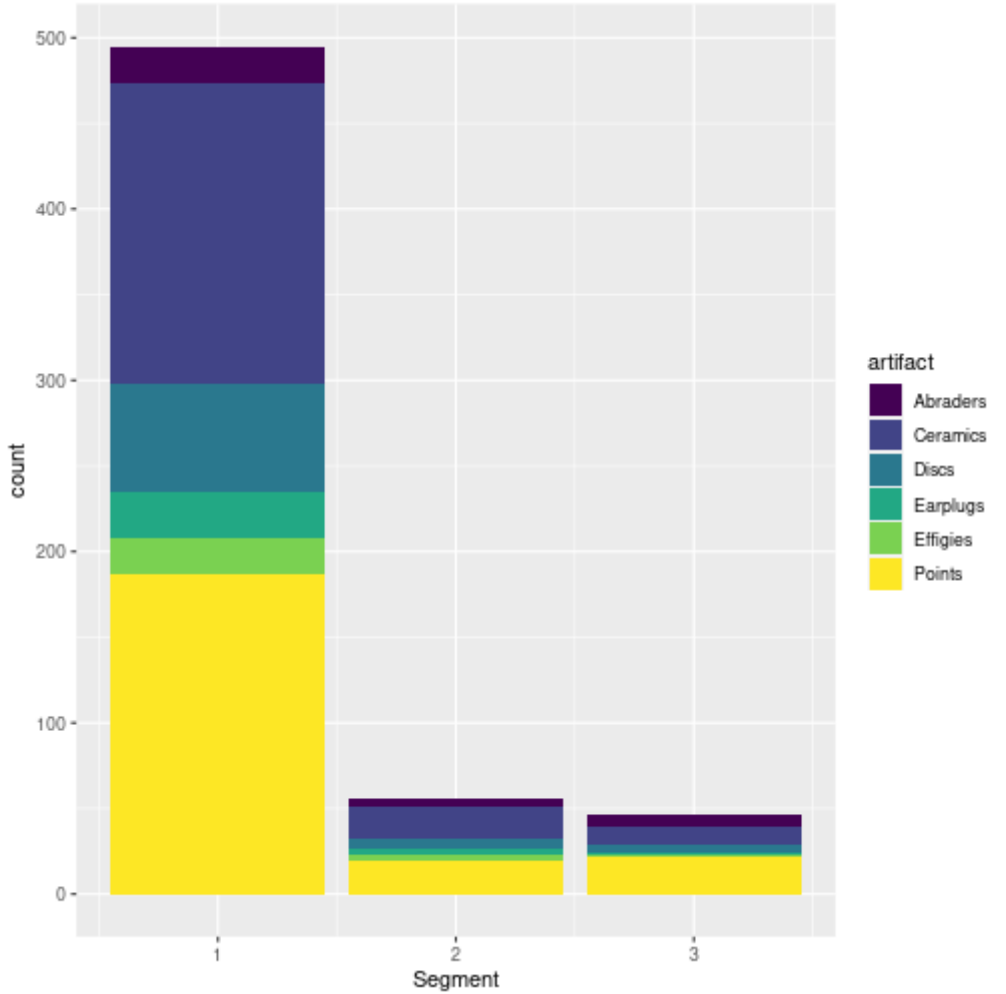
```
pits_data_long %>%
  ggplot(aes(x = Segment, y = count,
             fill = artifact)) +
    geom_col(position = "dodge", alph
    scale_fill_viridis_d()
```

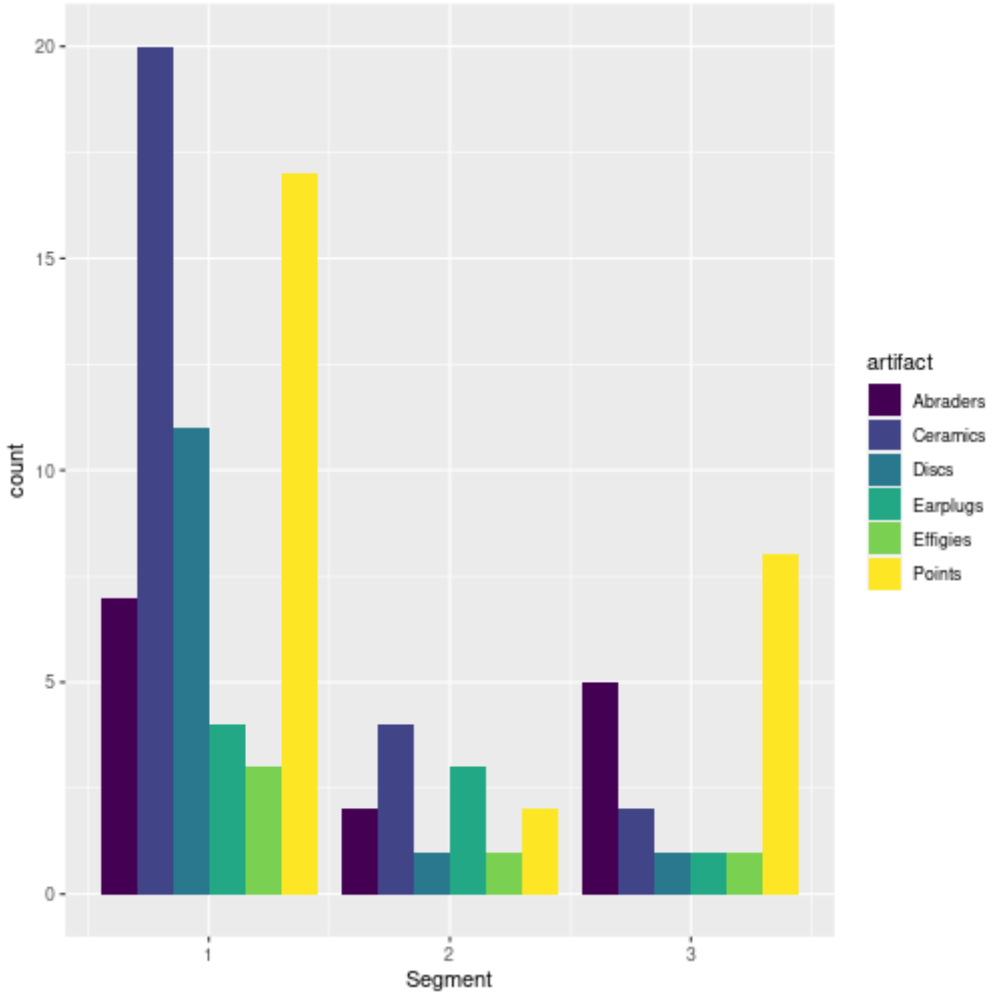


Cautionary tale

Check the numbers in your data frame!

We could already see there was an issue with the *count* when comparing plots





And a quick summary of artifacts could show us which plot was off:

```
pits_data_long %>%
  group_by(Location, artifact) %>%
  summarise(count = sum(count, na.rm = T))
```

```
## # A tibble: 12 × 3
## # Groups:   Location [2]
##    Location artifact count
##    <chr>    <chr>    <dbl>
##  1 Inside   Abraders    21
##  2 Inside   Ceramics   176
##  3 Inside   Discs       63
##  4 Inside   Earplugs    27
##  5 Inside   Effigies    21
##  6 Inside   Points     187
##  7 Outside  Abraders    11
##  8 Outside  Ceramics    30
##  9 Outside  Discs       10
## 10 Outside  Earplugs     5
## 11 Outside  Effigies     5
## 12 Outside  Points      41
```

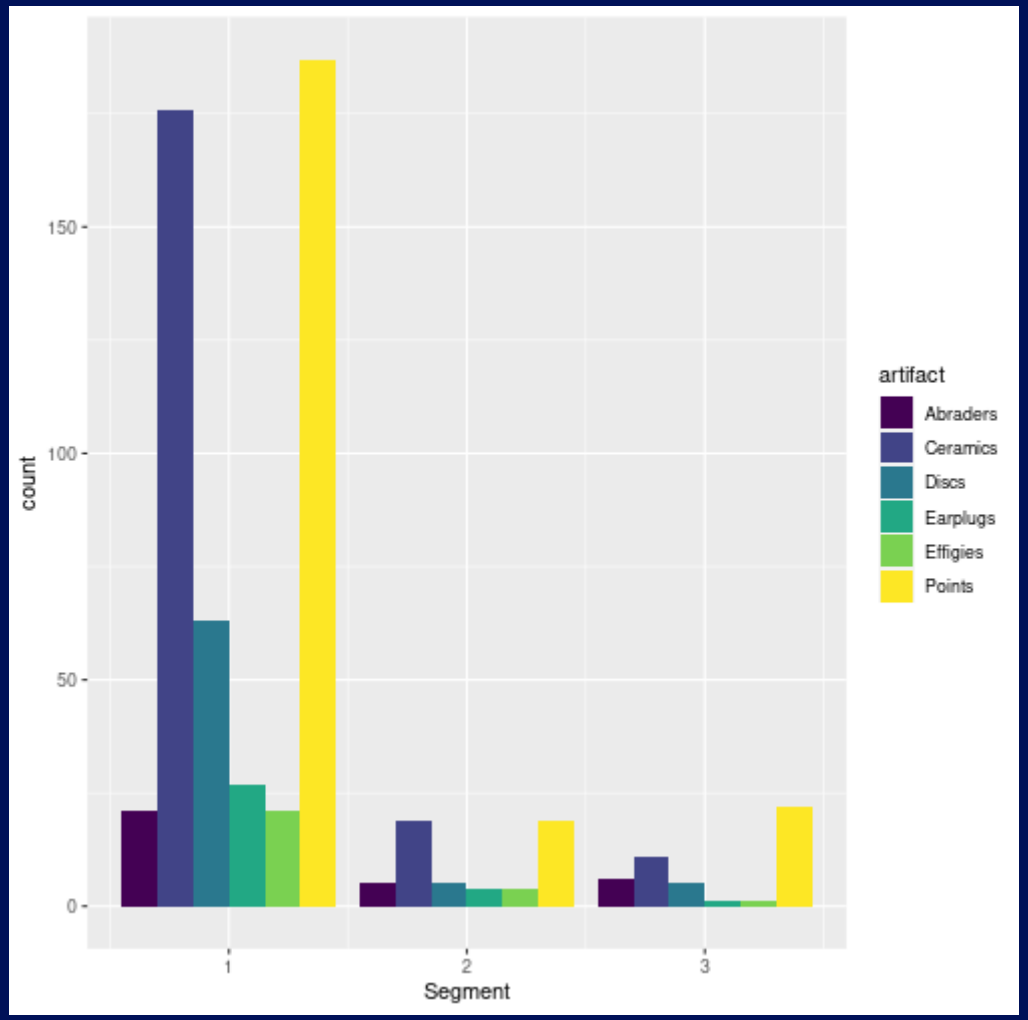
Exercise

Find a way to use `geom_col(position = "dodge")`

💡 **Hint:** Create a summary data frame like we did for the relative bar plot.

Solution

```
pits_data_long %>%
  group_by(Segment, artifact) %>%
  summarise(count = sum(count, na.rm
  ggplot(aes(x = Segment, y = count,
    geom_col(position = "dodge") +
    scale_fill_viridis_d()
```

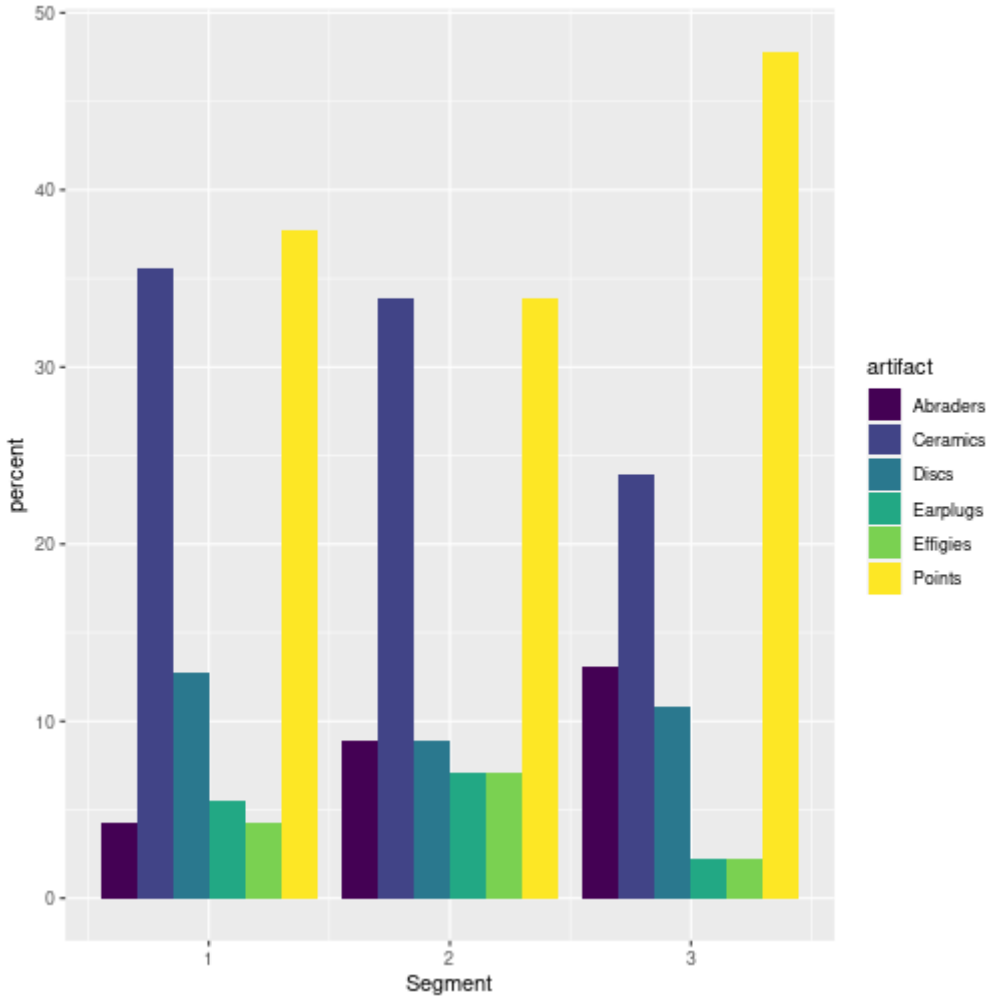


Prettification

Now let's make it look a little nicer!

Starting with relative counts:

```
pits_data_long %>%
  group_by(Segment, artifact) %>%
  summarise(count = sum(count, na.rm
  mutate(percent = count / sum(count)
  ggplot(aes(x = Segment, y = percent
    geom_col(position = "dodge") +
    scale_fill_viridis_d()
```



Prettification

To make life easier, we can store the current plot

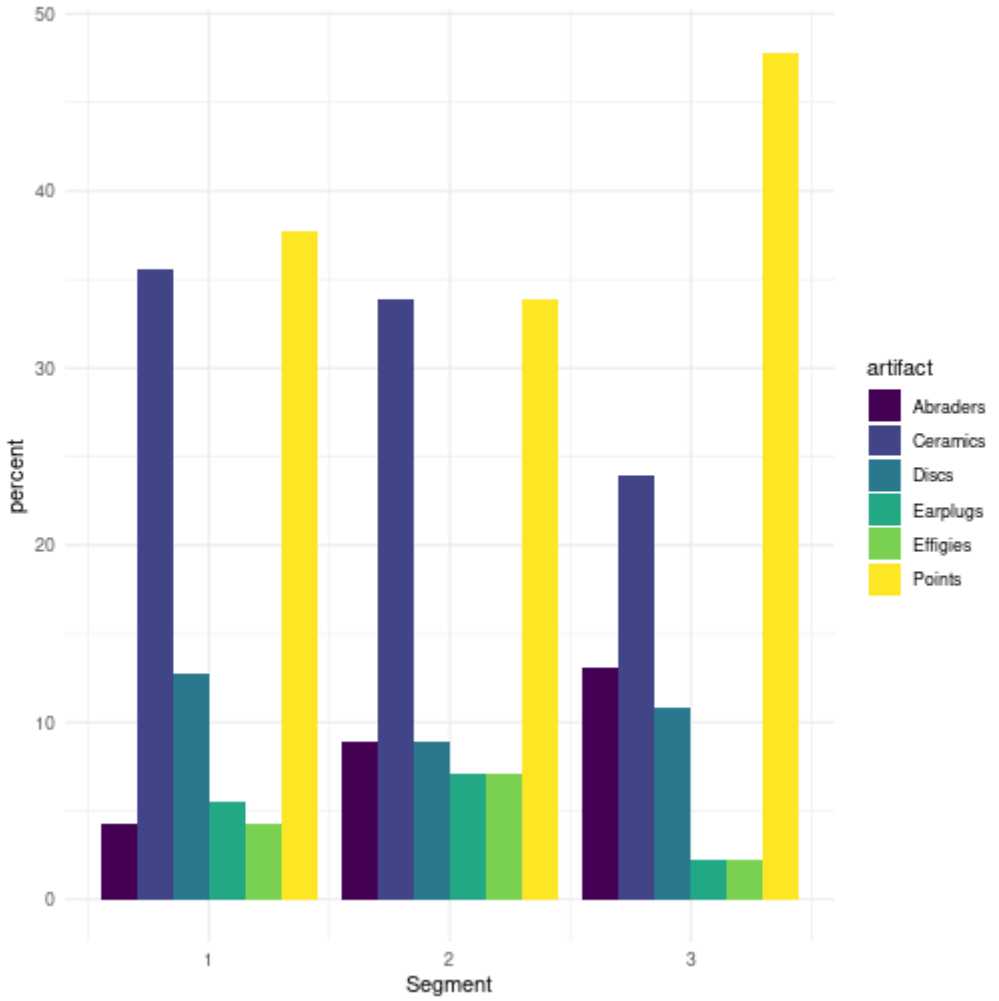
```
my_little_plot <- pits_data_long %>%
  group_by(Segment, artifact) %>%
  summarise(count = sum(count, na.rm = T)) %>%
  mutate(percent = count / sum(count) * 100) %>%
  ggplot(aes(x = Segment, y = percent, fill = artifact)) +
    geom_col(position = "dodge") +
    scale_fill_viridis_d()
```

And start customising

Prettification

Starting with the background:

```
my_little_plot +
  theme_minimal()
```

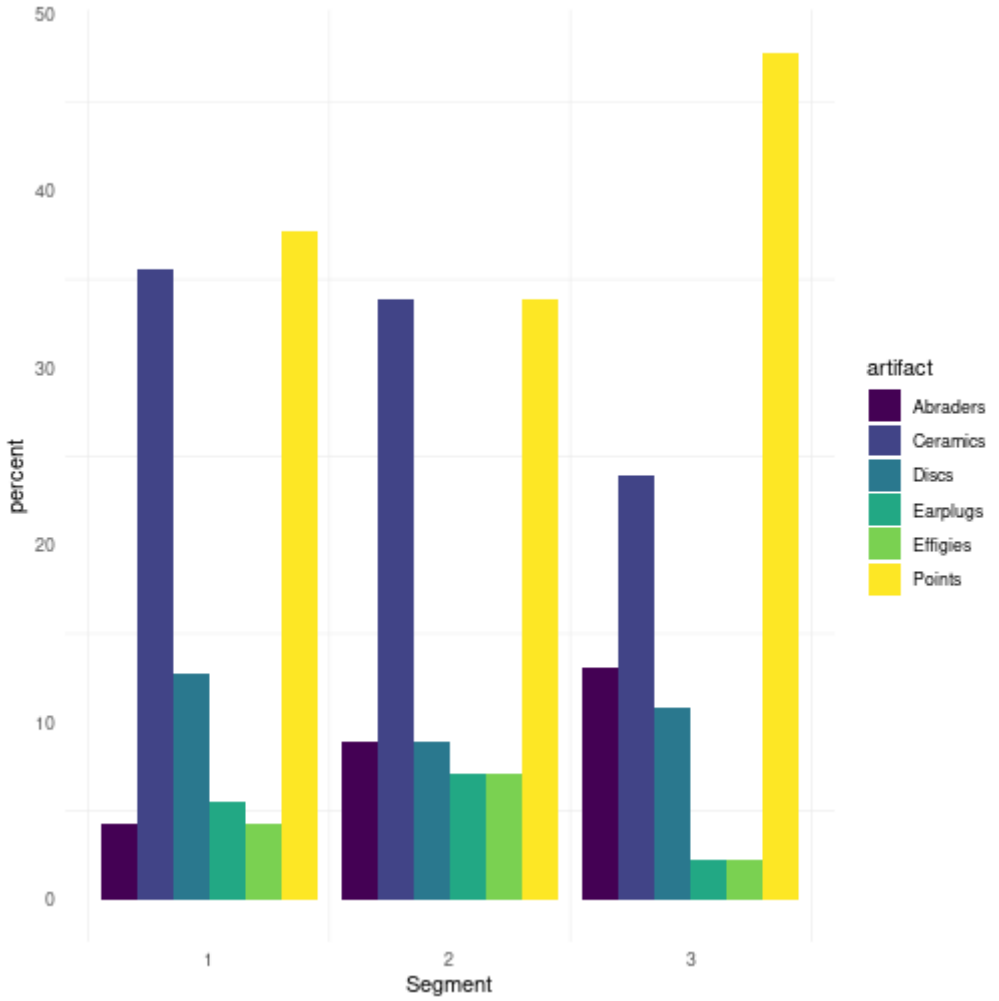


Prettification

We don't really need the vertical lines

```
my_little_plot +
  theme_minimal() +
  theme(panel.grid.major.x = element_
```

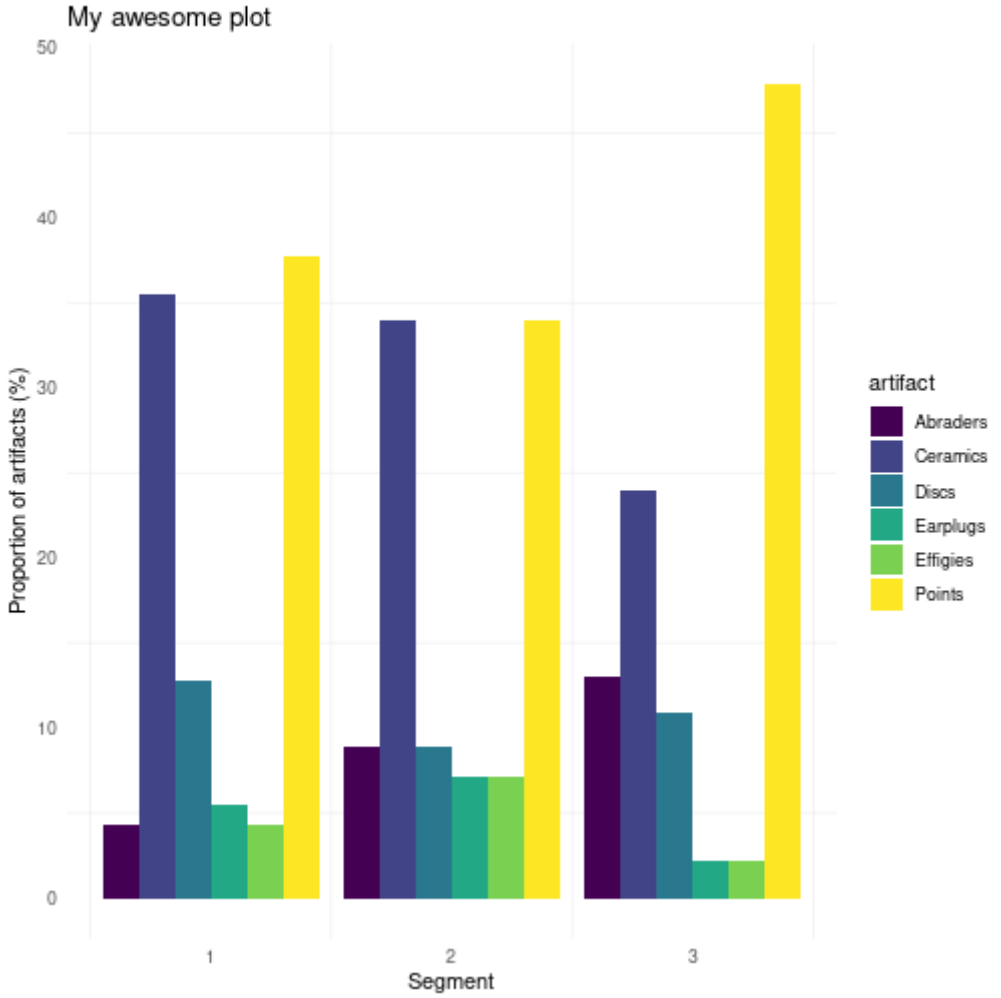
`element_blank()` removes objects



Prettification

Let's fix the labels

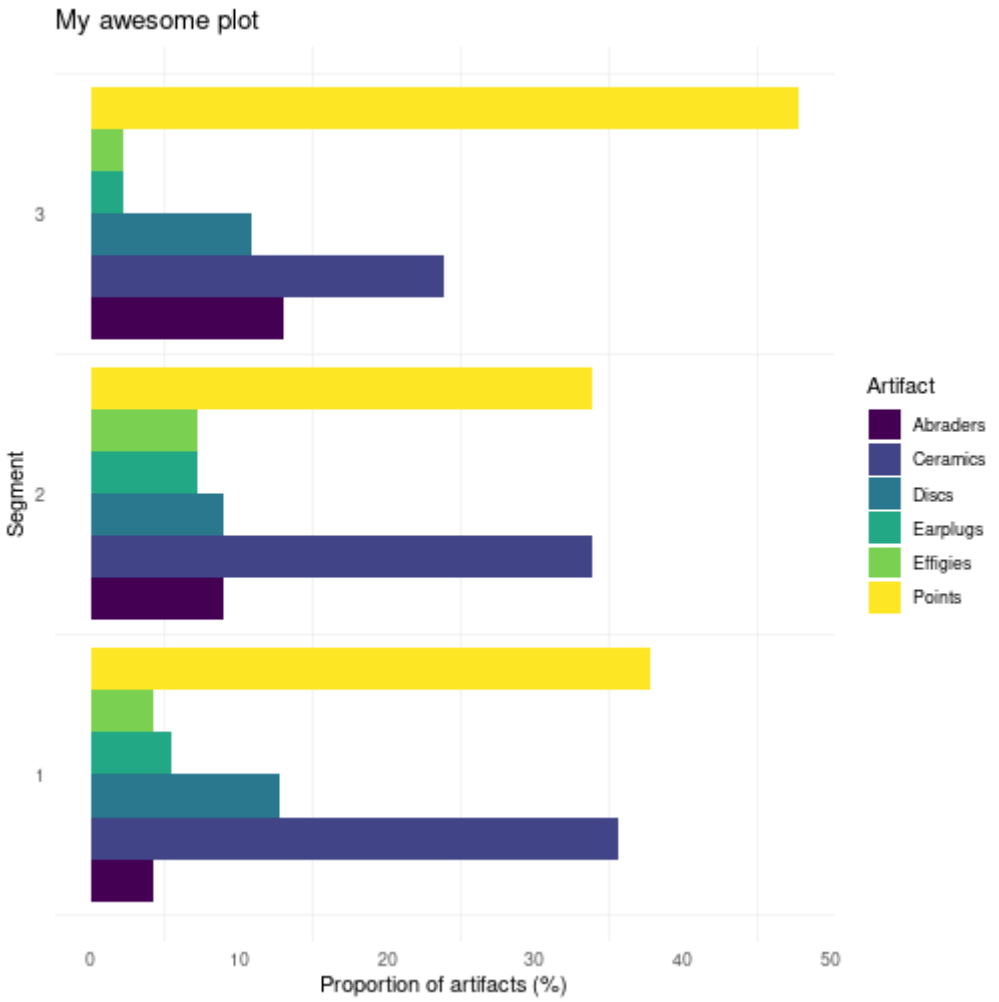
```
my_little_plot +
  theme_minimal() +
  theme(panel.grid.major = element_bl
  labs(y = "Proportion of artifacts (
      artifact = "Artifact",
      title = "My awesome plot")
```



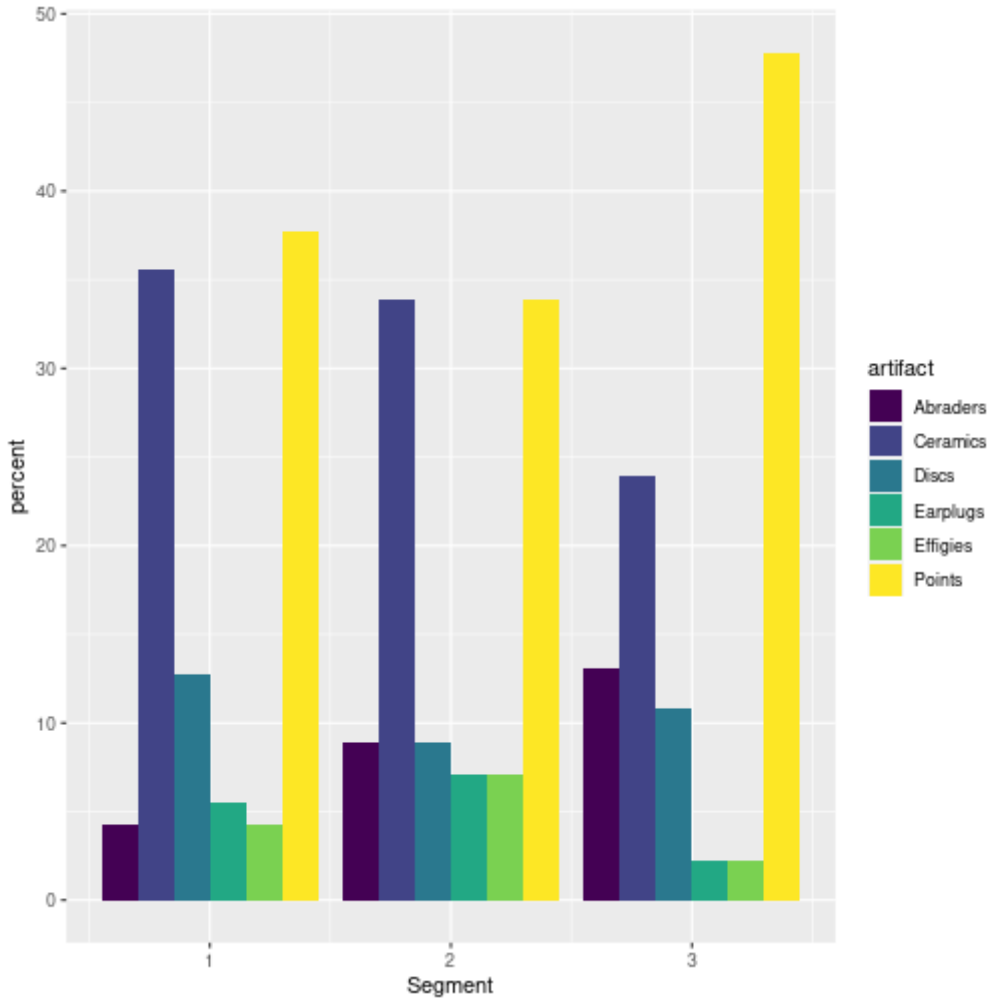
Prettification

Finally, flipping the axes increases readability (in my opinion)

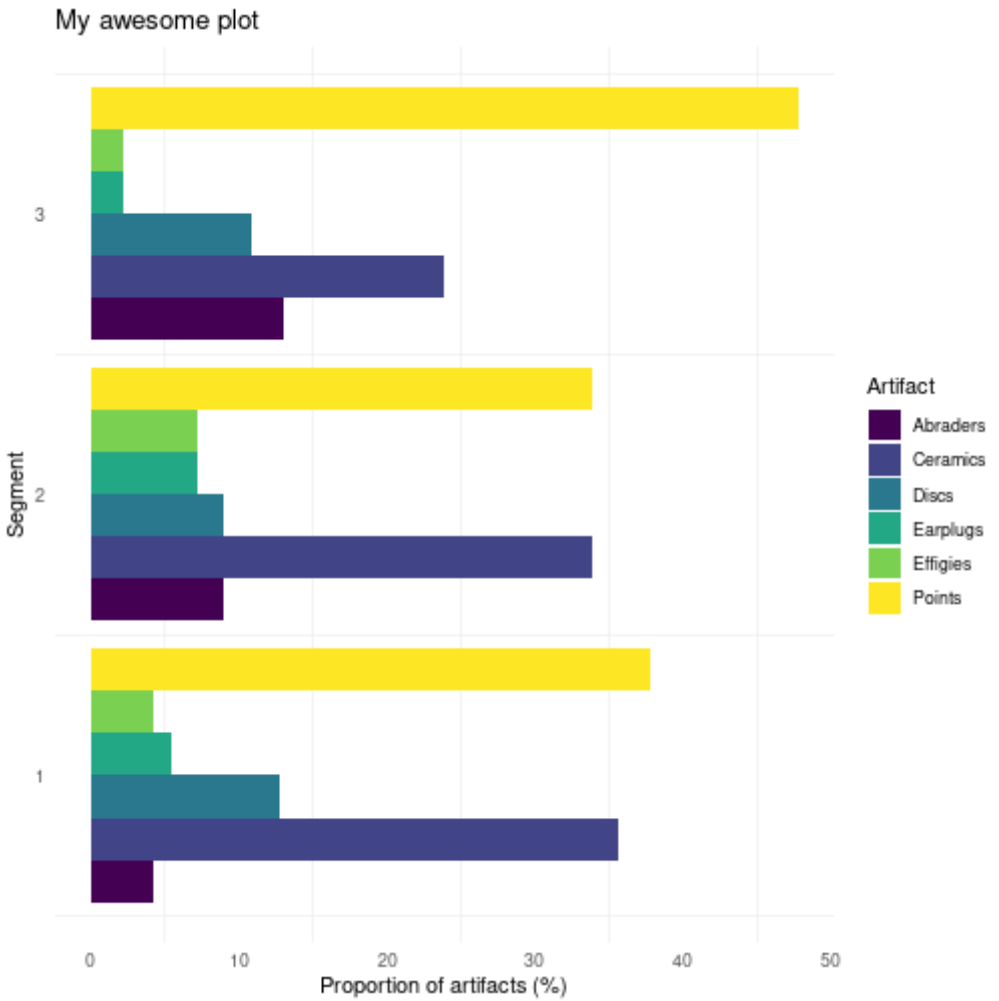
```
my_little_plot +
  theme_minimal() +
  theme(panel.grid.major.x = element_
  labs(y = "Proportion of artifacts (
    fill = "Artifact",
    title = "My awesome plot") +
  coord_flip()
```



Before



After




PIZZAZZ, YEAH.