Day 1

Caio Kenup

2023-07-13

A `ggplot` primer

To get started, we need to install and load the `ggplot2` package. Then we download a sample dataset (in this case, a dataset of beer reviews)

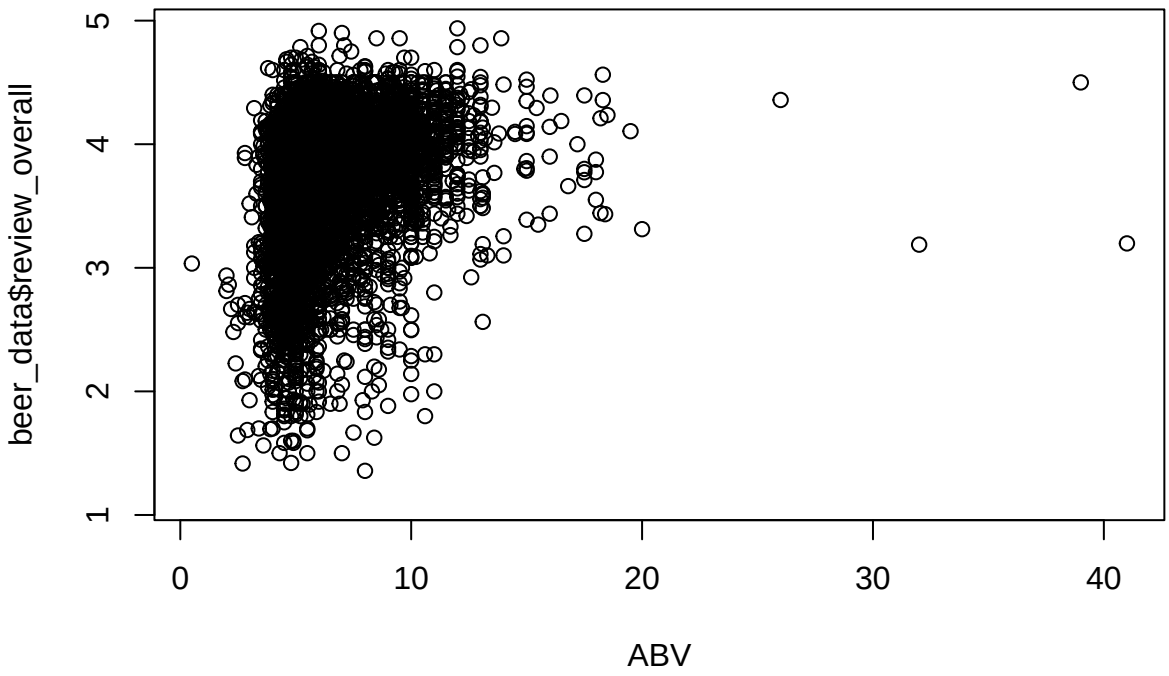
```
require(ggplot2)
require(dplyr)


url_data <- "https://raw.githubusercontent.com/KenupCF/ZSL_DataClub/main/Day%201/data/beer_reviews.csv"

url_connection<-url(url_data)

beer_data<-read.csv(url_connection)
```

Simple Scatterplot

‘

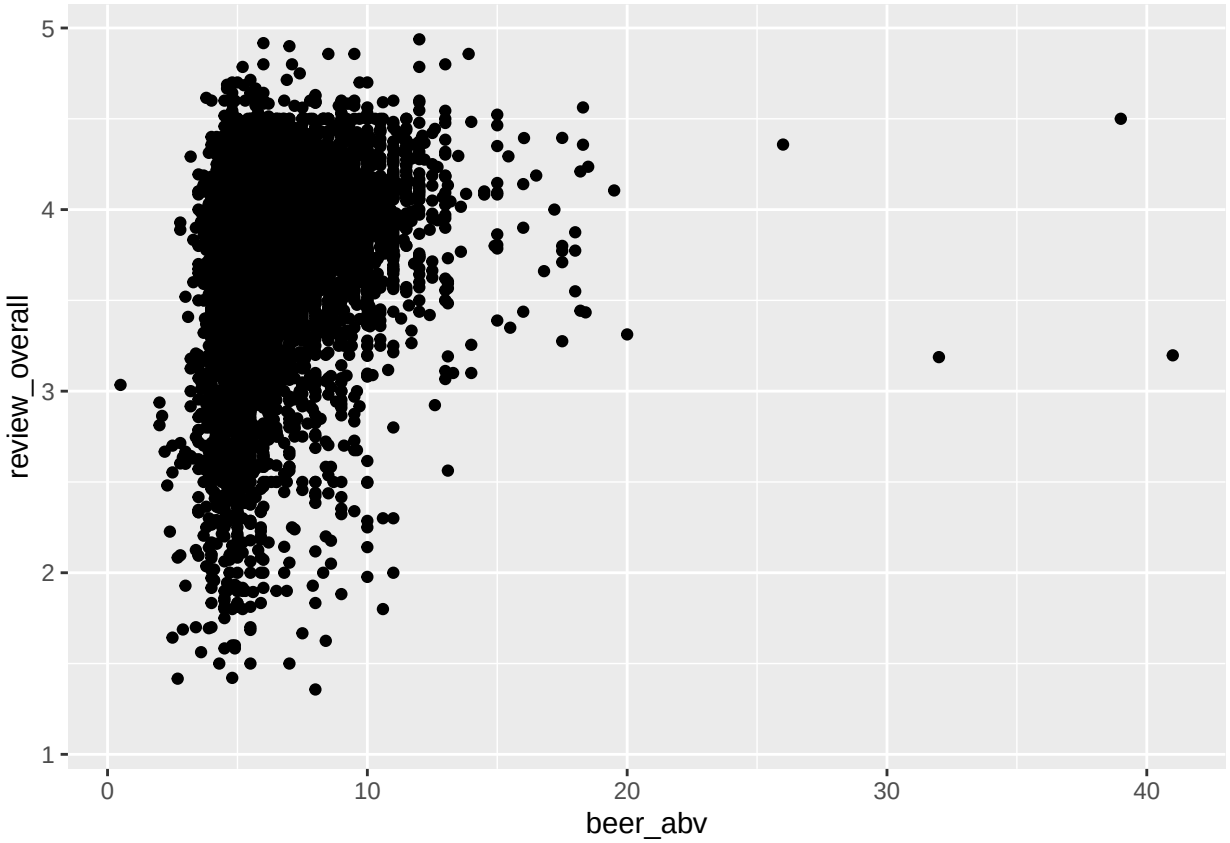
This is how you would do a scatterplot in base R

```
plot(y=beer_data$review_overall,
  x=beer_data$beer_abv,
  xlab="ABV")
```



And this is how you have the same scatterplot in ggplot2

```
ggplot(data=beer_data,
       mapping = aes(x = beer_abv, y= review_overall))+
      geom_point()
```



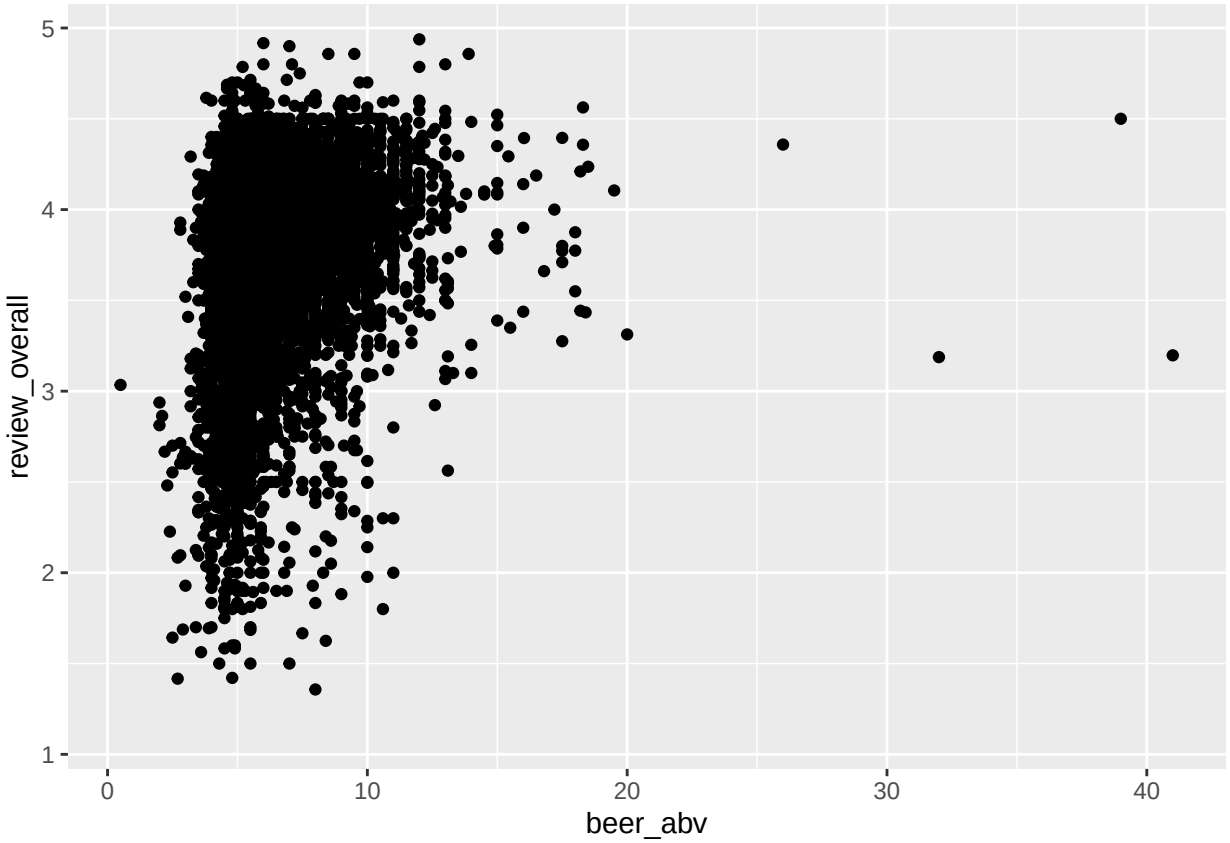
`geom_point()` is a function o the *geometry* family - these are the functions that "draw" stuff on your plot. Some of them are

- `geom_point()` - easy, plot points based on x and y coordinates
- `geom_line()` - similar to `geom_point()`, but connects those coordinates with lines
- `geom_bar()` - plot bars - quite different, there are options to calculate summary statistics and produce grouped or stacked bars.
- `geom_ribbon()` - plot areas in a plot, defined by a band (ymin and ymax)
- `geom_smooth()` - produces smoothed patterns in the data, using different methods

Each geometry has *required* and *accepted* aesthetics. Required aesthetics are plotting information necessary to plot the geometry (e.g. x and y coordinates for `geom_point()`). In general geometries require aesthetics that relate to where

Notice that the aesthetics can be defined on the main `ggplot()` function (as done previously), or in the specific `geom_*()`:

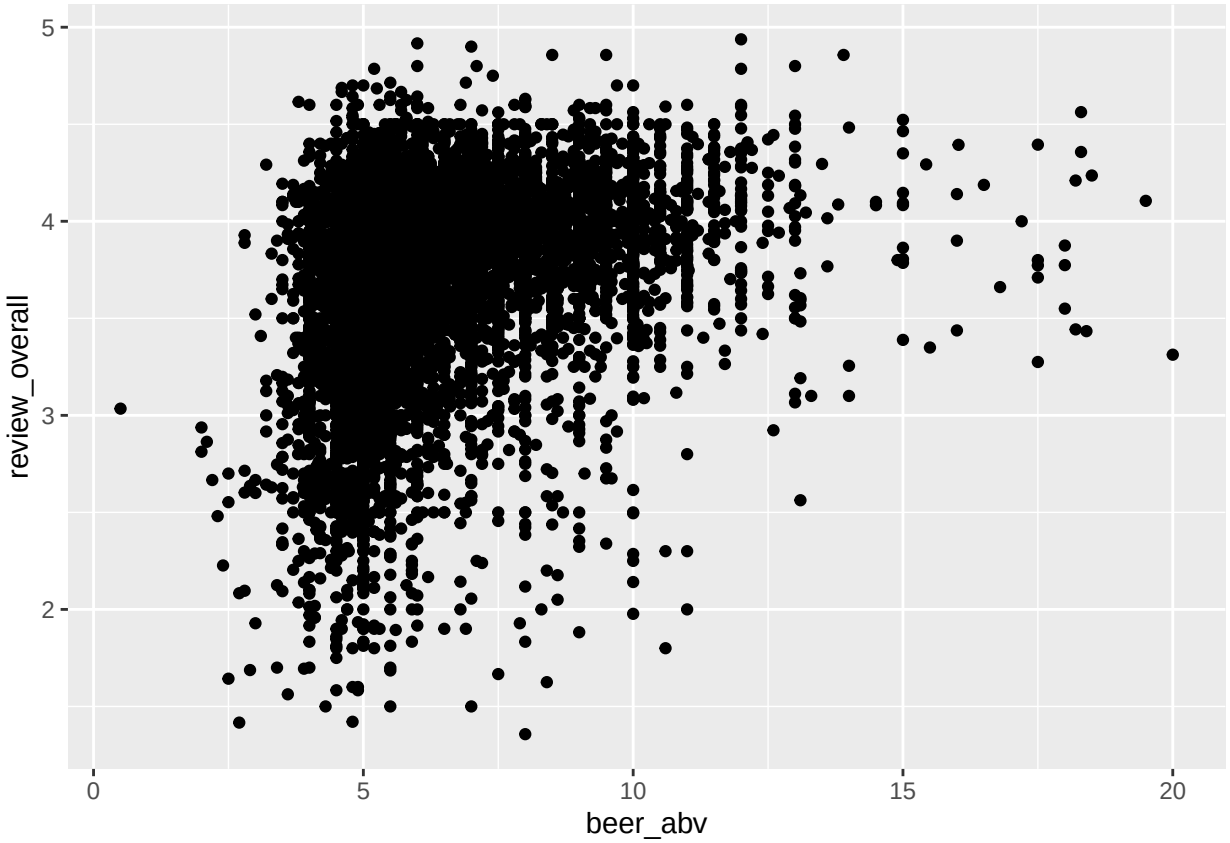
```
# You can save your plot as a R object
abv_score_plot <- ggplot(data=beer_data)+
  geom_point(mapping = aes(x = beer_abv,y=review_overall))

# And retrive the plot by calling that object
abv_score_plot
```



Adjusting the X and Y axis limits

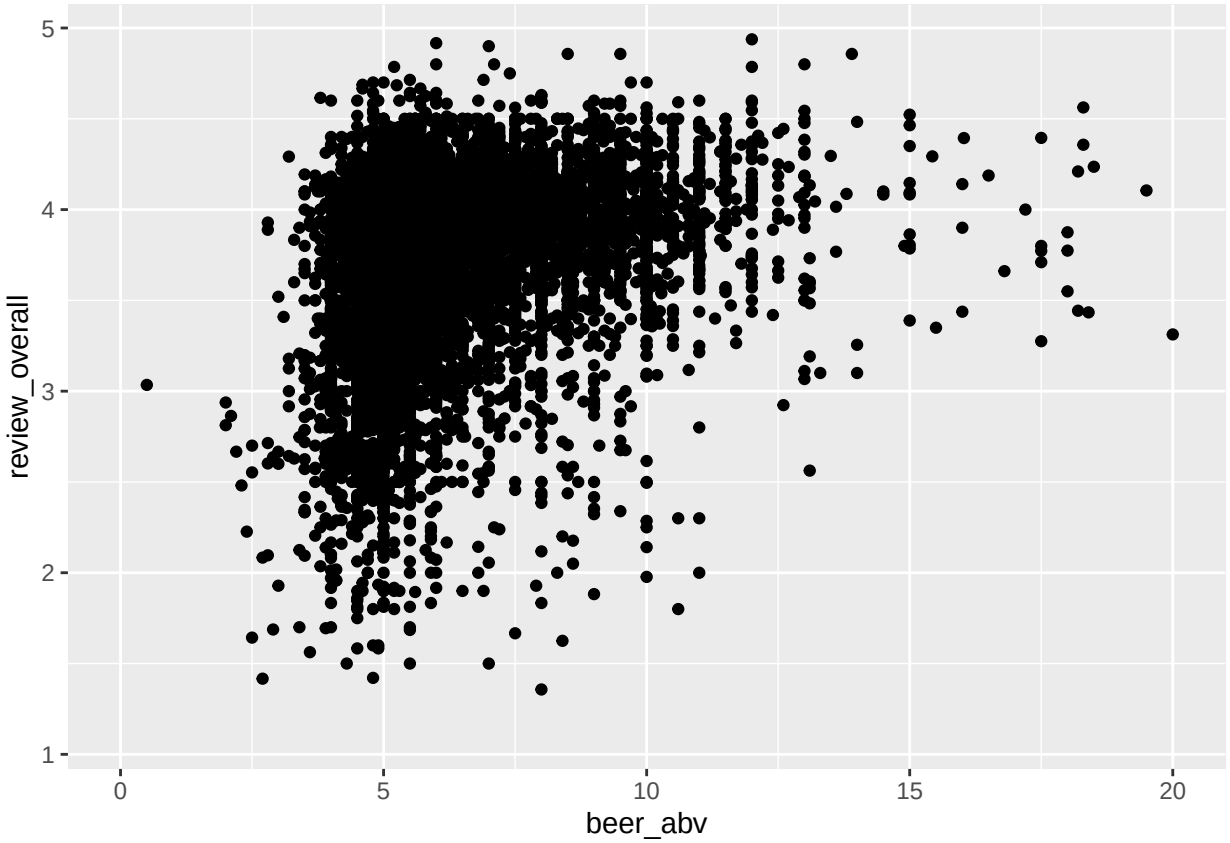
This way, you are removing any data points outside the range 0-20

```
require(dplyr)

ggplot(
    # This means you are
    data=beer_data %>%
         filter(beer_abv<=20)
       )+
  geom_point(mapping = aes(x = beer_abv,y=review_overall))
```



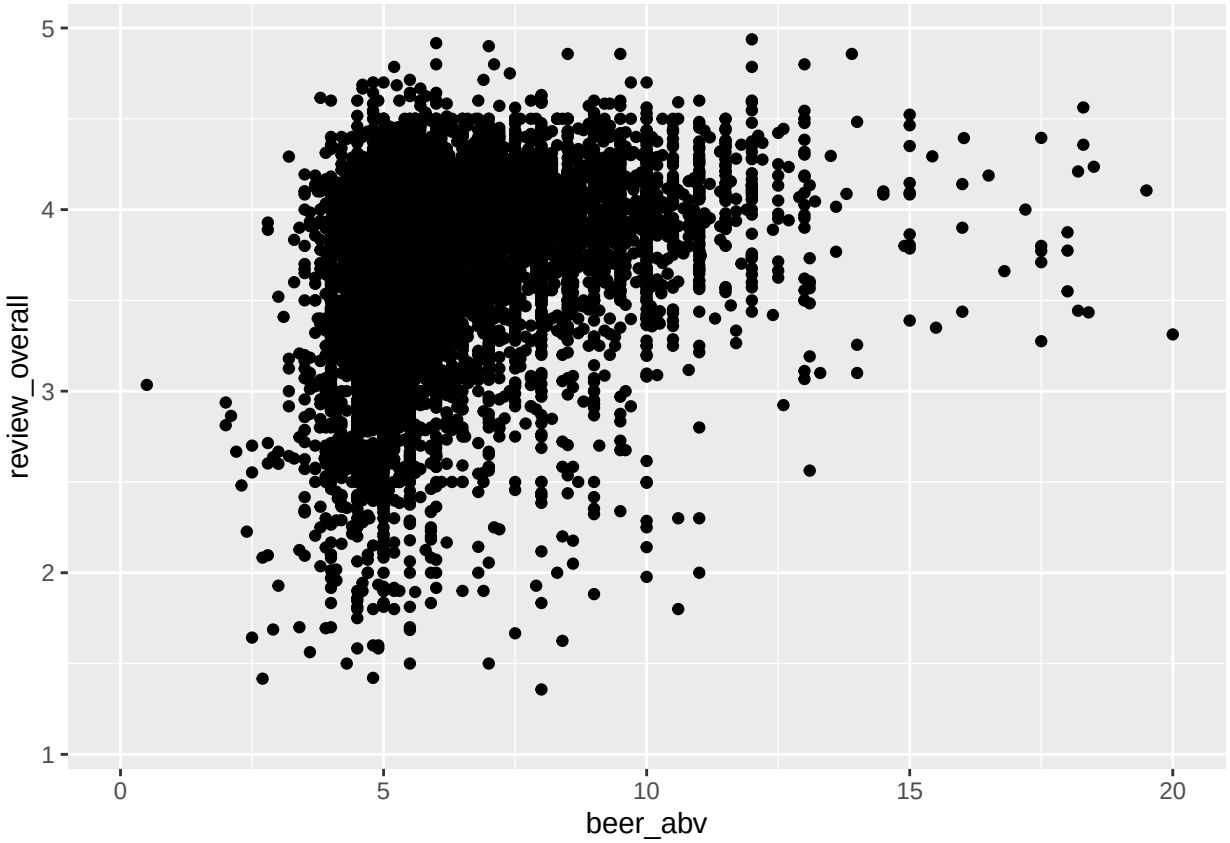
Through this method you explicitly limit your x-axis to 0-20 range. This does remove the points outside of the range, so if you are calculating summary statistics through a ggplot function, it will ignore those points.

```
abv_score_plot + xlim(0,20)
```



The other method is to change the axes limits by zooming in to the region of interest without deleting the points. This is done using `coord_cartesian()`. This does not remove any data points, just effectively provides a "zoom in/zoom out" capability.

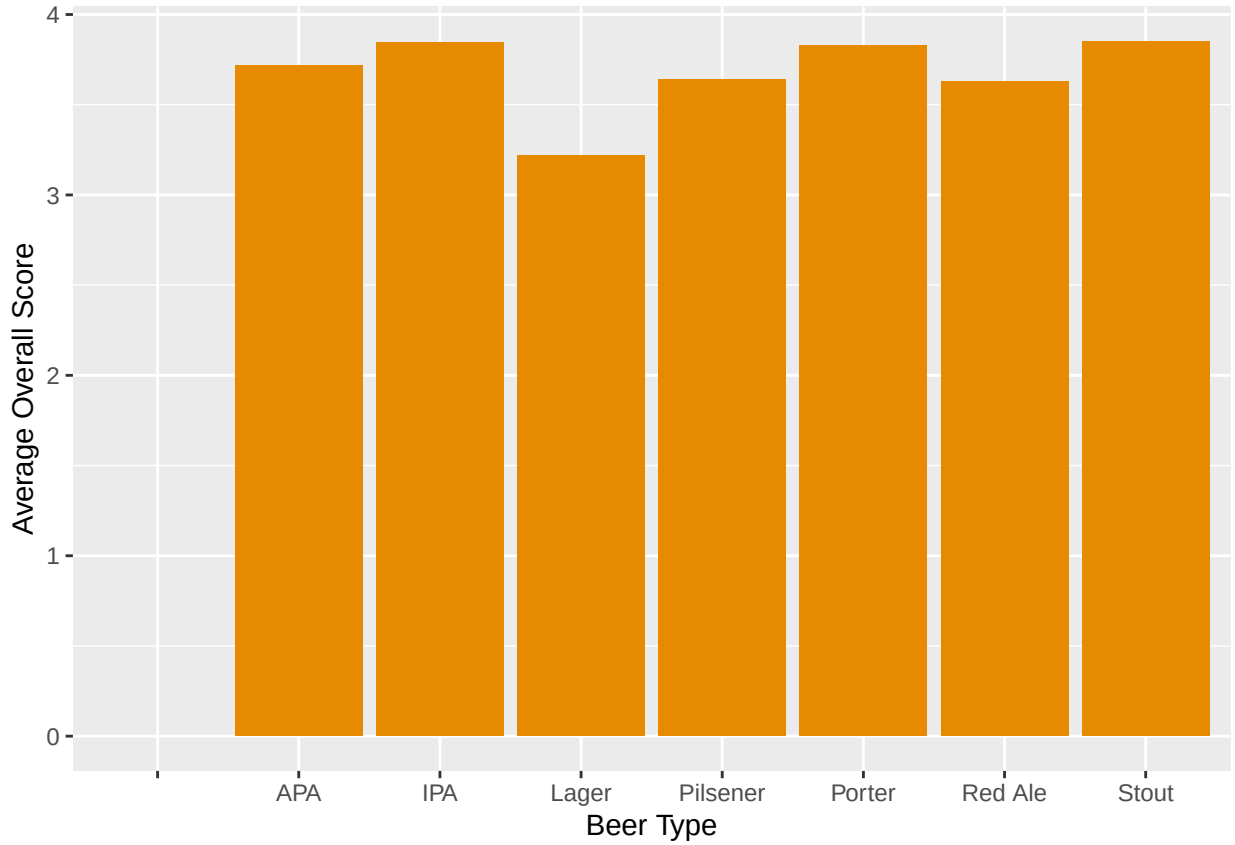
```
abv_score_plot+coord_cartesian(xlim=c(0,20))
```



Create a barplot

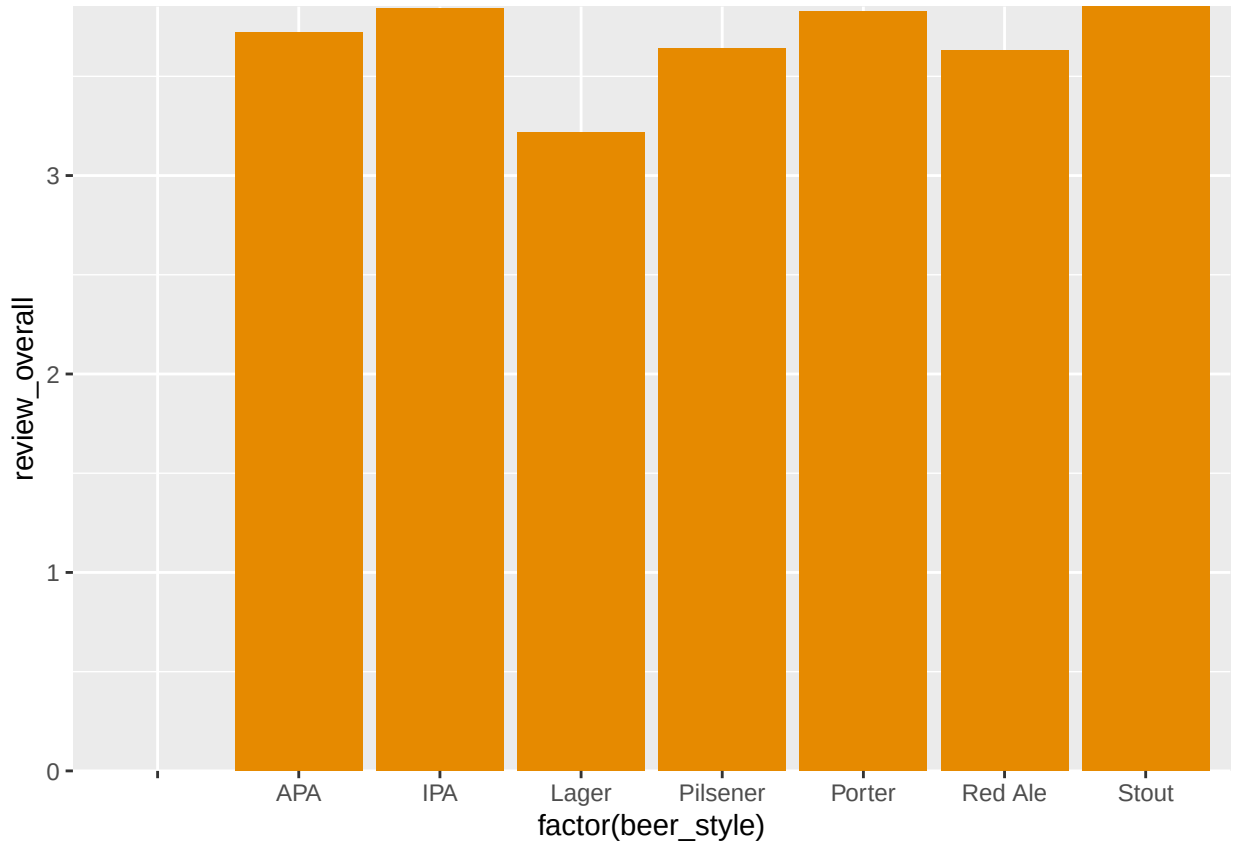
`geom_bar()` plots, well, bars. It can take a number of options:

```
ggplot(beer_data, mapping = aes(x = factor(beer_style), y = review_overall)) +
  ## Notice that I am defining color ('fill') outside of the "mapping" argument, therefore it is a fixe
  geom_bar(stat = "summary", fun = "mean", fill = "#E68A00") +
  labs(x = "Beer Type", y = "Average Overall Score")
```



Notice the bars are annoyingly levitating on the plot. I don't know about, but that really upsets me. Here we can introduce adjustment to axes on ggplot

```
ggplot(beer_data, mapping = aes(x = factor(beer_style), y = review_overall)) +
  ## Notice that I am defining color ('fill') outside of the "mapping" argument, therefore it is a fixe
  geom_bar(stat = "summary", fun = "mean", fill = "#E68A00") +
  ## Adjust y axis not to expand on both limits
  scale_y_continuous(expand=c(0,0))
```



Colors

Split data by color

```
abv_score_plot_color<-ggplot(beer_data,
                             mapping=aes(color = factor(beer_style),
                                         x = beer_abv,
                                         y=review_overall)) +
  geom_point()+
  xlim(c(0,20))
```

Changing colors

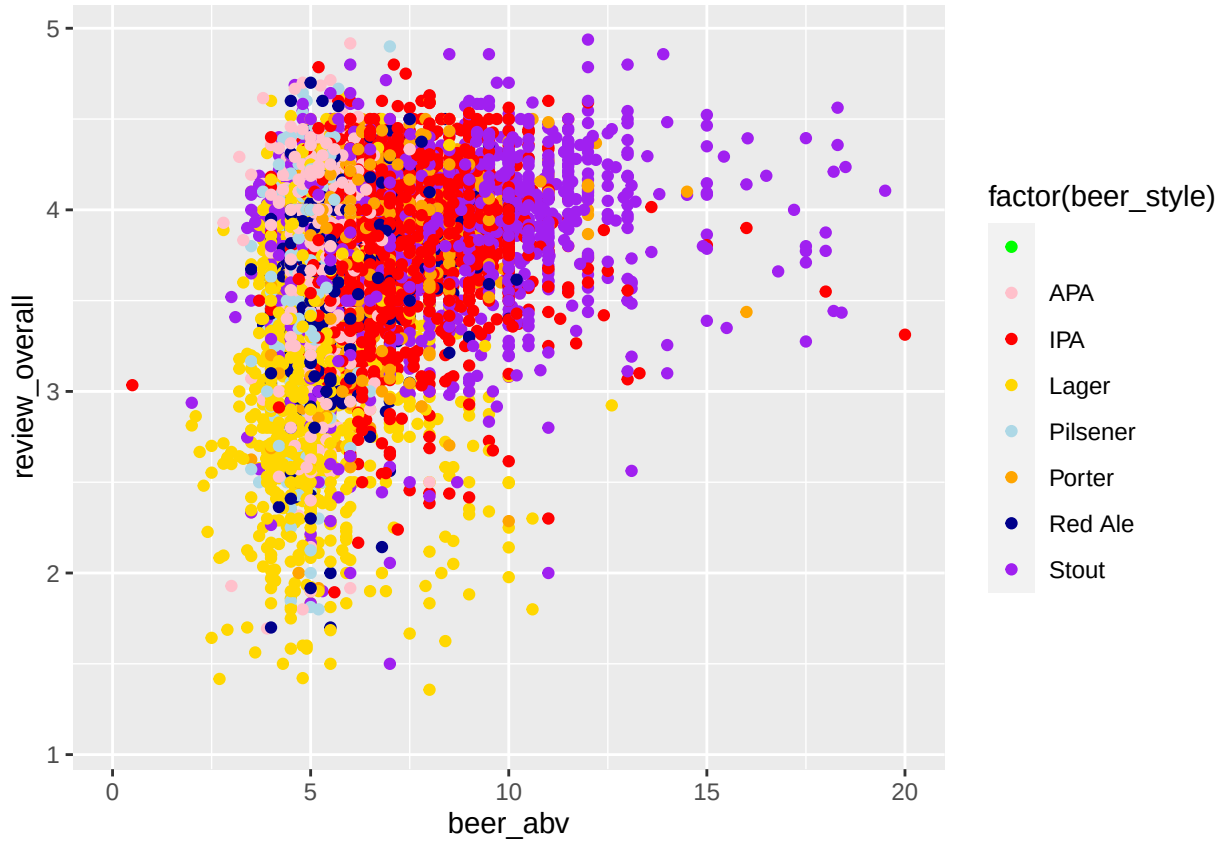
As you probably know (or noticed), the default palette from ggplot is kind of rubbish. We can alter the color legend on the plot with the `scale_fill` or `scale_color` family functions

```
# Create a color palette
color_palette<-c(
  "purple","gold","#FF0000","pink",
  "orange","darkblue","lightblue","green")

# Name this color palette with the possible categories
names(color_palette) <- unique(beer_data$beer_style)
```

```
color_palette
```

```
##       Stout       Lager         IPA         APA      Porter     Red Ale
##    "purple"      "gold"   "#FF0000"      "pink"    "orange"  "darkblue"
##    Pilsener
## "lightblue"     "green"
```

```
# Generate plot with the manual color scale
abv_score_plot_color+
  scale_color_manual(values=color_palette)
```



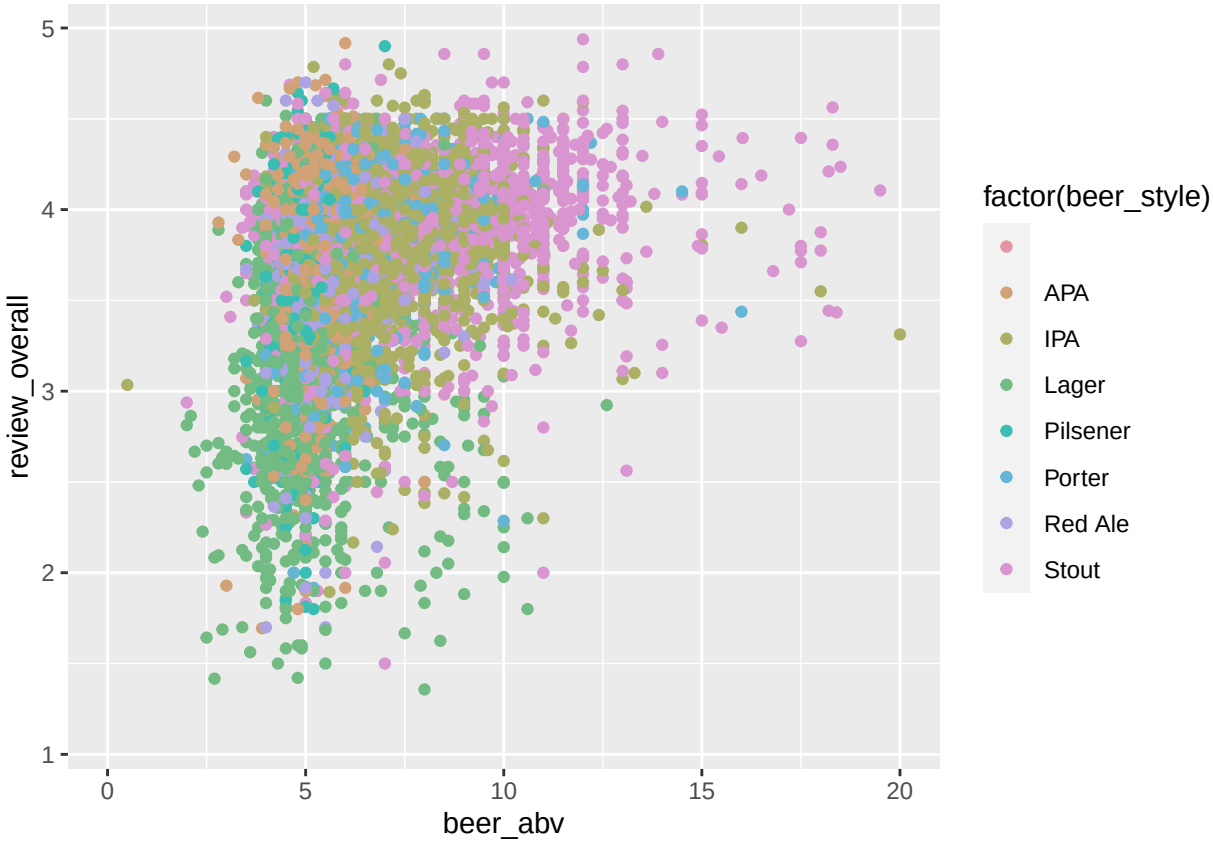
Using color packages

```
require(colorspace)
require(viridis)
```

We can see all pre-made palettes from `colorspace`

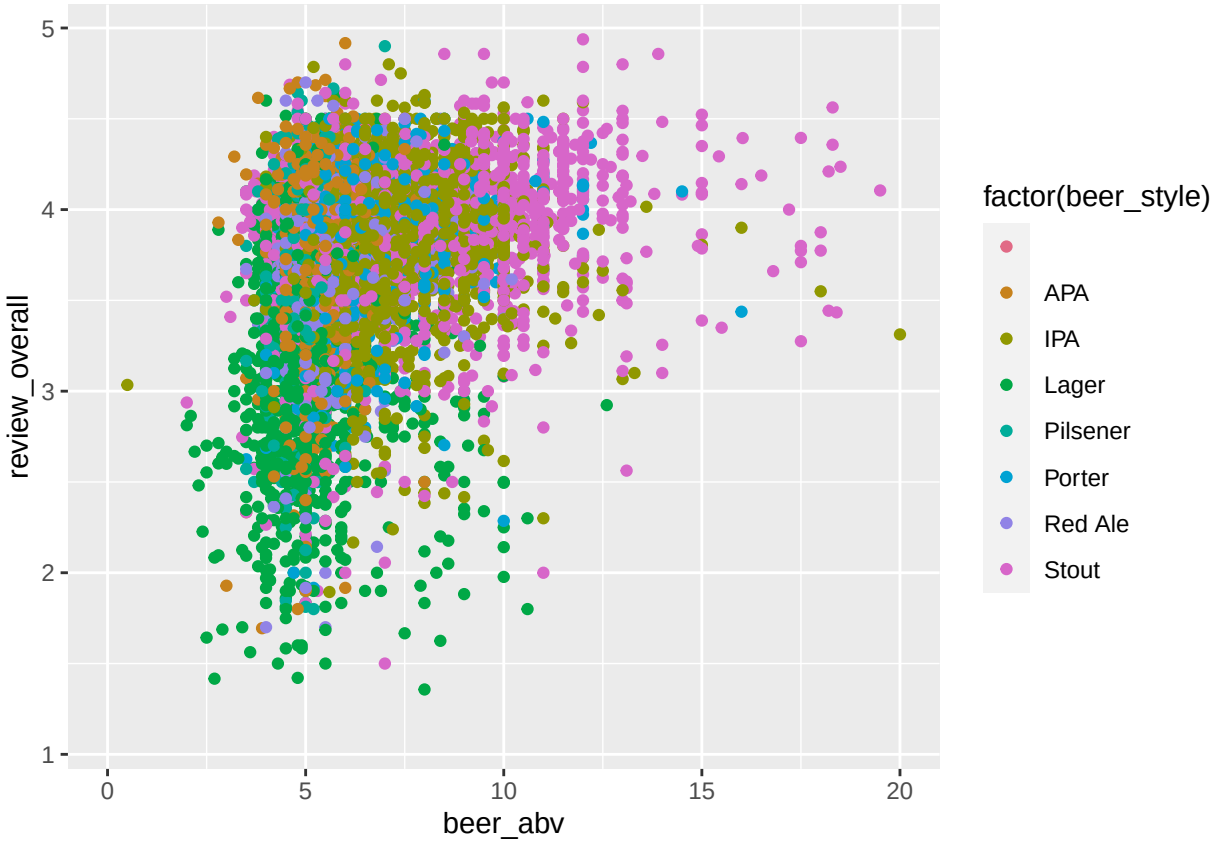
```
colorspace::hcl_palettes(plot = TRUE)
```


```
palette <- colorspace::rainbow_hcl(n = length(unique(beer_data$beer_style)))


abv_score_plot_color+
  scale_color_manual(values=palette)
```



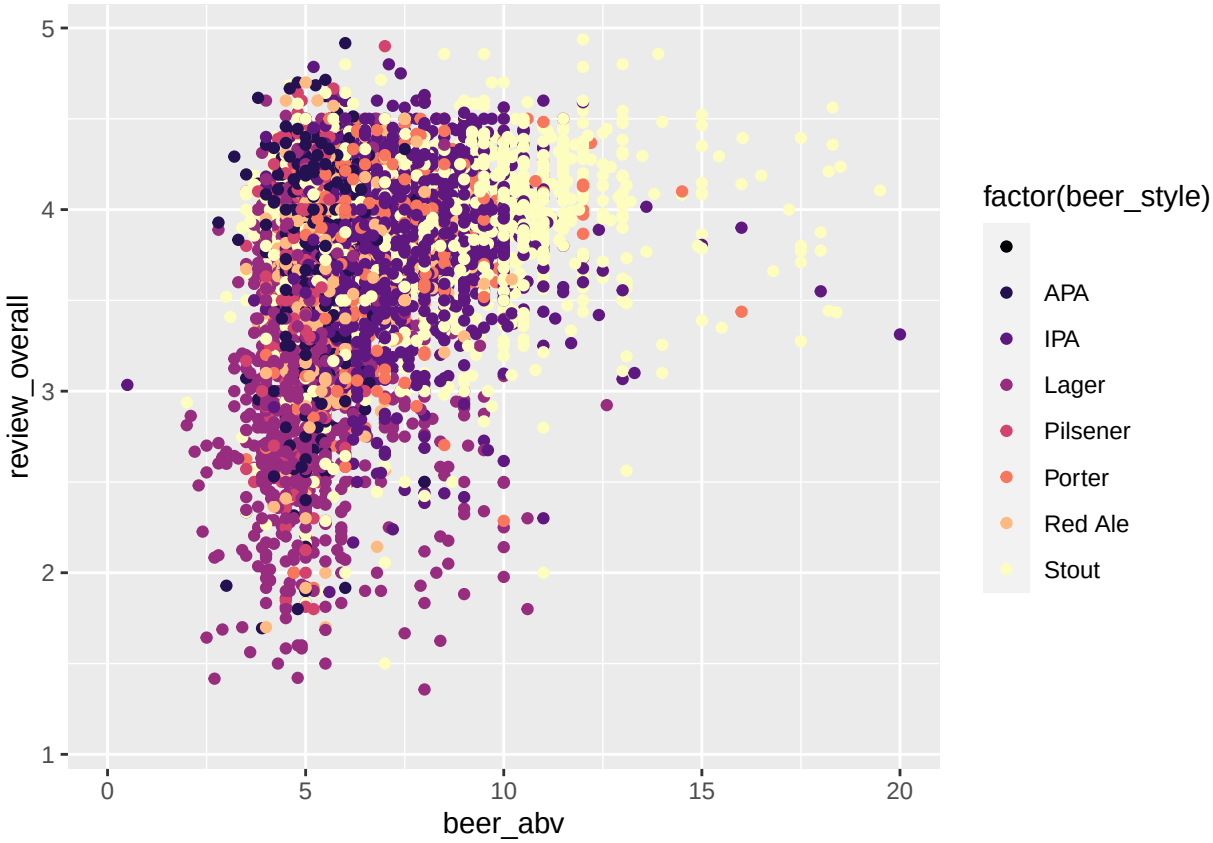
`colorspace` also comes with wrapper functions that replace the native ggplot2 ones. In these functions, you can use a

```
abv_score_plot_color+
  colorspace::scale_color_discrete_qualitative(palette="Dark 3")
```



The `viridis` package also provides pre-made palettes

```
viridis_palette<-viridis(n = length(unique(beer_data$beer_style)),option = "magma")

abv_score_plot_color+
  scale_color_manual(values=viridis_palette)
```



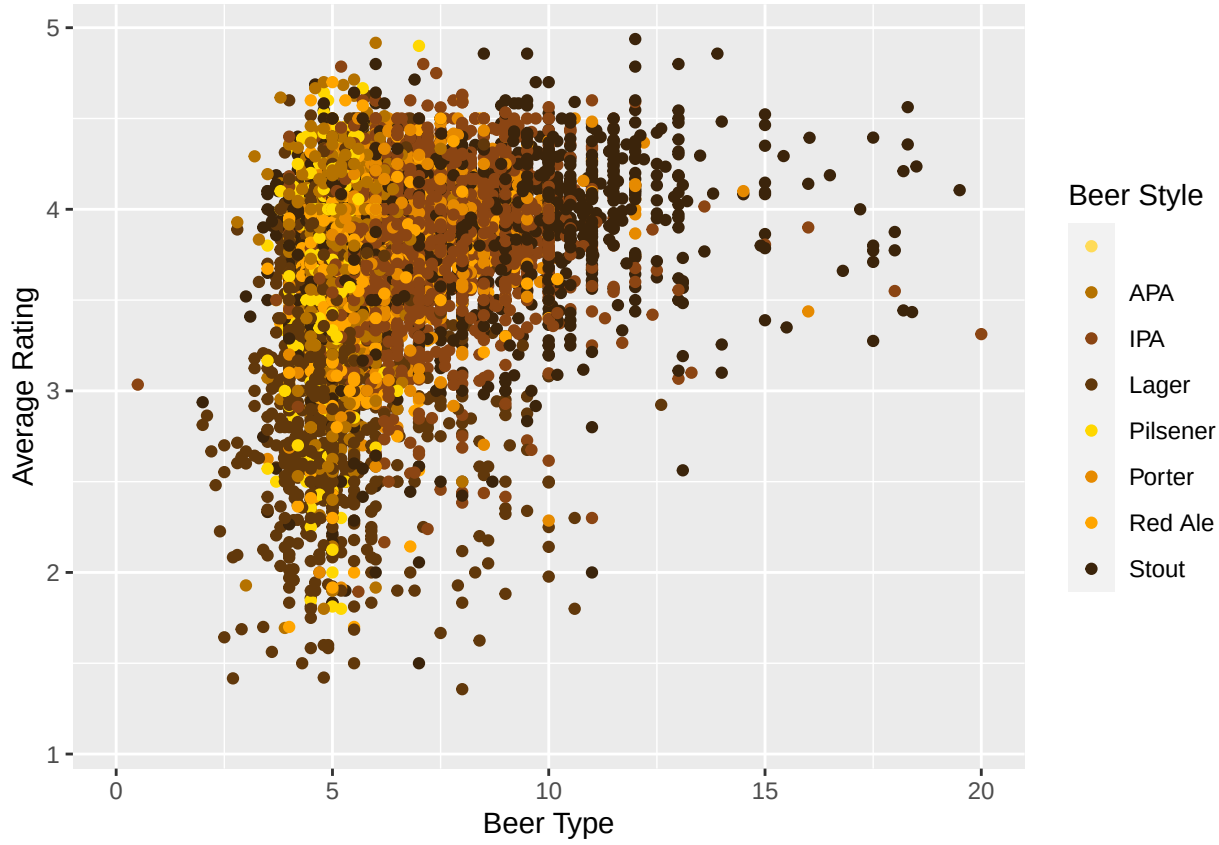
Other resources

Personally, I use http://www.coolors.co to pick palettes for my graphs. Feel free to browse it.

A nifty thing is that each palette comes with the color information embedded on its link. We can use this to our advantage

```
## Function to convert coolors.co link to a character vector in R
coolors2vector<-function(url){

  require(stringr)

  # Remove the 'website' part of the string, leaving only the hex codes
  url<-gsub(x=url,"https://coolors.co/","")

  # Split the hex codes by the character "-",
  #leaving us with a vector of codes
  vector<-str_split(url,"-")[[1]]

  # Add the hashtag character to the front of each element to
  # make it recognizable as a hexcode
  resu<-paste("#",vector,sep="")

  return(resu)

}
```

```r
# Create a beer palette from a coolors.co link
beer_palette<-coolors2vector("https://coolors.co/ffdb58-ffd700-ffa500-e68a00-b57200-8b4513-61380b-3b240")

# Name it with the styles of beer
names(beer_palette) <- rev(unique(beer_data$beer_style))

# Create plot
abv_score_plot_color+
  scale_color_manual(values = beer_palette)+
  labs(x = "Beer Type", y = "Average Rating",color="Beer Style")
```



In addition, when I am out of ideas for a color palette (or just lazy), I also use chatGPT to come up with some (your mileage may vary)

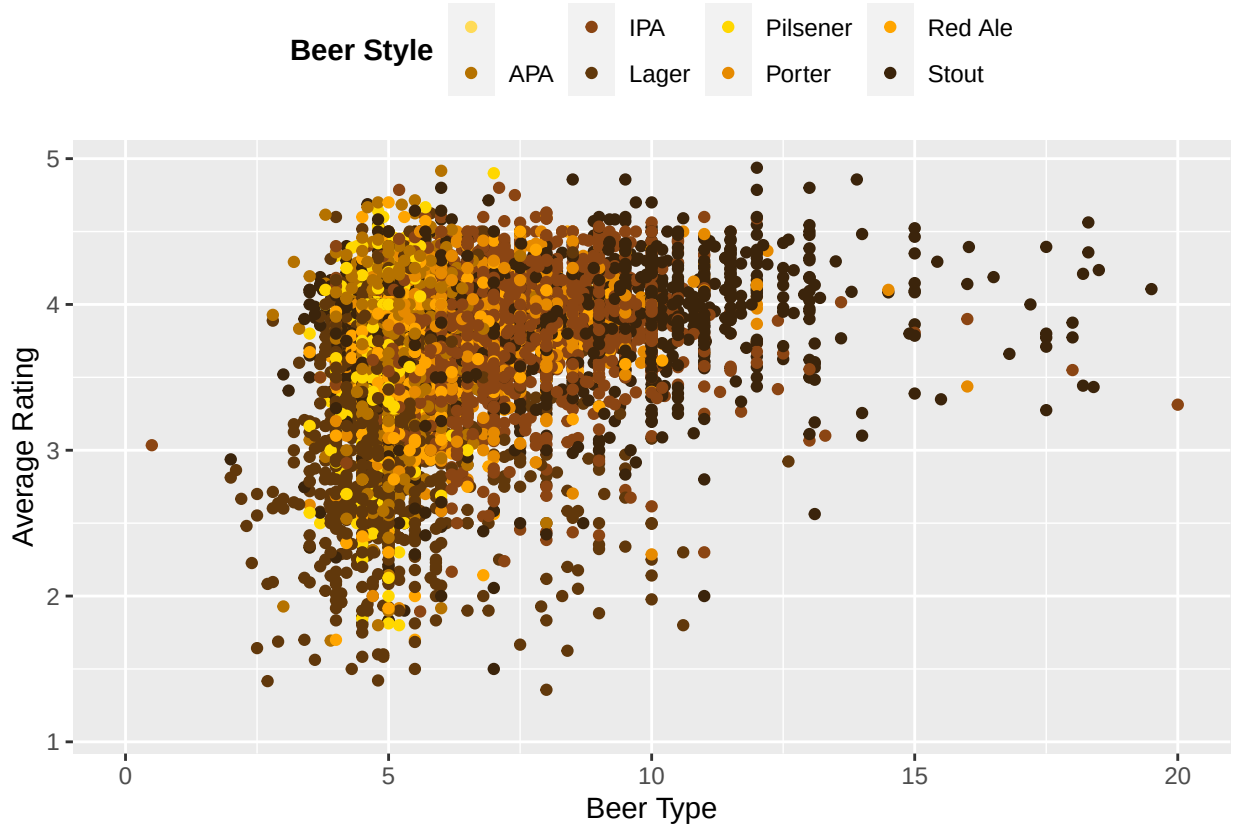
A link with the prompt I usually use: https://tinyurl.com/chatgptcolorprompt

Adding 'themes'

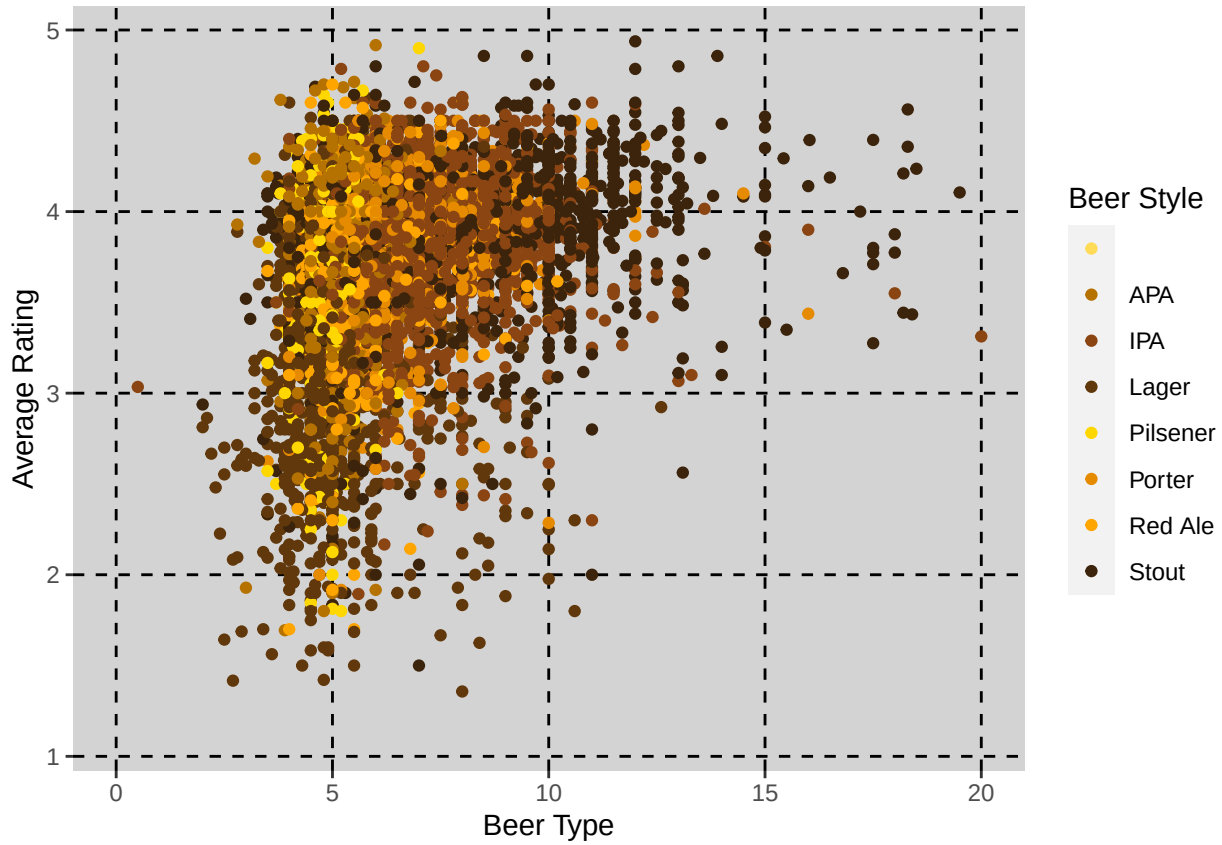
Themes are specifications on how elements are present. Not only geometries, but other elements of the plot (text, background, ticks, ticklabels, you name it.

```r
abv_score_plot_color+
  scale_color_manual(values = beer_palette)+
  labs(x = "Beer Type", y = "Average Rating",color="Beer Style")+
  ## One argument to make legend title bold, one to make it go on top of the plot
```

```
  theme(legend.title = element_text(face = "bold"),
          legend.position = "top")
```




```
abv_score_plot_color+
  scale_color_manual(values = beer_palette)+
  labs(x = "Beer Type", y = "Average Rating",color="Beer Style")+
  ## Customize the plot background and grid lines
  theme(panel.background = element_rect(fill = "lightgray"),
          panel.grid.major = element_line(color = "black", linetype = "dashed"),
          panel.grid.minor = element_blank())
```

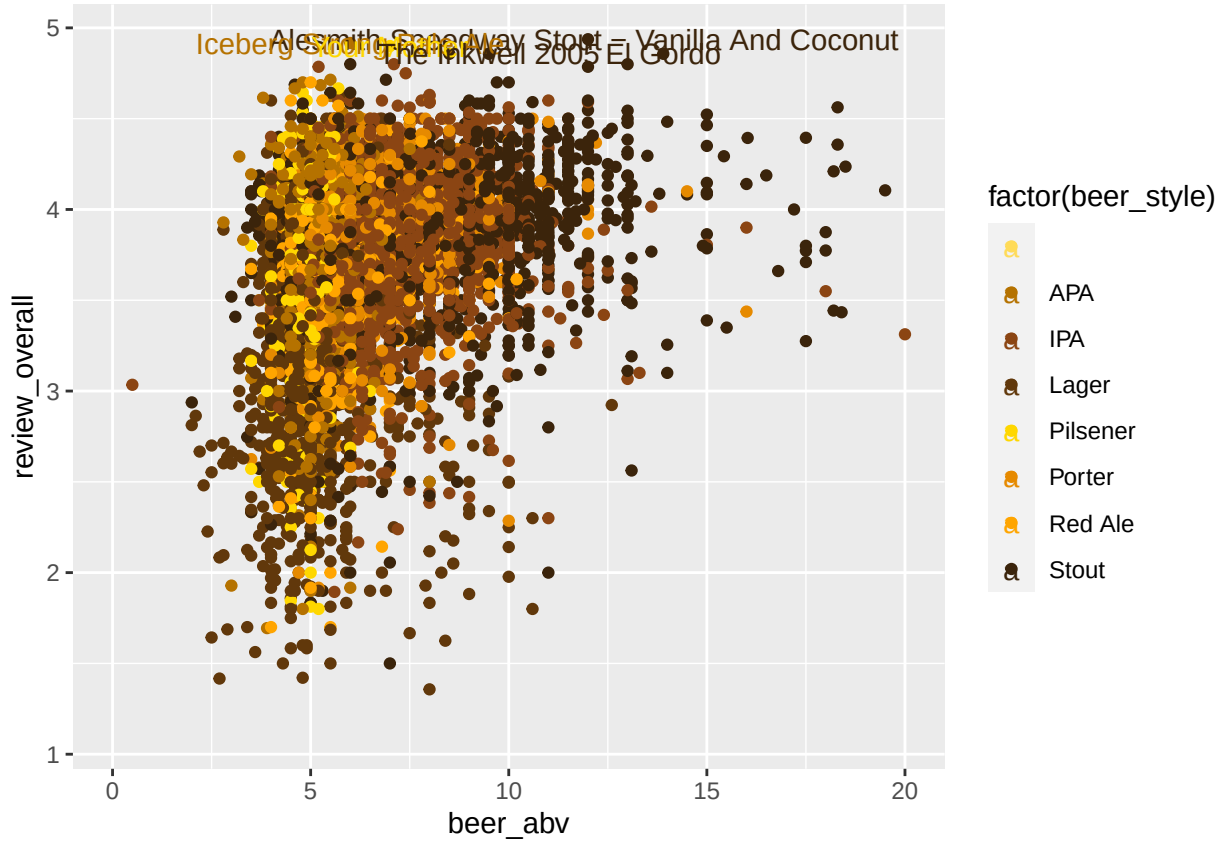


Adding text labels

What if I want to add text labels to data points in the plot? For the plot we are working on, there are way too many data points for that. But I can single out the top 5 best rated beers in the dataset!

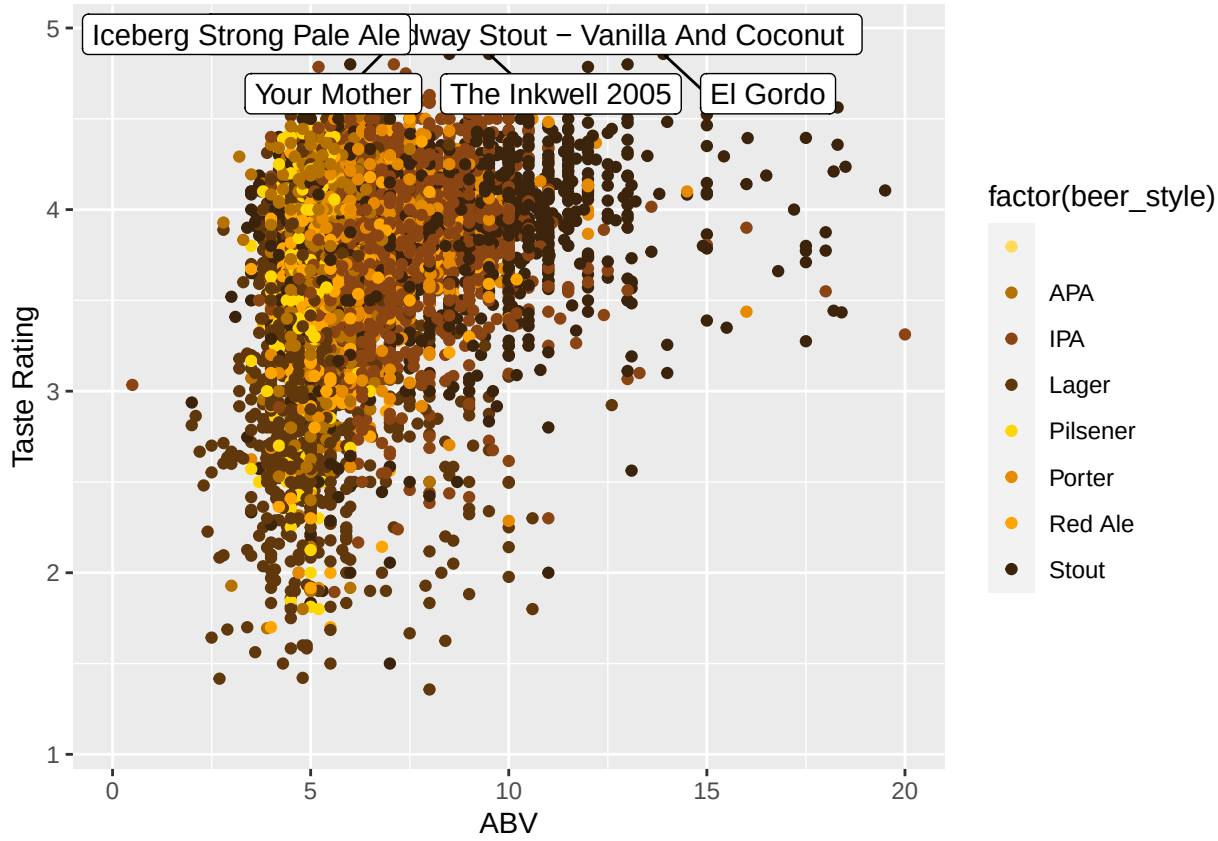
```
# Get top 5 scoring beers

top5<-beer_data%>%

  # Filter values that are NOT NA (that`s what "!" means)
  dplyr::filter(!is.na(beer_abv))%>%

  # Arrange values by descending rating
  dplyr::arrange(desc(review_overall))%>%

  # Creatig a column called 'rank' that goes from 1 to the number of obs
  dplyr::mutate(rank=1:n())%>%

  # Filtering for only the first 5 rankings
  dplyr::filter(rank <= 5)



abv_score_plot_color+
```

```
  # Create a text layer, but using as dataset a dataset containing only the top 5 beers
  geom_text(data=top5,mapping=aes(x=beer_abv,y=review_overall,
                                  label=beer_name))+
  scale_color_manual(values = beer_palette)
```



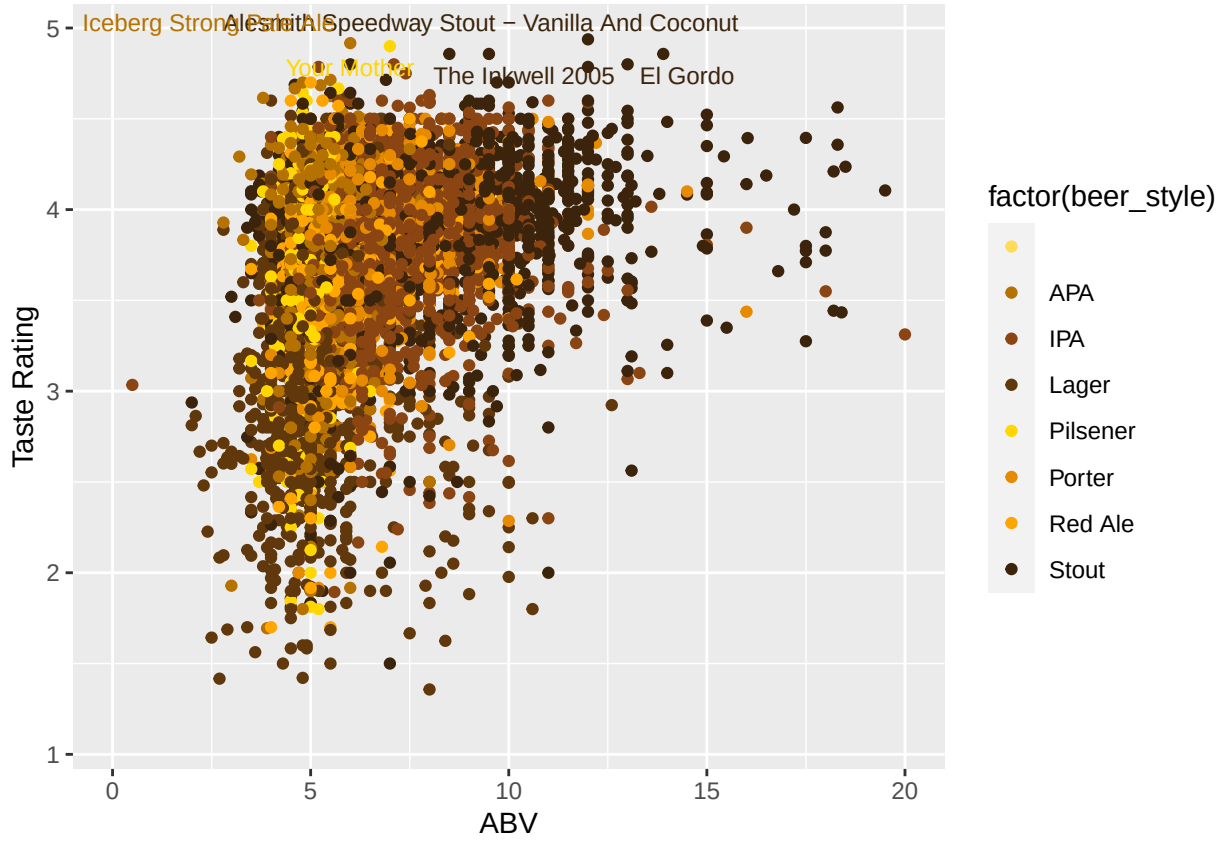
This looks like rubbish! The names are overlapping, and it is not clear which label belongs to each datapoint. Some googling suggests that the package `ggrepel` might help

```
require(ggrepel)

abv_score_plot_color+
  ggrepel::geom_label_repel(data=top5,mapping=aes(x=beer_abv,y=review_overall,label=beer_name),color="bl
  scale_color_manual(values = beer_palette)+
  labs(x="ABV",y="Taste Rating")
```



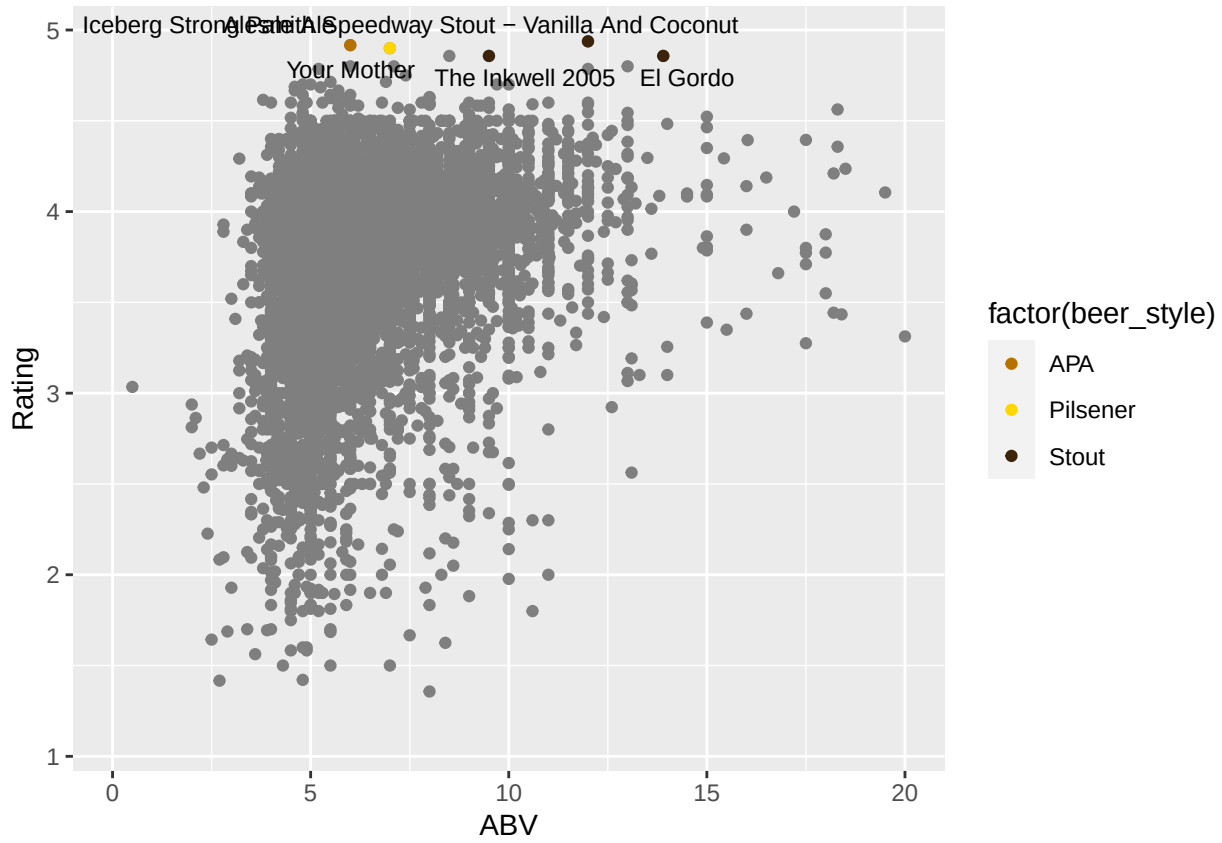
That looks slightly better, but there is still overlap. Maybe we can change the size of the font. Also, notice that there is an 'a' on the legend key. This is because the legend is account for 'text' layer on the plot. We can get ride of that as well since it's the same info as the point colors.

```
abv_score_plot_color+
  ggrepel::geom_text_repel(data=top5,mapping=aes(x=beer_abv,y=review_overall,label=beer_name),size=3)+
  scale_color_manual(values = beer_palette)+
  # Set the label to the empty string "" (or any other string).
  guides(color = guide_legend(override.aes = aes(label = "")))+
  labs(x="ABV",y="Taste Rating")
```



It's still very messy. That's usually the issue we have when there is too much data. It's good though, because it forces us to think about what information do we really want to convey in each plot. The goal of a figure is not to show every single piece of information of your data - that's what the actual data is for. So let's say I do not care about the role the beer type plays in this relationship - I just want focus on the relationship between ABV and rating. I also want to point out the best rated beers, and their beer type. Therefore I can remove the colors of the dataset and keep it just for the top 5.

```
ggplot(beer_data, aes(x = beer_abv,y=review_overall)) +

  geom_point(color="grey50")+

  ggrepel::geom_text_repel(
    data=top5,
    mapping=aes(x=beer_abv,y=review_overall,label=beer_name),
    color="black",size=3)+

  geom_point(data=top5,mapping=aes(color=factor(beer_style)))+

scale_color_manual(values = beer_palette)+
  xlim(c(0,20))+
labs(x="ABV",y="Rating")
```



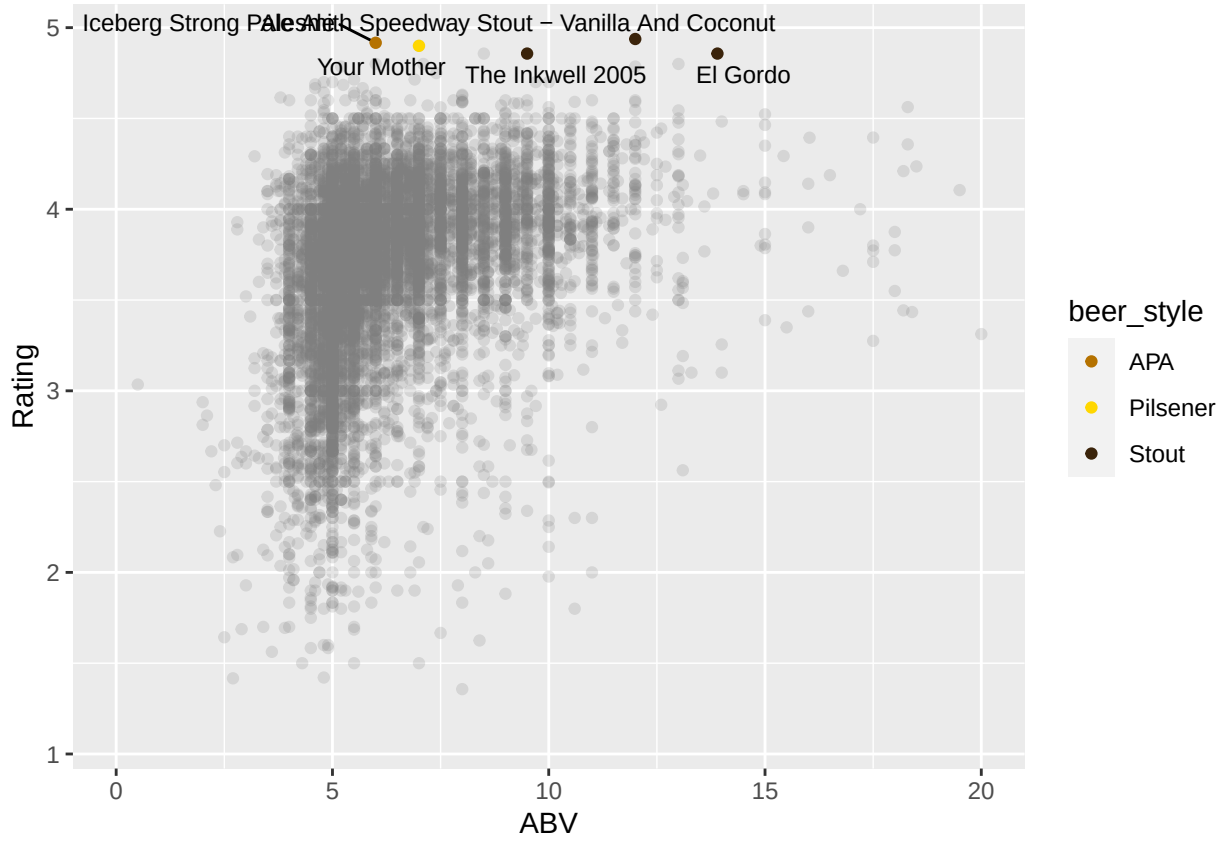
Looking better! But notice the colors changed. This is because we are only using a color mappipng for the top 5 beers, which do not contain all 5 categories. One way to do that is for that dataset to acknowledge all 5 categories.

```
top5<-beer_data%>%
  dplyr::filter(!is.na(beer_abv))%>%
  # This line makes beer_style a factor variable BEFORE it is filtered
  dplyr::mutate(beer_style=factor(beer_style))%>%
  dplyr::arrange(desc(review_overall))%>%
  dplyr::mutate(rank=1:n())%>%
  dplyr::filter(rank <= 5)

# As you can see, now we have all 7 levels encoded
levels(top5$beer_style)
```

```
## [1] "APA"      "IPA"      "Lager"    "Pilsener" "Porter"   "Red Ale"  "Stout"
```

```
# Run the plot again
ggplot(beer_data, aes(x = beer_abv,y=review_overall)) +
    geom_point(color="grey50",alpha=.2)+
    ggrepel::geom_text_repel(data=top5,mapping=aes(x=beer_abv,y=review_overall,label=beer_name),color="
    geom_point(data=top5,mapping=aes(color=beer_style))+
    scale_color_manual(values = beer_palette)+
    # Set the label to the empty string "" (or any other string).
    guides(color = guide_legend(override.aes = aes(label = "")))+
```

```
   xlim(c(0,20))+
  labs(x="ABV",y="Rating")
```

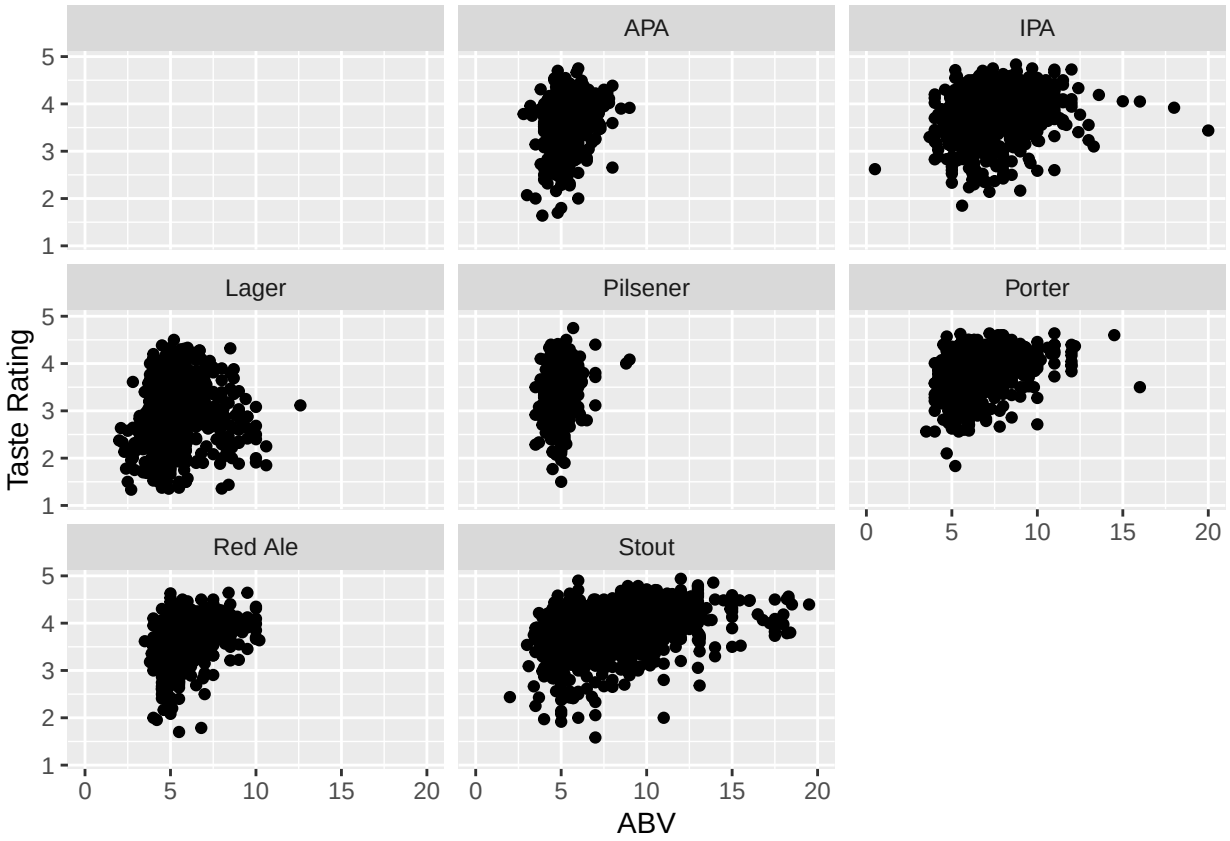


Creating multi-figures

`facet_wrap()`

You can split a plot you are making by a grouping variable, using the functions `facet_wrap()` and `facet_grid()`

```
ggplot(beer_data, aes(x = beer_abv,y=review_taste)) +
  geom_point()+
  facet_wrap(~beer_style)+
  labs(x="ABV",y="Taste Rating")+
  xlim(0,20)
```



This is a way to combine plots that have the same information. Sometimes it is intesting to present two plots with complementary information together in the same figure. For that I use the `cowplot` package

`cowplot`

Oops I didn't finish that (: