Arkaraj Mukherjee: H.W. 2

Loading necessary libraries:

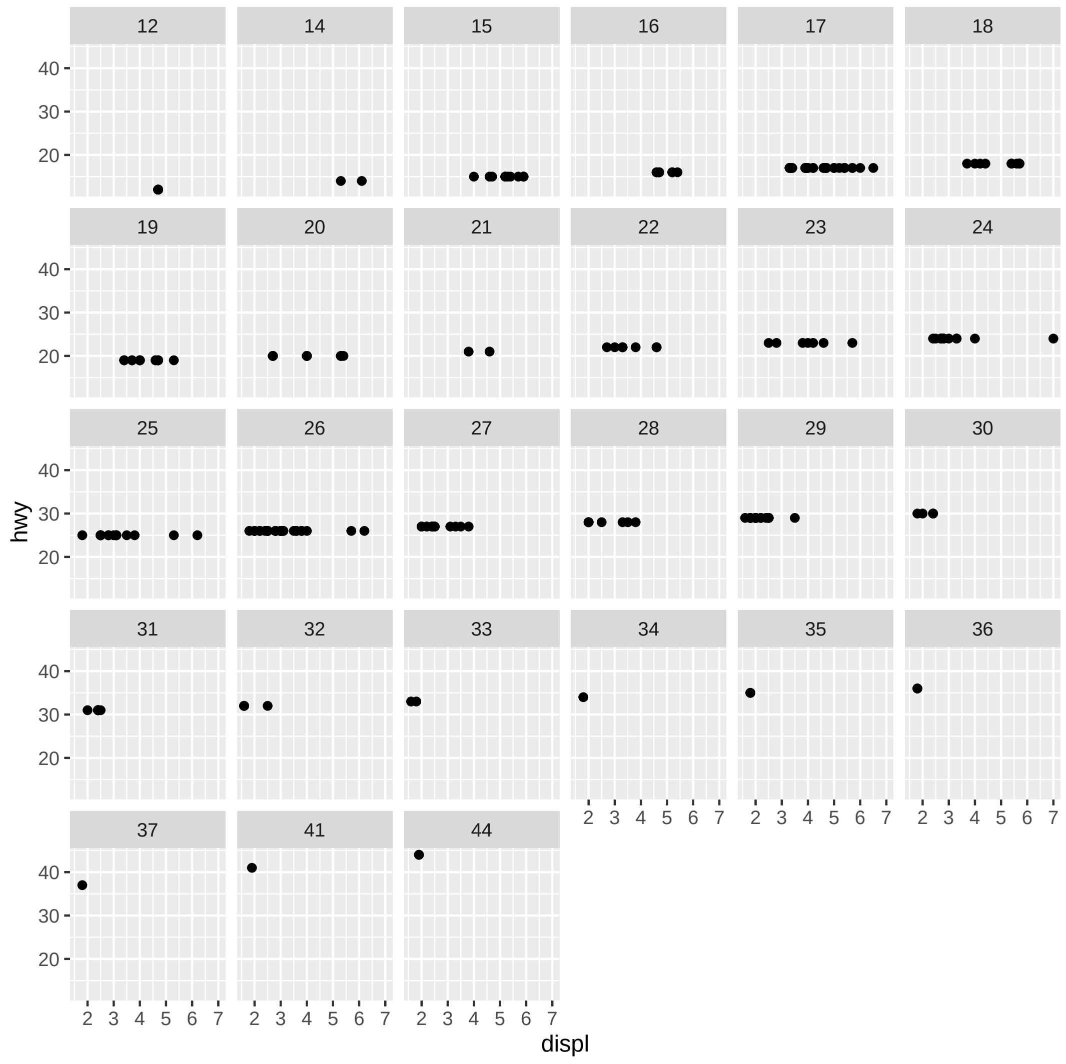
```
library("tidyverse");


## - Attaching core tidyverse packages ------------ tidyverse 2.0.0 -
## v dplyr     1.1.4     v readr     2.1.6
## v forcats   1.0.1     v stringr   1.6.0
## v ggplot2   4.0.1     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr     1.3.2
## v purrr     1.2.0
## - Conflicts --------------------- tidyverse_conflicts() -
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts
to become errors
```

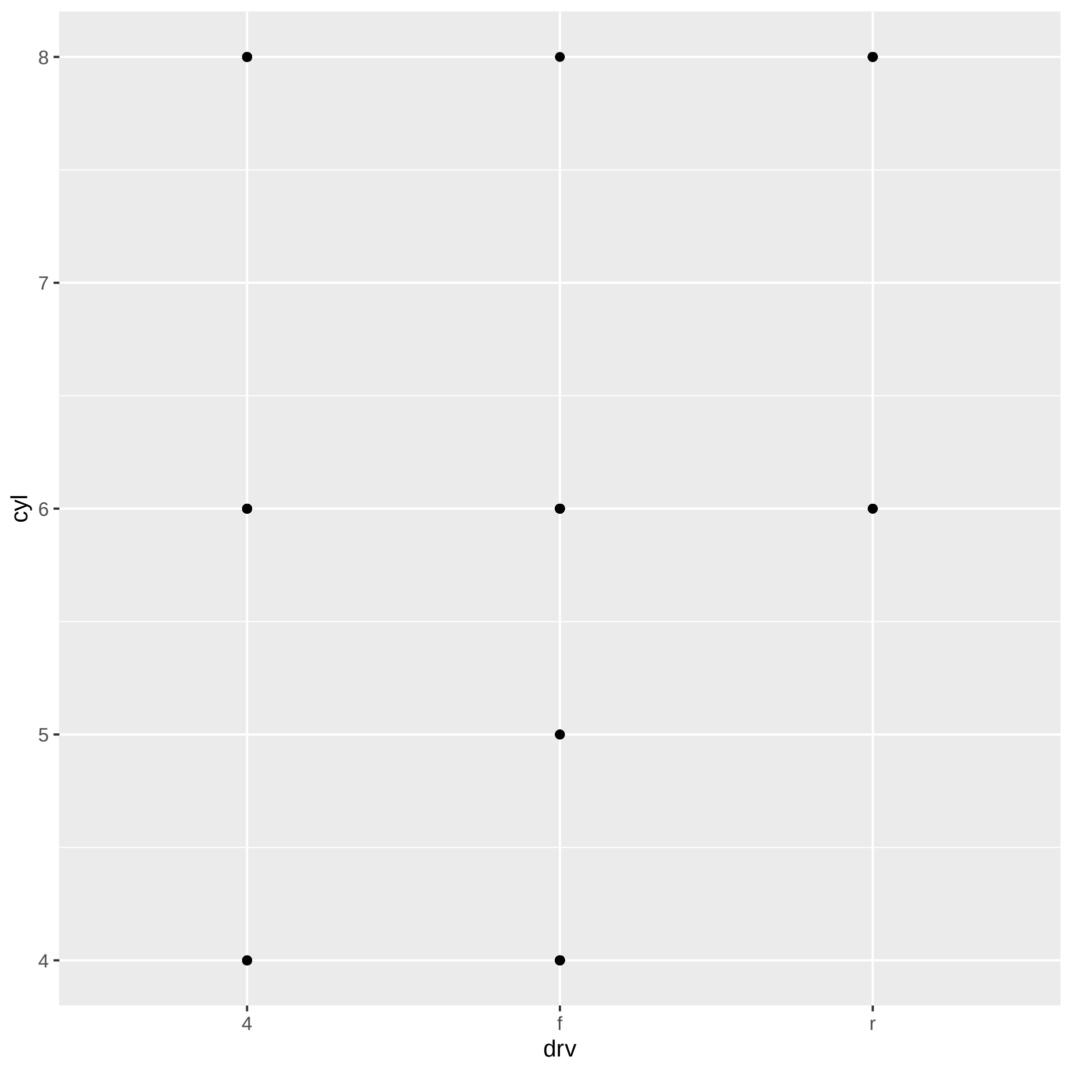
Solution 1:

(a) 1. If we facet a continuous variable then R will generate a seperate plot for every value this variable takes. Even though this does not always produce errors, it isn't always useful as the figure becomes increasingly cluttered as the range of the faceted variable gets larger in cardinality. For example, in the dataset `mpg` we facet one of its continuous variables `cty` while plotting `displ`×`hwy` graphs. This is still intelligible but it won't be if the `cty` took maybe ten more values even.

```
ggplot(data = mpg) +
geom_point(mapping = aes(x = displ, y = hwy)) +
facet_wrap(~hwy)
```



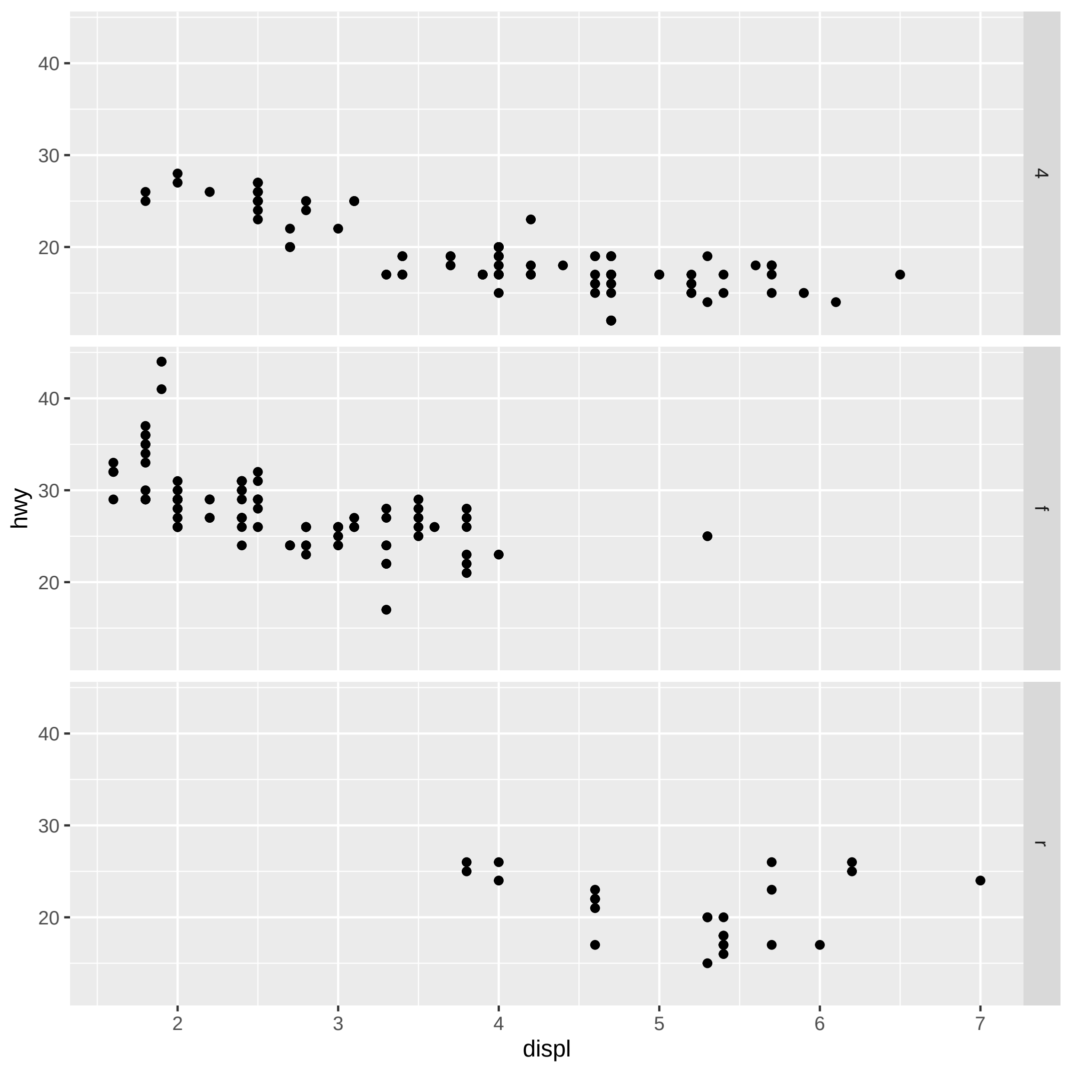
2.
```
ggplot(data = mpg) +
geom_point(mapping = aes(x = drv, y = cyl))
```



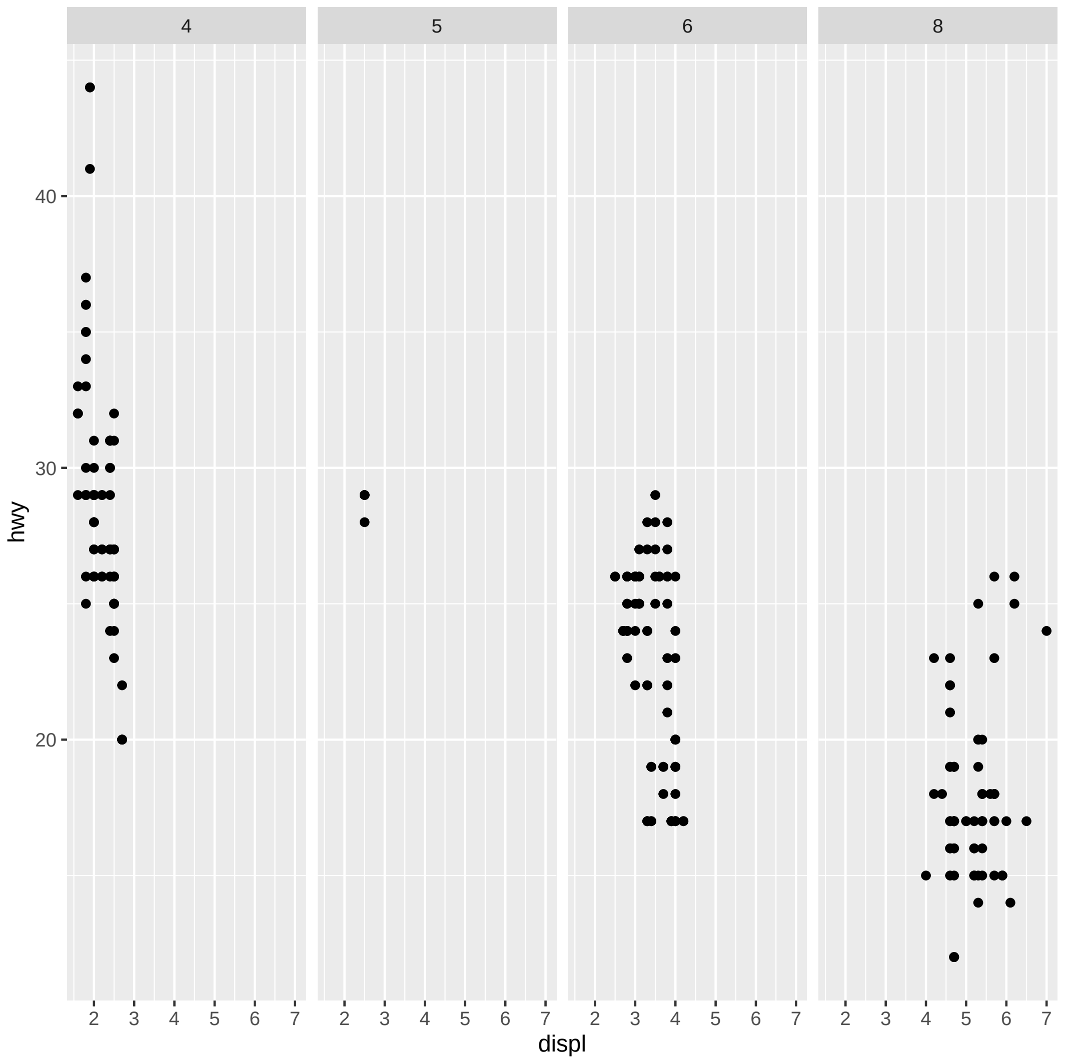
From this plot we can infer that there are no vehicles which are 4wd with 5 cylinders, rear wheel drive with 7 cylinders etc. and naturally the plots corresponding to these tuples in `drv`×`cyl` are empty.

3. for `facet_grid`, R interprets the dot as a placeholder for "nothing", this is necessary as `facet_grid` expects terms on both sides of the tilde( ) to make a matrix of panels following the structure `term_on_left`×`term_on_right` and in this case the usage of dot on the left results in a column of plots as R facets by "nothing" (hence, it does not facet) on the rows and only facets on the columns whereas a dot on the right results in a row of plots due to a similar reason.

```r
ggplot(data = mpg) +
geom_point(mapping = aes(x = displ, y = hwy)) +
facet_grid(drv ~ .)
```

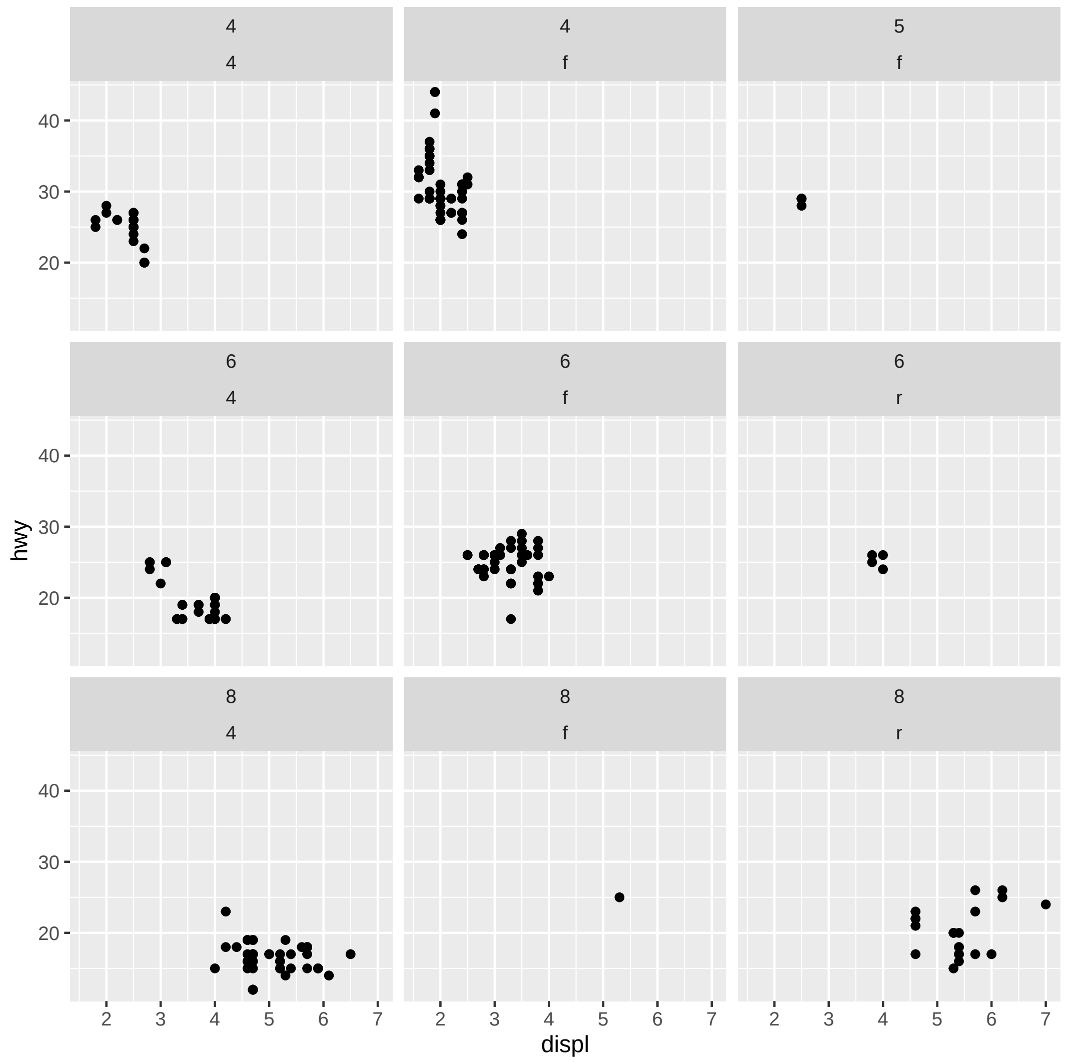


```r
ggplot(data = mpg) +
geom_point(mapping = aes(x = displ, y = hwy)) +
facet_grid(. ~ cyl)
```

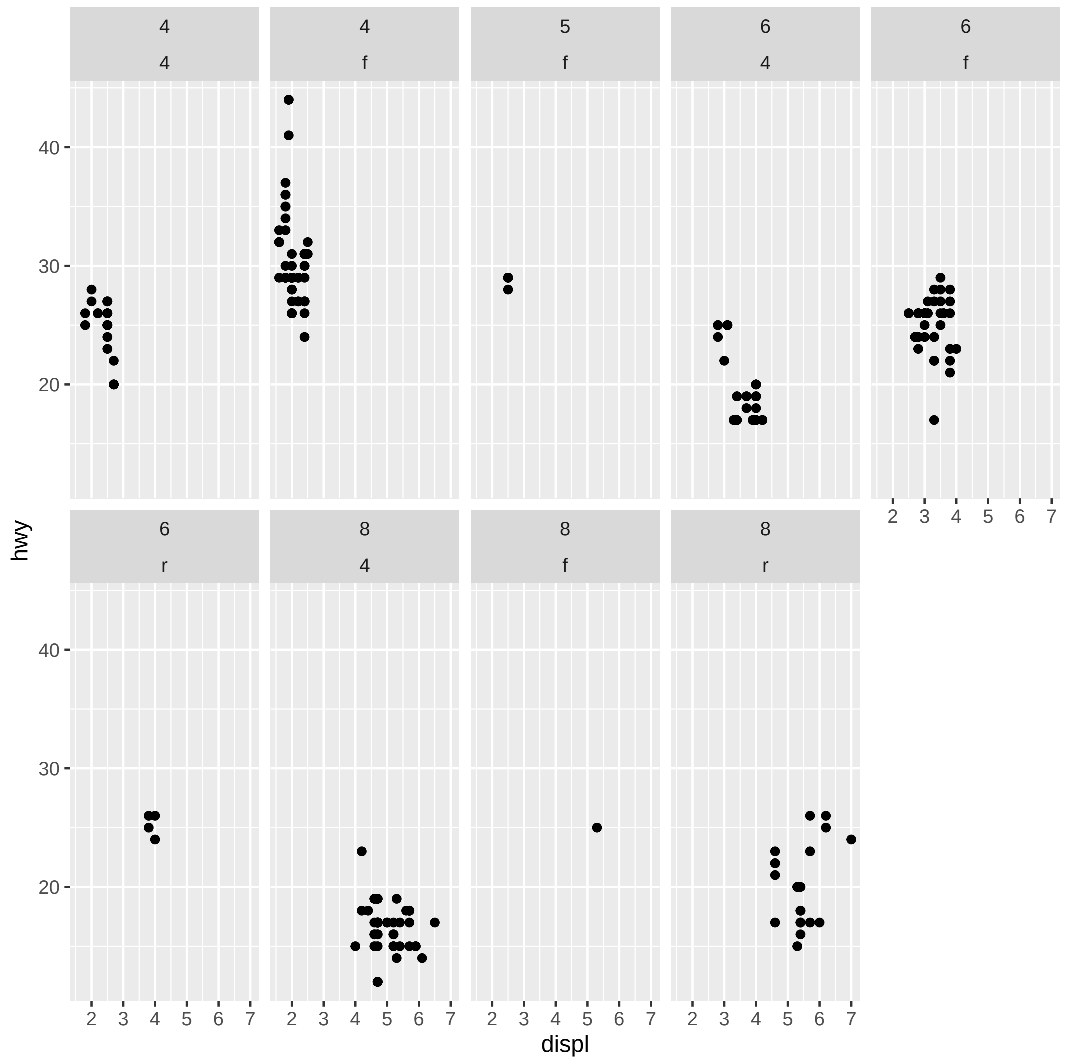


4. Plots using colour aesthetic to effectively add another dimension to the plot often loose some information due to the data points overlapping or nearly covering some point whereas this is not a problem when using `facet_wrap` as data points with different values of this parameters have completely seperate plots alloted to them and no information about this parameter is lost. It might seem from the above line that `facet_wrap` is overall superior but it is not, when the parameter to be added on (be it via faceting or using colour aesthetic) takes many values, the `facet_wrap` will get extremely cluttered and eventually rendered useless for continuous variables taking on more than maybe 50 values whereas the colour aesthetic can easily account for this by using the numerous different shades of colours available. On top of this, its not possible to infer globar properties of the parameter being faceted from `facet_wrap` as the points are in completetly seperate plots - which isn't a case when using colour aesthetic. All in all, we could say that `facet_wrap` usually scales better for larger datasets unless the parameter to be faceted takes on way too many values.
5. `nrow` and `ncol` fix the maximum number of rows and columns used on the screen, here is an example:

```
ggplot(data = mpg) +
geom_point(mapping = aes(x = displ, y = hwy)) +
facet_wrap(vars(cyl,drv))
```



```
ggplot(data = mpg) +
geom_point(mapping = aes(x = displ, y = hwy)) +
facet_wrap(vars(cyl,drv), nrow=2)
```



The other options that control the layout of individual panels are `as.table,strip.position,dir`. `facet_grid` does not have these arguements because the number of columns and rows are already fixed by its inputs.

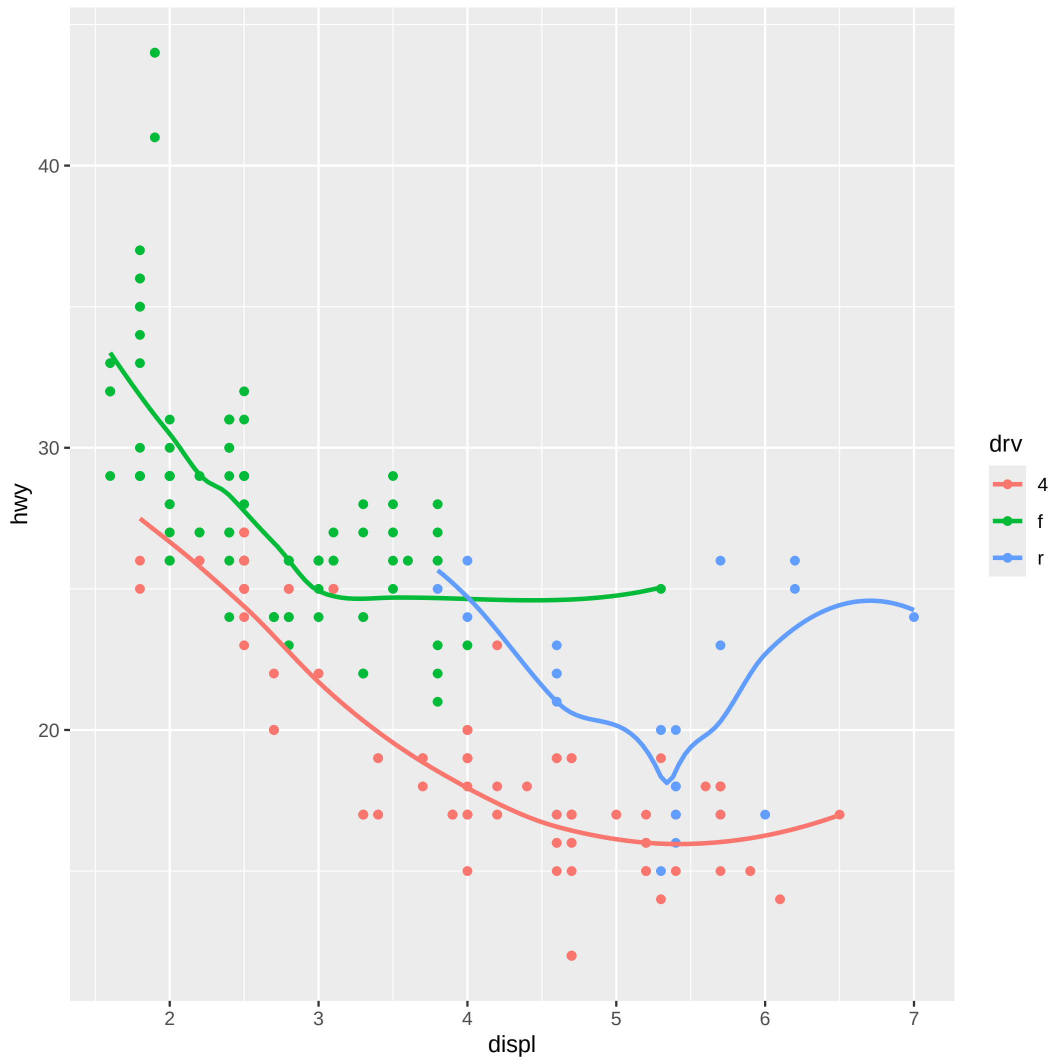
6. This is to avoid the potentially english labels from overlapping with the adjacent labels.

(b)

1. We would use the `geom_line,geom_boxplot,geom_histogram,geom_area` respectively.
2. I think we will see two plots on the dataset `mpg` with the `x` and `y` axis being `displ` and `hwy` respectively, each grouped (with a legend) according to `drv` using colour, layered on top of eachother: a scatter plot and a smooth line plot which will not have a confidense interval (due to `se = FALSE`, its on by default.) - other than that I think the two layers will use the same colours for each class and hence there will only be one legend. Now we run the code:

```r
ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = drv)) +
geom_point() +
geom_smooth(se = FALSE)
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

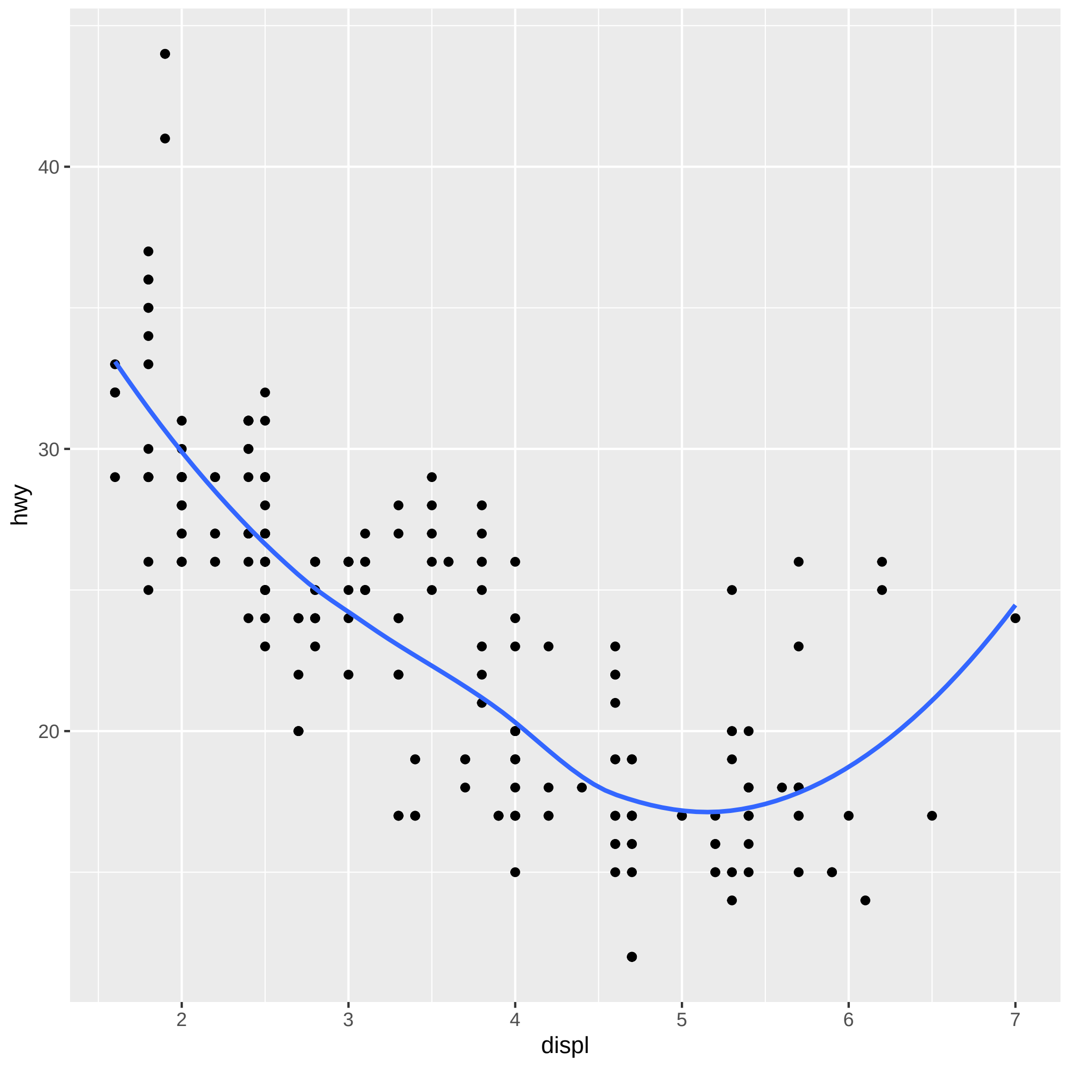


my prediction was not wrong but I didn't think that the legends would overlap as in the line and dot would both be included in the "legend box"(s).

3. `show.legend = FALSE` stops a legend from being displayed in the plot. If this is removed then a legend appears if it was supposed to anyways and doesn't if it wasn't supposed to (for example, when using `group`). It was used earlier to remove the legend because ggplot2 will automatically group the data for these geoms whenever you map `color` to a discrete variable and make a legend as well, using `show.legend = FALSE` stops the legend from being displayed.
4. The `se` arguement to `geom_smooth` toggles inclusion for the confidence interval of data being plotted.
5. These will produce the same plots as in one we have a global mapping that gets applied to both geoms and in the other we use the same mapping for each geom but locally.
6. The R code is written going towards the right and downwards:

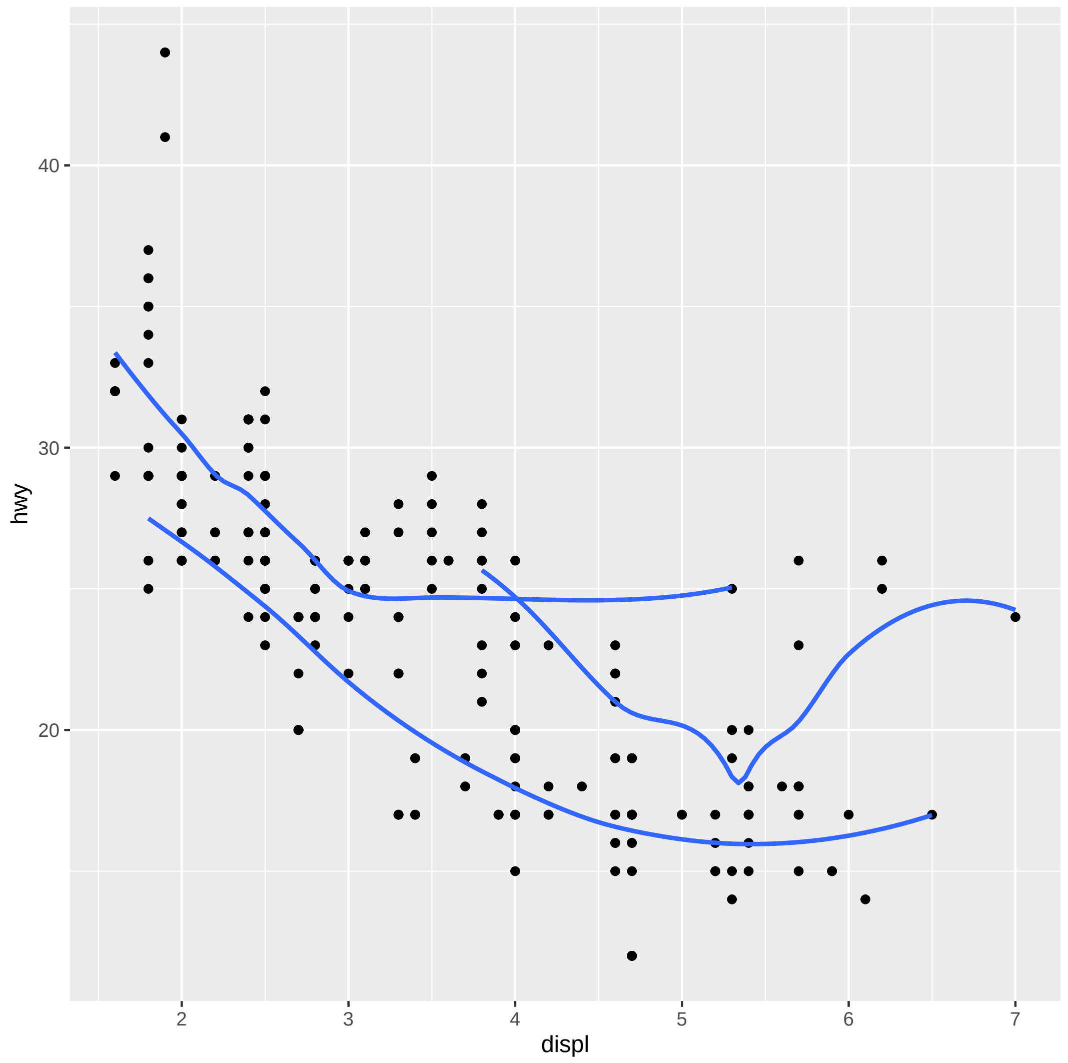
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
geom_point() +
geom_smooth(se = FALSE)

## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



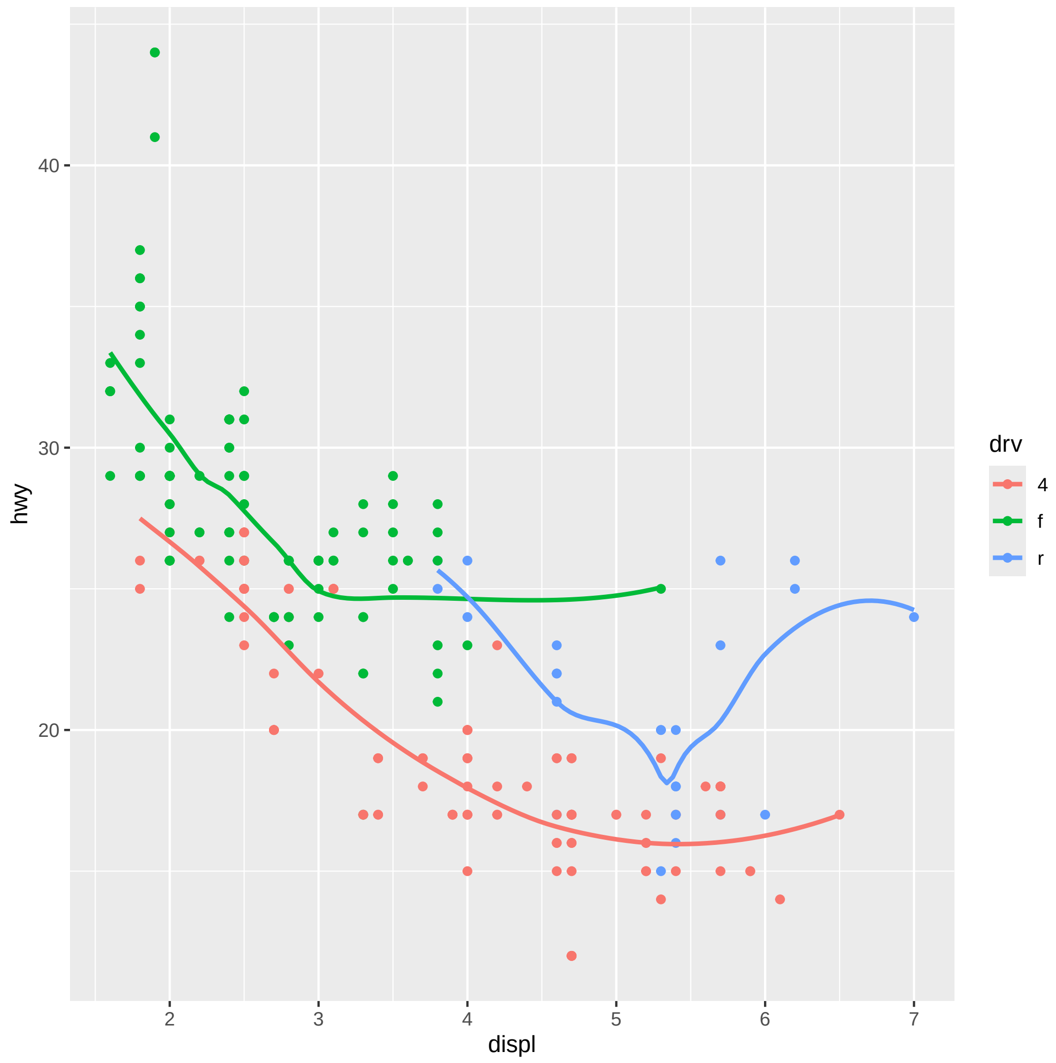
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
geom_point() +
geom_smooth(mapping = aes(group = drv), se = FALSE)

## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



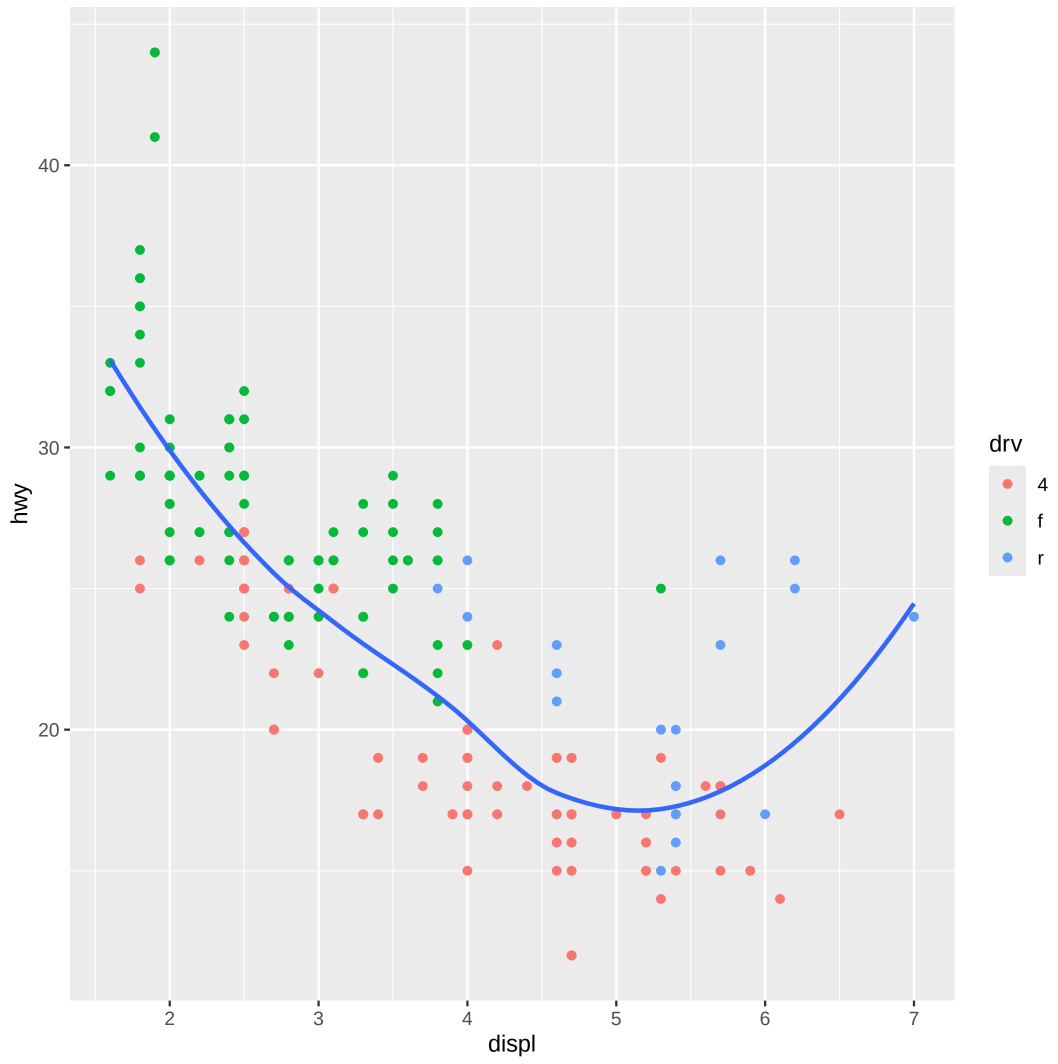
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = drv)) +
geom_point() +
geom_smooth(se = FALSE)

## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



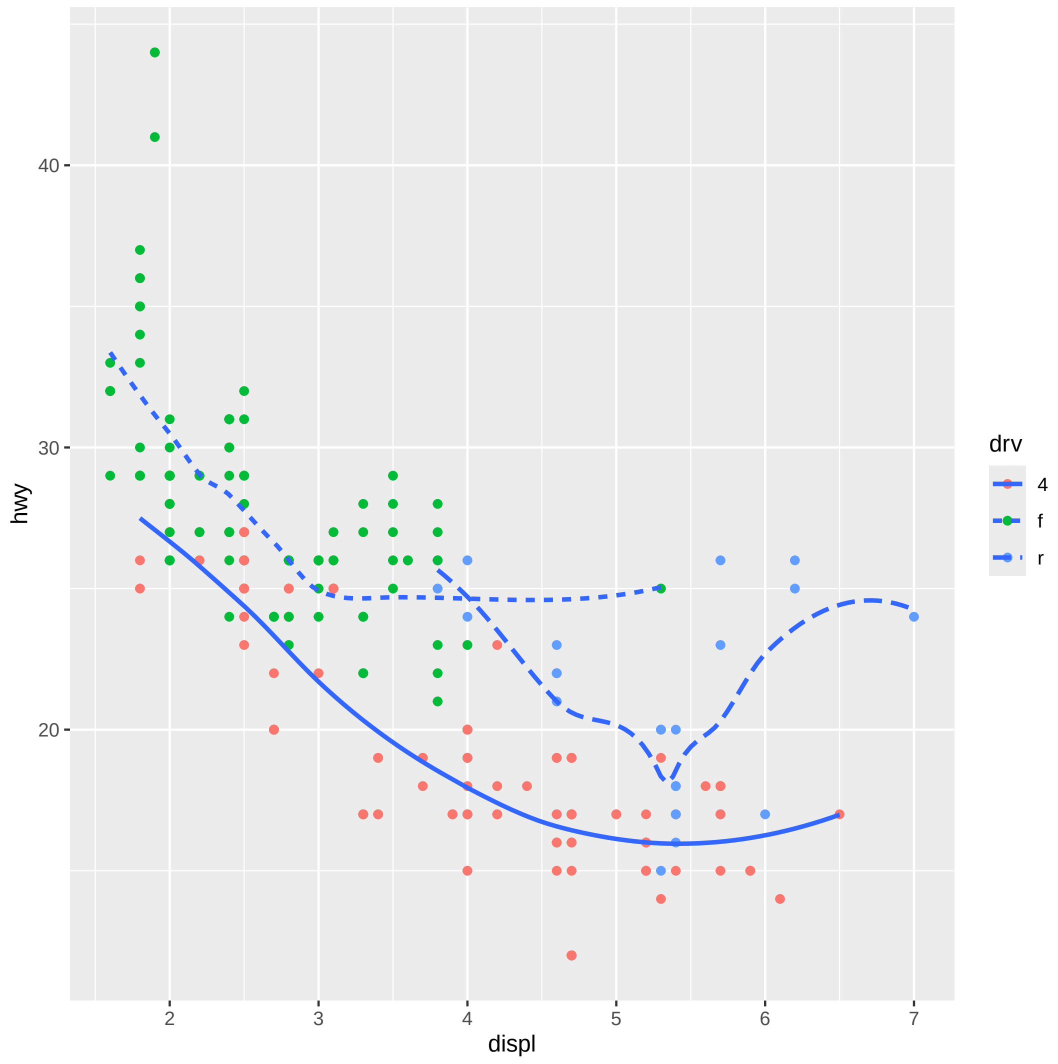
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
geom_point(mapping = aes(color = drv)) +
geom_smooth(se = FALSE)

## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

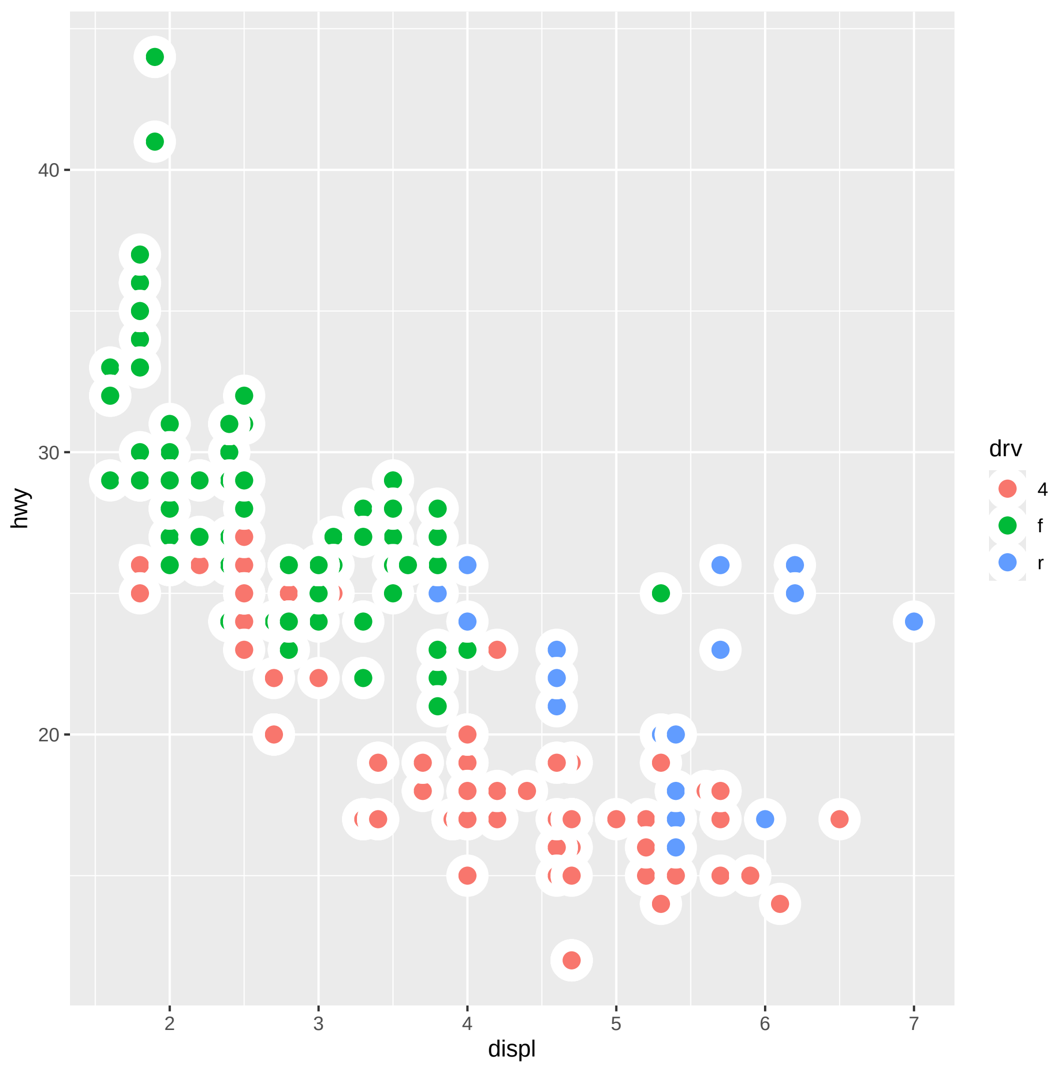


```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
geom_point(mapping = aes(color = drv)) +
geom_smooth(mapping = aes(linetype = drv),se = FALSE)

## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

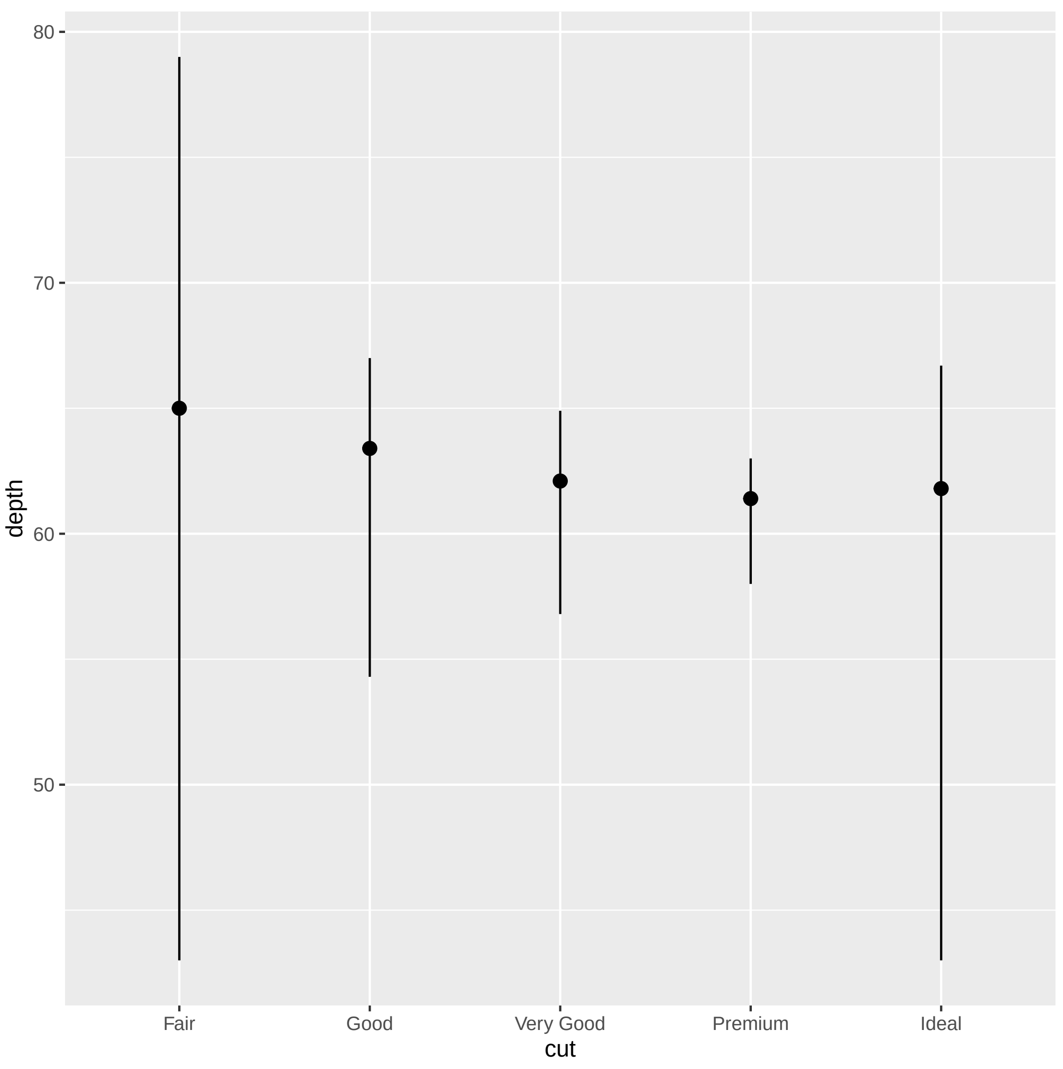


```
ggplot(data = mpg) +
geom_point(mapping = aes(x = displ, y = hwy, fill = drv),
 shape = 21, size = 4, stroke = 4, color = "white")
```



(c) 1. The default geom associated with `stat_summary()` is `geom_pointrange`. We could rewrite it as follows, by calling the specific stat `summary` to do the necessary calculations.

```
ggplot(data = diamonds) +
geom_pointrange(mapping = aes(x = cut, y = depth),
stat = "summary",
fun = median,
fun.min = min,
fun.max = max)
```



2. `geom_col()` creates a bar chart where the height of the bars represents specific values provided in the data, requiring you to map a variable to `y` (height). The difference is in the default statistical transformation as `geom_bar()` calculates the bar height automatically by counting rows (with `stat = "count"`), whereas `geom_col()` expects the heights to be pre-calculated and present in the data (as it uses `stat = "identity"` which does nothing extra).
3. - `geom_bar` ↔ `stat_count`
   - `geom_boxplot` ↔ `stat_boxplot`
   - `geom_density` ↔ `stat_density`
   - `geom_histogram` ↔ `stat_bin`
   - `geom_smooth` ↔ `stat_smooth`
   - `geom_quantile` ↔ `stat_quantile`
   - `geom_bin2d` ↔ `stat_bin2d`

   Almost all of them share the same names and they depend on eachother, for example: `geom_bar` uses `stat_count` to calculate the heights neccesary for the bars and `stat_count` uses `geom_bar` to plot the results it generates.

4. `stat_smooth` computes the variables: `y` the predicted value, `ymin` the lower bound for the confidence interval, `ymax` the upper bound of the same, `se` the standard error and has parameters: `method` for the smoothing function to use (An example being `loess`), `fo2rmula` for the equation used for smoothing, `se` to specify whether to display the confidence interval, `level` to specify the confidence level, `span` (only valid for `method = loess`) to control how much the line wiggles (almost as if bounding the variaton of the curve), `n` the number of points at which it has to evaluate the smoother.

5. By default, `ggplot2` treats each x-axis category as its own group, meaning proportions are calculated within that category (summing to 1). Setting `group = 1`(we could have writte `group = "meaow"` too, everything just needs to be dumped into a single bucket so that the proportions aren't calculated locally but rather globally on this one group which has it all) overrides this, putting all data into a single group so that proportions are calculated relative to the total dataset.

(d)

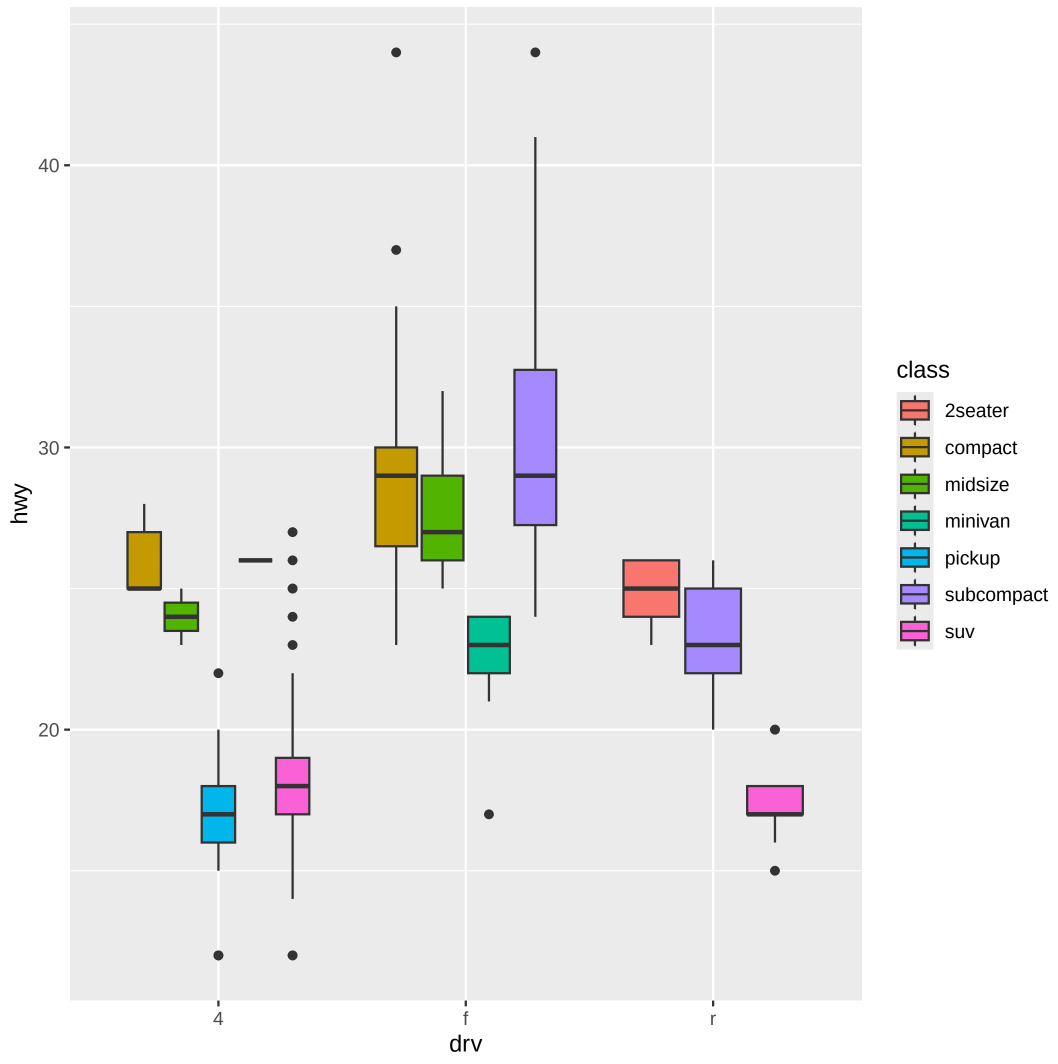
1. There is a lot of mass around (18,26) which is not visible through the scatter plot. This can be fixed by just using a jitter plot instead which makes this visible.

2. The parameters `width, height` control the amount of horizontal and vertical jitter respectively.

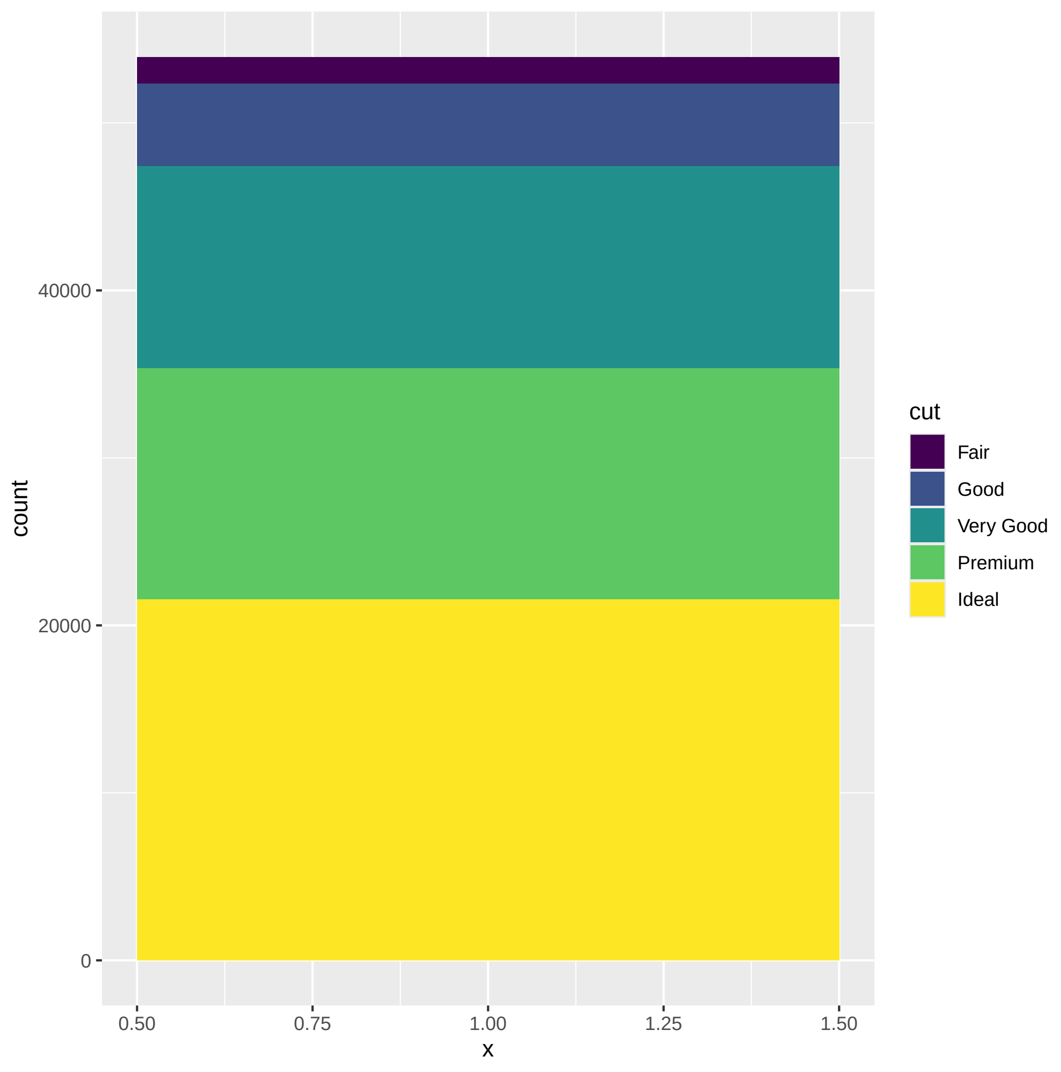
3. The jitter plot aims to make every data point visible, be it ever so slightly inaccurat due to the random noise whereas count plots add more information to the plot by effectively labelling each data point with the number of data points in the same location. They both have the same goal of helping us in the case of overplotting but they do it in vastly different ways, jitter plots convey mass through no. of points we visually see around some location whereas count plots explicitly mention the mass at each location.

4. The default position adjustment for boxplots is `dodge2`, this moves the boxplots along the x-axis ensuring that they do not overlap (hence the name "dodge"). Here's a demonstration with the `mpg` dataset:

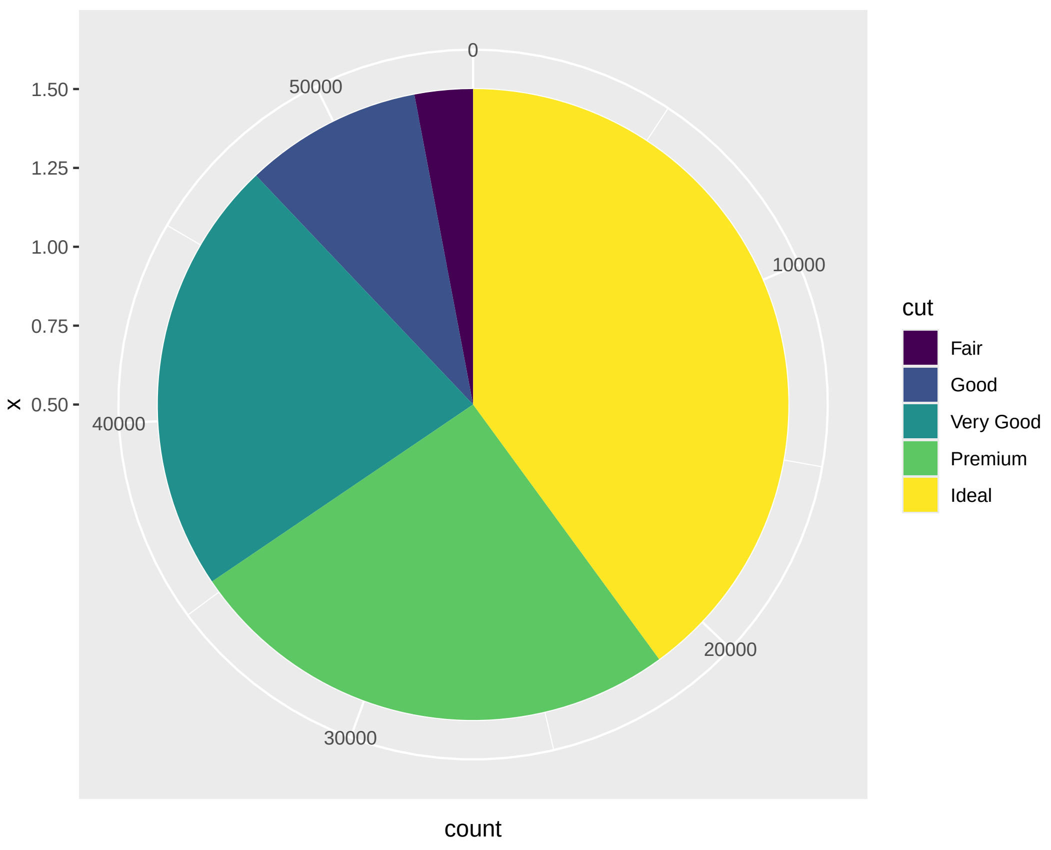
```r
ggplot(data = mpg) +
geom_boxplot(mapping = aes(x = drv, y = hwy, fill = class))
```



(e) 1.
```
ggplot(data = diamonds) +
geom_bar(mapping = aes(x = 1, fill = cut), width = 1)
```



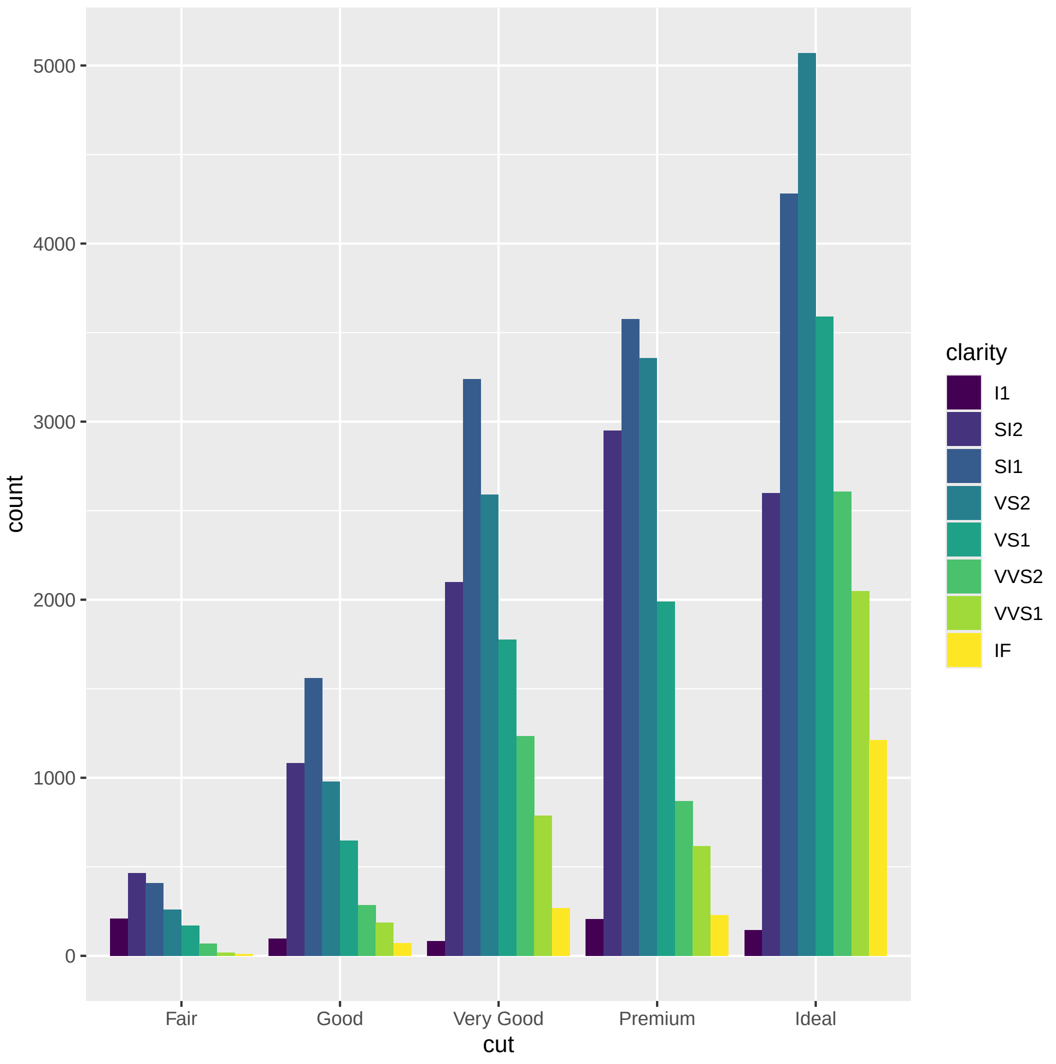
```
ggplot(data = diamonds) +
geom_bar(mapping = aes(x = 1, fill = cut), width = 1) +
coord_polar(theta = "y")
```



2. `labs` is used to add the labels to a plot hence making it more accessible to others. It can also be used to add tags and captions etc. to differentiate between different plots.
3. `coord_map` projects a portion of the earth, which is approximately spherical, onto a flat 2D plane using any projection define by the 'mapproj' package. Map projections do not, in general, preserve straight lines, so this **requires considerable computation** whereas `coord_quickmap` is a **quick approximation** that does preserve straight lines. It works best for smaller areas closer to the equator.
4. `coord_fixed` lets us see exactly how the parameters grow with respect to eachother as here one unit on the x-axis and has the same visual length as that on the y-axis. `geom_abline` is used to draw a straight line that stretches infinitely, on the plot. It takes slope and intercept as input along with some other aesthetic parameters like color, linetype etc.

(f) 1. `diamonds` is a dataset of containing the prices and other attributes like cut, depth etc. of almost 54000 diamonds. This has 10 variables namely: `price, carat, cut, colour, clarity, depth`.

2.
   ```
   ggplot(data = diamonds) +
   geom_bar(mapping = aes(x = cut, fill = clarity), position = "dodge")
   ```



3. The total number of diamonds increases as the quality of the cut increases across almost all the cuts and the number of diamonds is mostly concentrated towards the sl1,vs2,vs1 clarity so there are more diamonds of "slightly to very slightly included" clarity as per GIA terminology.