

Análisis de Datos

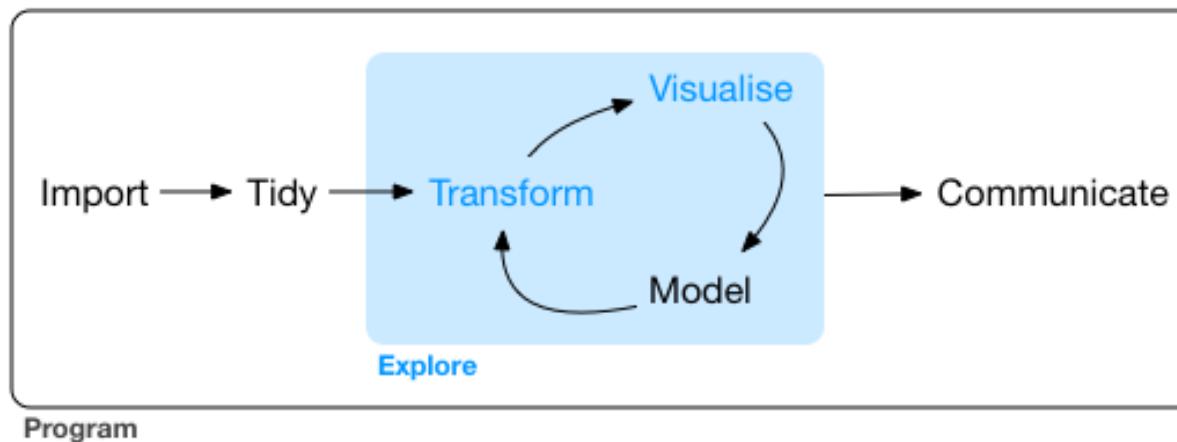
Tema 2 - Análisis Exploratorio de los Datos

2.1 Visualización

Roi Naveiro

Análisis Exploratorio de los Datos

- El arte de observar los datos, generar hipótesis y testearlas.
- **Objetivo:** generar preguntas prometedoras para, posteriormente, explorarlas en mayor profundidad



tidyverse

Queremos analizar los datos de forma **reproducible**



El objeto tibble

El objeto tibble

- Todos los paquetes de tidyverse trabajan con tibbles
- El objeto tibble es similar al data frame que hemos visto
- No obstante, incluye algunas novedades que lo hacen especialmente conveniente para el análisis de datos.
- Antes de empezar, debemos cargar la librería **tidyverse**

```
library(tidyverse)
```

Creación de tibbles

- Podemos crear un tibble desde un dataframe

```
as_tibble(iris)
```

```
## # A tibble: 150 × 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##       <dbl>      <dbl>       <dbl>       <dbl>   <fct>
## 1         5.1        3.5        1.4        0.2  setosa
## 2         4.9        3.0        1.4        0.2  setosa
## 3         4.7        3.2        1.3        0.2  setosa
## 4         4.6        3.1        1.5        0.2  setosa
## 5         5.0        3.6        1.4        0.2  setosa
## 6         5.4        3.9        1.7        0.4  setosa
## 7         4.6        3.4        1.4        0.3  setosa
## 8         5.0        3.4        1.5        0.2  setosa
## 9         4.4        2.9        1.4        0.2  setosa
## 10        4.9        3.1        1.5        0.1 setosa
## # ... with 140 more rows
```

Creación de tibbles

- Podemos crear un tibble desde vectores, de manera similar a cómo hicimos con dataframes

```
dt <- tibble(  
  p = c("A", "B", "C", "D", "E"),  
  x = 1:5,  
  y = 2,  
  z = x + y  
)  
dt
```

```
## # A tibble: 5 × 4  
##   p      x     y     z  
##   <chr> <int> <dbl> <dbl>  
## 1 A        1     2     3  
## 2 B        2     2     4  
## 3 C        3     2     5  
## 4 D        4     2     6  
## 5 E        5     2     7
```

Creación de tibbles

- Podemos dar nombres complejos a las columnas de un tibble

```
dt <- tibble(  
  `Número de :)` = 1:5,  
  ` ` = 2,  
  `300` = 300  
)  
dt
```

```
## # A tibble: 5 × 3  
##   `Número de :)` ` ` `300`  
##       <int> <dbl> <dbl>  
## 1           1     2    300  
## 2           2     2    300  
## 3           3     2    300  
## 4           4     2    300  
## 5           5     2    300
```

Tibbles vs Data Frames

Printing

Los tibble tienen un método **print** refinado que por defecto solo muestra las 10 primeras filas

```
iris
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
## 7          4.6         3.4          1.4         0.3  setosa
## 8          5.0         3.4          1.5         0.2  setosa
## 9          4.4         2.9          1.4         0.2  setosa
## 10         4.9         3.1          1.5         0.1  setosa
## 11         5.4         3.7          1.5         0.2  setosa
## 12         4.8         3.4          1.6         0.2  setosa
## 13         4.8         3.0          1.4         0.1  setosa
## 14         4.3         3.0          1.1         0.1  setosa
## 15         5.8         4.0          1.2         0.2  setosa
## 16         5.7         4.4          1.5         0.4  setosa
```

Tibbles vs Data Frames

Printing

Los tibble tienen un método **print** refinado que por defecto solo muestra las 10 primeras filas

```
dt <- as_tibble(iris)
```

Tibbles vs Data Frames

Printing

Esto es así para no sobrecargar la consola con los prints. ¿Y si necesitamos más de 10 filas? ¡PIPES!

```
dt %>% print(n=11, width=Inf)
```

```
## # A tibble: 150 × 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##       <dbl>      <dbl>      <dbl>      <dbl> <fct>
## 1         5.1        3.5        1.4        0.2 setosa
## 2         4.9        3.0        1.4        0.2 setosa
## 3         4.7        3.2        1.3        0.2 setosa
## 4         4.6        3.1        1.5        0.2 setosa
## 5         5.0        3.6        1.4        0.2 setosa
## 6         5.4        3.9        1.7        0.4 setosa
## 7         4.6        3.4        1.4        0.3 setosa
## 8         5.0        3.4        1.5        0.2 setosa
## 9         4.4        2.9        1.4        0.2 setosa
## 10        4.9        3.1        1.5        0.1 setosa
## 11        5.4        3.7        1.5        0.2 setosa
## # ... with 139 more rows
```

Tibbles vs Data Frames

Printing

En RStudio, esto nos permite explorar el dataframe en una pantalla diferente.

```
dt %>% View()
```

Tibbles vs Data Frames

Subsetting

Para seleccionar una única variable usar `$` (por nombre) o `[[]` (por nombre o posición)

```
# Extraer por nombre
dt$Species[1:10]
```

```
## [1] setosa setosa setosa setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

```
# Extraer por posicion
dt[[5]][1:10]
```

```
## [1] setosa setosa setosa setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

Tibbles vs Data Frames

Subsetting

También en pipe

```
dt %>% .$Species
```

```
## [1] setosa    setosa    setosa    setosa    setosa    setosa    setosa
## [7] setosa    setosa    setosa    setosa    setosa    setosa    setosa
## [13] setosa   setosa    setosa    setosa    setosa    setosa    setosa
## [19] setosa   setosa    setosa    setosa    setosa    setosa    setosa
## [25] setosa   setosa    setosa    setosa    setosa    setosa    setosa
## [31] setosa   setosa    setosa    setosa    setosa    setosa    setosa
## [37] setosa   setosa    setosa    setosa    setosa    setosa    setosa
## [43] setosa   setosa    setosa    setosa    setosa    setosa    setosa
## [49] setosa   setosa    versicolor versicolor versicolor versicolor
## [55] versicolor versicolor versicolor versicolor versicolor versicolor
## [61] versicolor versicolor versicolor versicolor versicolor versicolor
## [67] versicolor versicolor versicolor versicolor versicolor versicolor
## [73] versicolor versicolor versicolor versicolor versicolor versicolor
## [79] versicolor versicolor versicolor versicolor versicolor versicolor
## [85] versicolor versicolor versicolor versicolor versicolor versicolor
## [91] versicolor versicolor versicolor versicolor versicolor versicolor
## [97] versicolor versicolor versicolor versicolor virginica virginica
## [103] virginica virginica virginica virginica virginica virginica
```

Tibbles vs Data Frames

Convertir de nuevo a data frame

```
class( as.data.frame(dt) )  
  
## [1] "data.frame"
```

Visualización

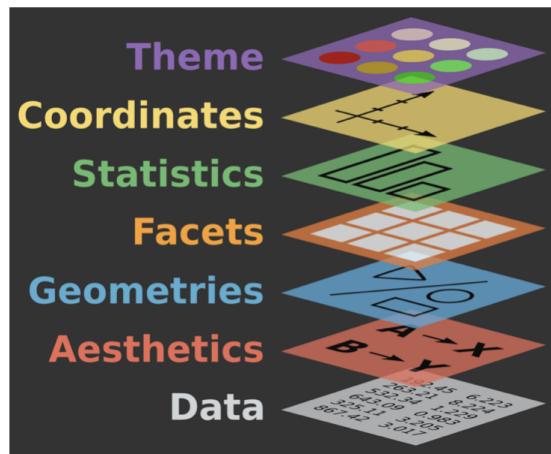
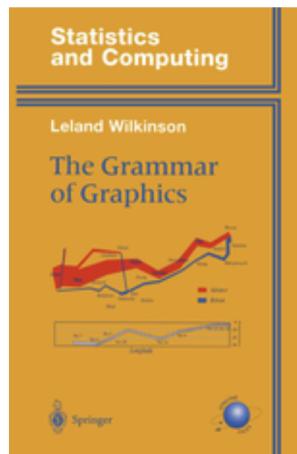
Visualización

Creación y estudio de representaciones visuales de los datos

- En esta parte del curso nos centraremos en aprender a manejar **ggplot2** un paquete de tidyverse realmente útil para la visualización
- EDA incluye, además de visualización, la manipulación de datos
- EDA combina la visualización y la manipulación con la curiosidad y el escepticismo de cada uno para preguntar y responder cuestiones interesantes acerca de los datos

ggplot2

- ggplot2 es el paquete de visualización de tidyverse
- gg significa Grammar of Graphics
- Está inspirado en el libro Grammar of Graphics de Leland Wilkinson



GG: herramienta que permite describir de forma secuencial los componentes de un gráfico

Visualización

Primero cargamos las librerías necesarias

```
library(tidyverse)
library(gapminder)
```

Ojeamos los datos

```
glimpse(gapminder)
```

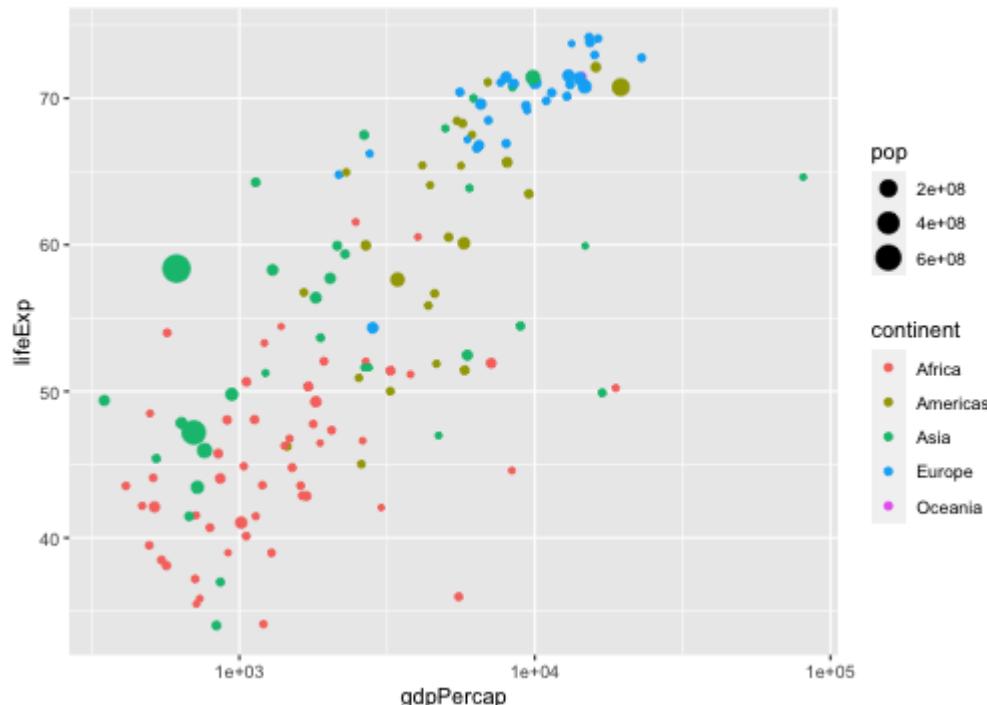
```
## #> #> Rows: 1,704
## #> #> Columns: 6
## #> #> $ country    <fct> "Afghanistan", "Afghanistan", "Afghanist...
## #> #> $ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia...
## #> #> $ year       <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982...
## #> #> $ lifeExp    <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, ...
## #> #> $ pop        <int> 8425333, 9240934, 10267083, 11537966, 13...
## #> #> $ gdpPercap   <dbl> 779.4453, 820.8530, 853.1007, 836.1971, ...
```

Visualización: Visualizando la relación entre PIB per capita y esperanza de vida

Primera visualización

¿Qué hace cada línea?

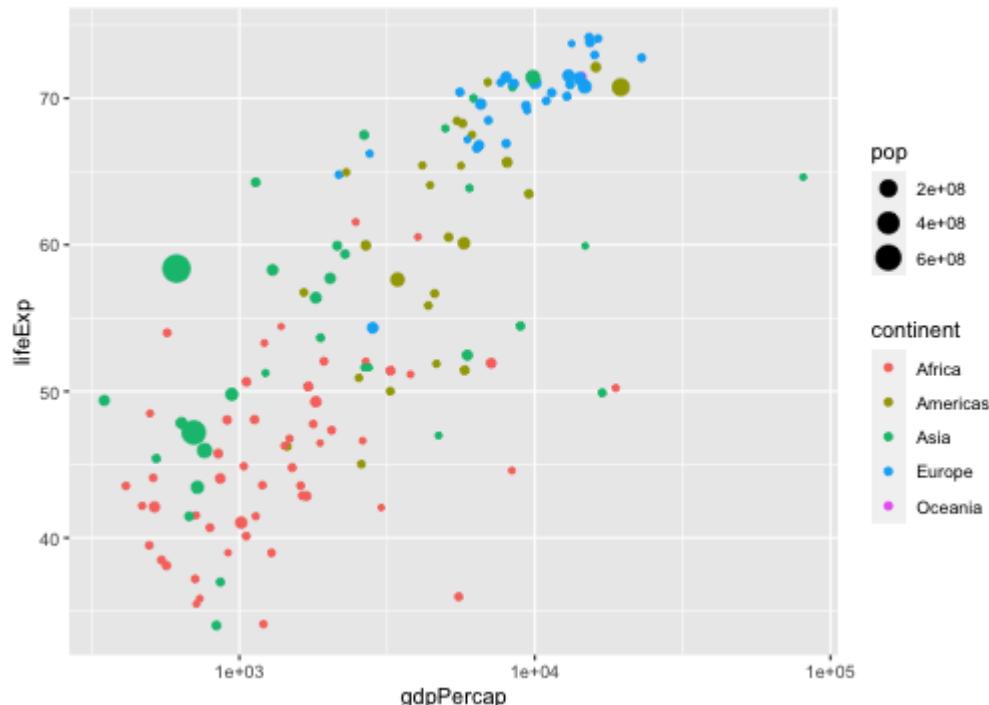
```
gapminder67 <- gapminder %>% filter(year == "1967")
ggplot(gapminder67,
       aes(gdpPercap, lifeExp, size = pop, color = continent)) +
  geom_point() +
  scale_x_log10() # convert to log scale
```



Primera visualización

¿Qué hace cada línea?

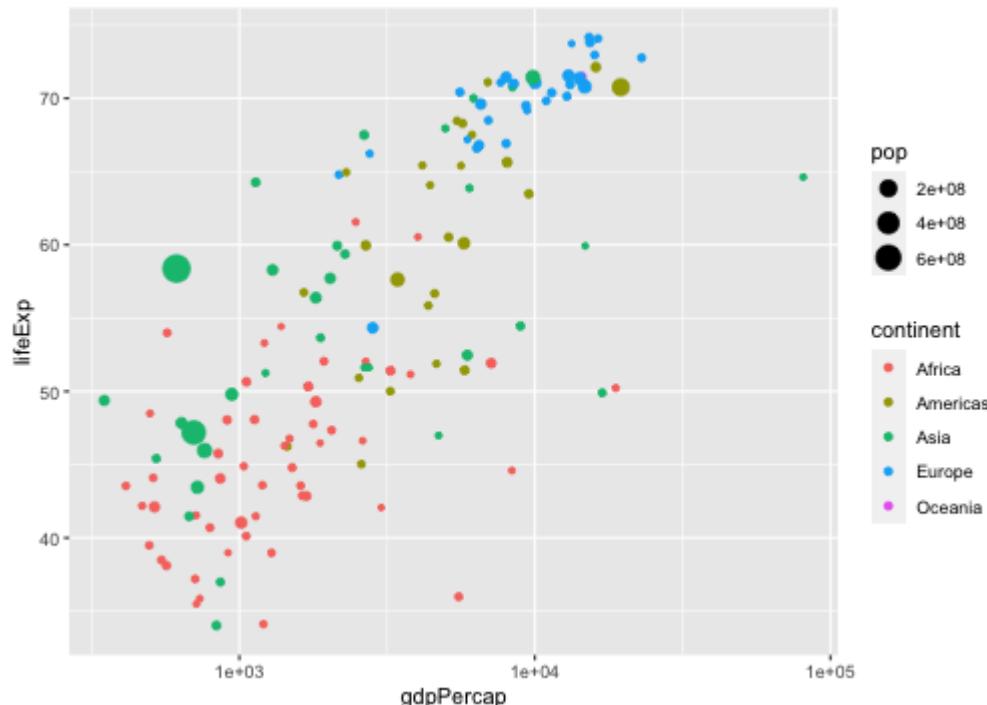
```
gapminder67 <- gapminder %>% filter(year == "1967")
ggplot(gapminder67,
       aes(gdpPercap, lifeExp, size = pop, color = continent)) +
  geom_point() +
  scale_x_log10() # convert to log scale
```



Primera visualización

¿Qué hace cada línea?

```
gapminder67 <- gapminder %>% filter(year == "1967")
ggplot(gapminder67,
       aes(gdpPercap, lifeExp, size = pop, color = continent)) +
  geom_point() +
  scale_x_log10()
```



Funcionamiento **ggplot2**

- La estructura del código de **ggplot2** es la siguiente

```
ggplot(data = [dataset],  
       mapping = aes(x = [x-variable], y = [y-variable])) +  
       geom_xxx() +  
       otras opciones
```

- El **mapping** define como las variables son mapeadas a propiedades visuales del gráfico.

Documentación ggplot2.tidyverse.org

Funcionamiento ggplot2

- Alternativamente

```
ggplot(data = [dataset]) +  
  geom_xxx( mapping = aes(x = [x-variable], y = [y-variable])) +  
  other options
```

Paso a Paso

Ojeamos los datos

```
glimpse(gapminder)
```

```
## Rows: 1,704
## Columns: 6
## $ country    <fct> "Afghanistan", "Afghanistan", "Afghanist...
## $ continent   <fct> Asia, Asia, Asia, Asia, Asia, Asia...
## $ year        <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982...
## $ lifeExp     <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, ...
## $ pop         <int> 8425333, 9240934, 10267083, 11537966, 13...
## $ gdpPercap   <dbl> 779.4453, 820.8530, 853.1007, 836.1971, ...
```

Miramos **?gapminder**

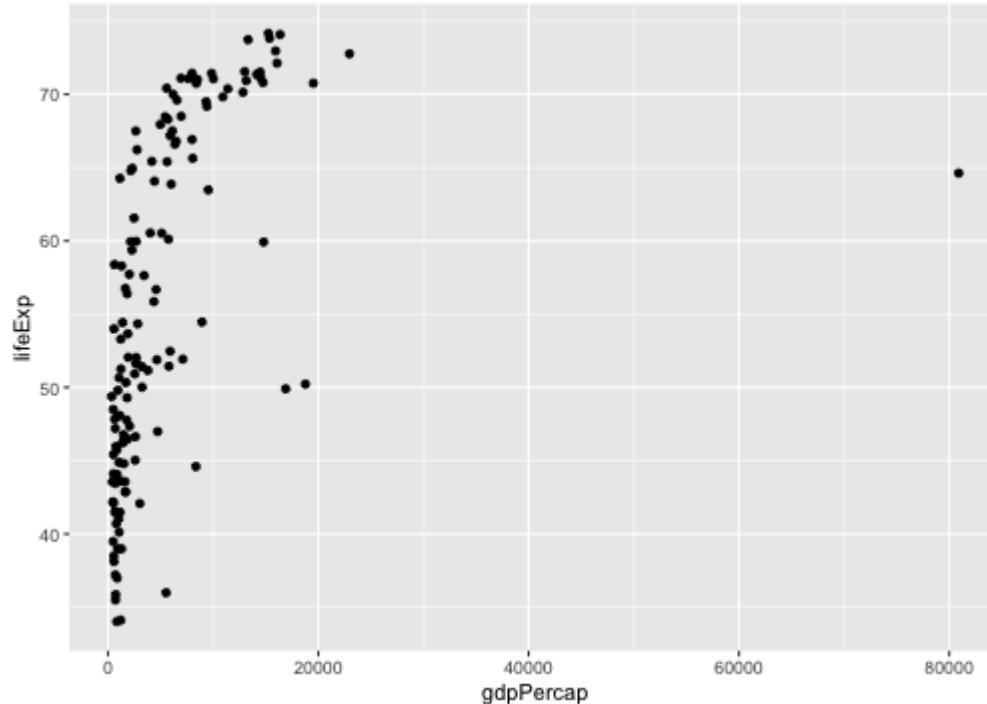
Paso a Paso

```
gapminder67 <- gapminder %>% filter(year == "1967")  
gapminder67
```

```
## # A tibble: 142 × 6  
##   country    continent  year lifeExp      pop gdpPercap  
##   <fct>      <fct>     <int>  <dbl>      <int>     <dbl>  
## 1 Afghanistan Asia      1967    34.0  11537966     836.  
## 2 Albania      Europe    1967    66.2  1984060    2760.  
## 3 Algeria      Africa    1967    51.4  12760499    3247.  
## 4 Angola       Africa    1967    36.0  5247469    5523.  
## 5 Argentina    Americas  1967    65.6  22934225    8053.  
## 6 Australia    Oceania   1967    71.1  11872264   14526.  
## 7 Austria      Europe    1967    70.1  7376998    12835.  
## 8 Bahrain      Asia      1967    59.9  202182    14805.  
## 9 Bangladesh   Asia      1967    43.5  62821884     721.  
## 10 Belgium     Europe   1967    70.9  9556500    13149.  
## # ... with 132 more rows
```

Paso a Paso

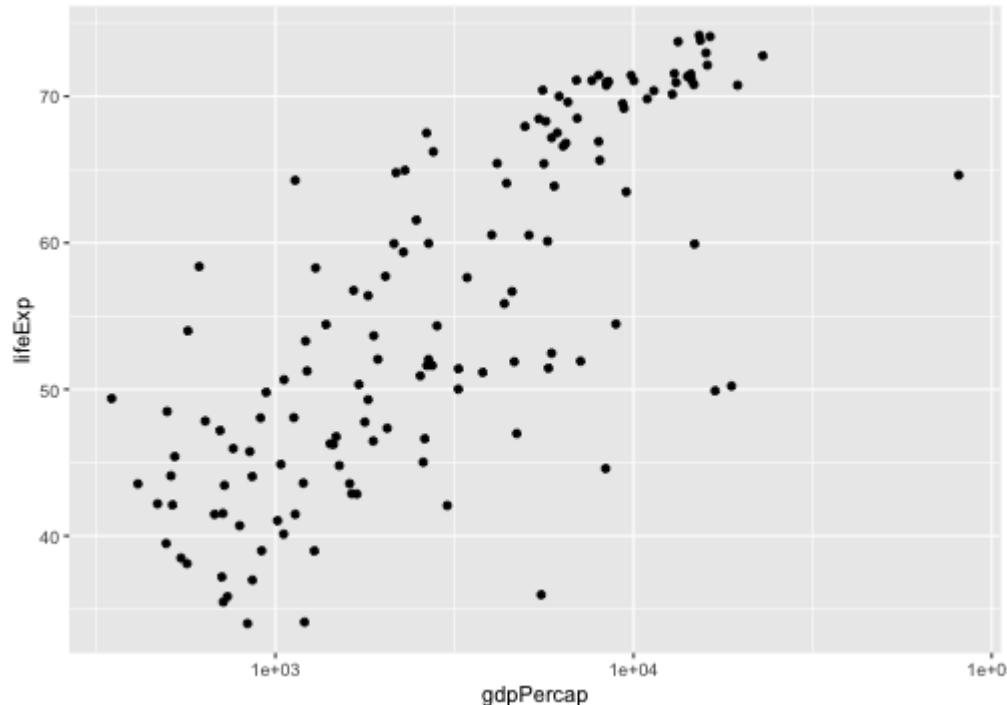
```
ggplot(gapminder67, aes(x = gdpPercap, y = lifeExp) ) +  
  geom_point()
```



¿Cómo describirías la relación entre pib per capita y esperanza de vida?

Paso a Paso

```
ggplot(gapminder67, aes(x = gdpPercap, y = lifeExp) ) +  
  geom_point() +  
  scale_x_log10()
```



¿Cómo describirías la relación entre pib per capita y esperanza de vida?

Variables adicionales

Es posible que queramos representar más de dos variables. Podemos hacerlo a través de otros elementos del gráfico (en mapping usamos solo x e y, podemos añadir más)

- **aesthetics:**
 - shape
 - color
 - size
 - alpha (transparencia)
- **faceting:** varias gráficas

Visualización: Aesthetics

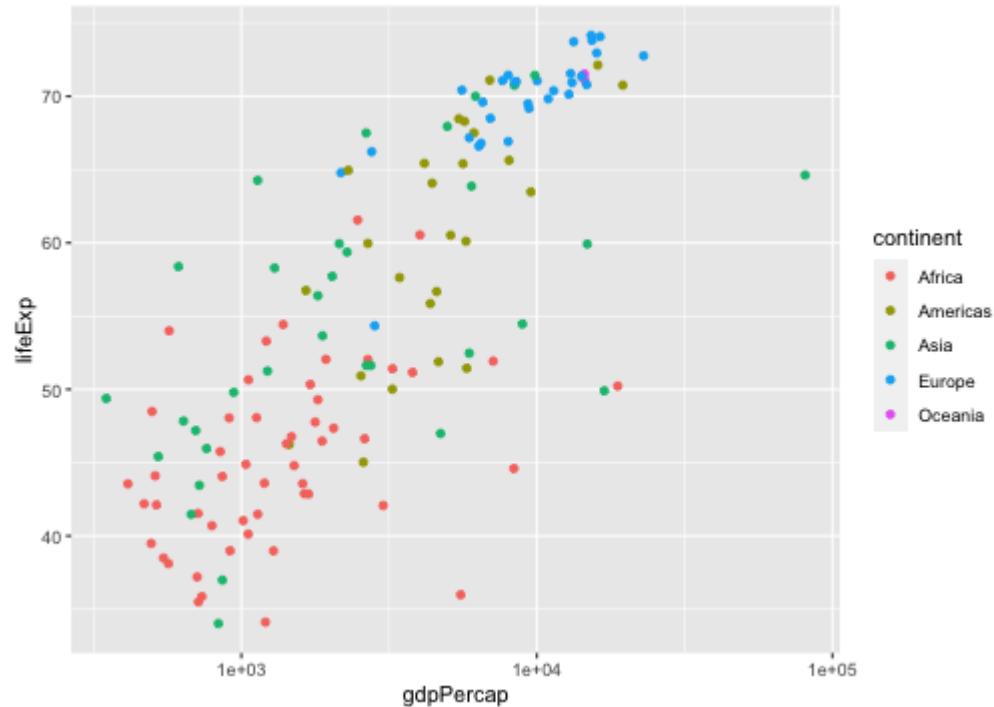
Aesthetics

Las siguientes características visuales del gráfico pueden ser utilizadas para representar una variable extra

- shape
- color
- size
- alpha (transparencia)

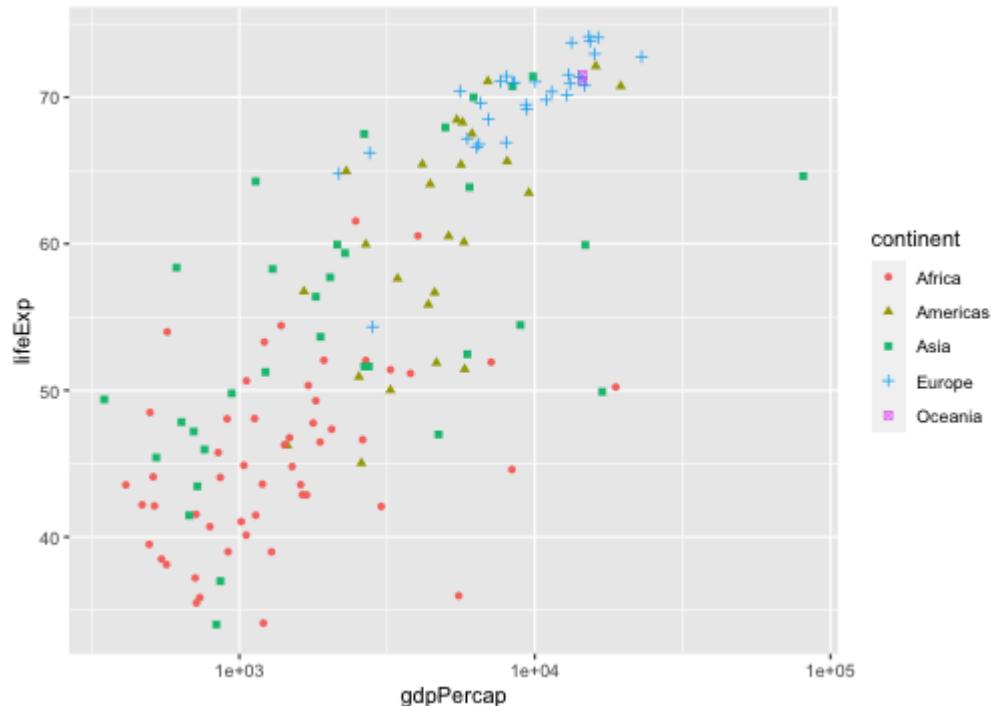
Representamos el continente

```
ggplot(gapminder67, mapping = aes(x = gdpPercap, y = lifeExp, color = continent)) +  
  geom_point() +  
  scale_x_log10()
```



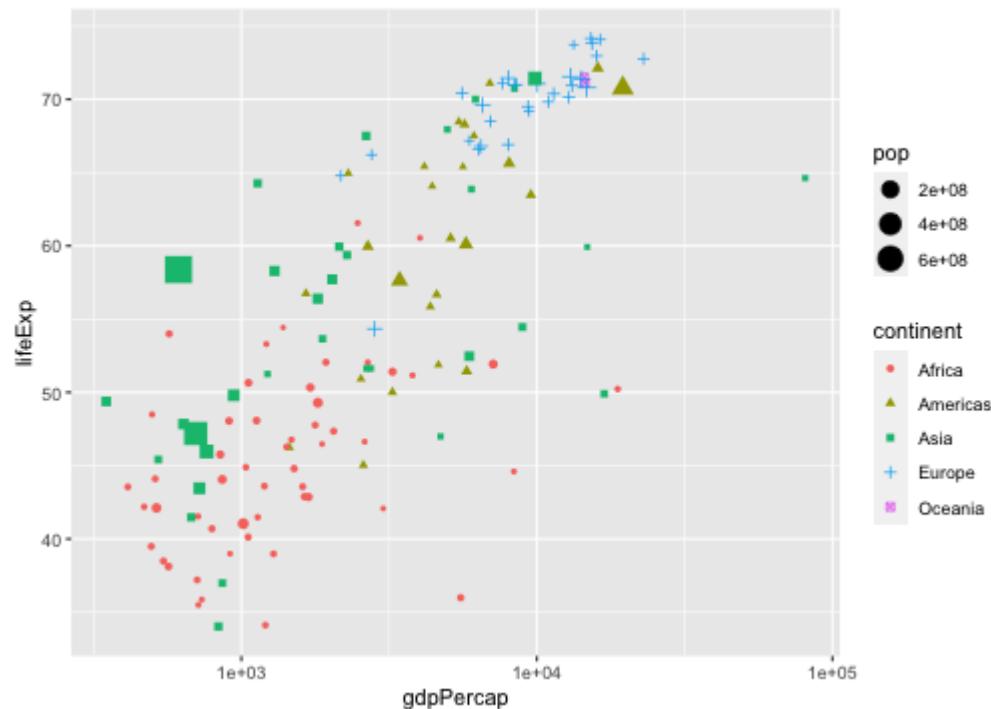
Representamos el continente

```
ggplot(gapminder67, mapping = aes(x = gdpPercap, y = lifeExp, color = continent,  
shape = continent) ) +  
  geom_point() +  
  scale_x_log10()
```



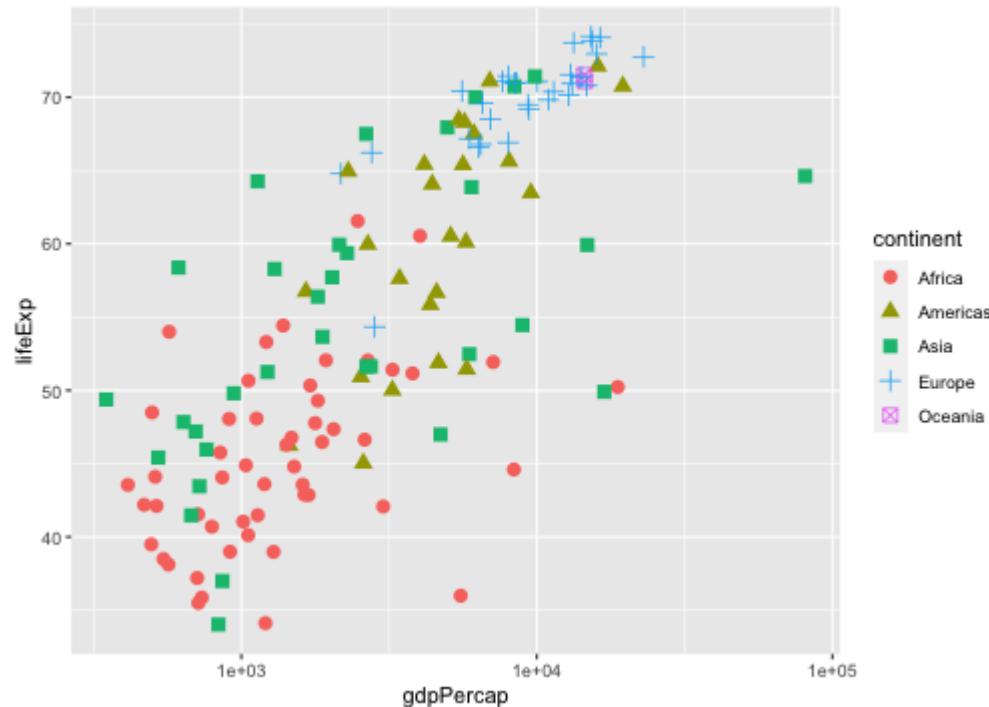
Representamos el continente y la población

```
ggplot(gapminder67, mapping = aes(x = gdpPercap, y = lifeExp, color = continent,  
shape = continent, size = pop) ) +  
  geom_point() +  
  scale_x_log10()
```



Representamos el continente y la población

```
ggplot(gapminder67, mapping = aes(x = gdpPercap, y = lifeExp, color = continent,  
shape = continent, size = pop) ) +  
  geom_point(size=3) +  
  scale_x_log10()
```



Resumen

- Variables continuas
- Variables discretas

aesthetics	discreta	continua
color	diferentes colores	gradiente
size	pasos discretos	mapeo linear entre variable y radio
shape	diferentes formas	no funciona

aes para representar variables usando características del gráfico

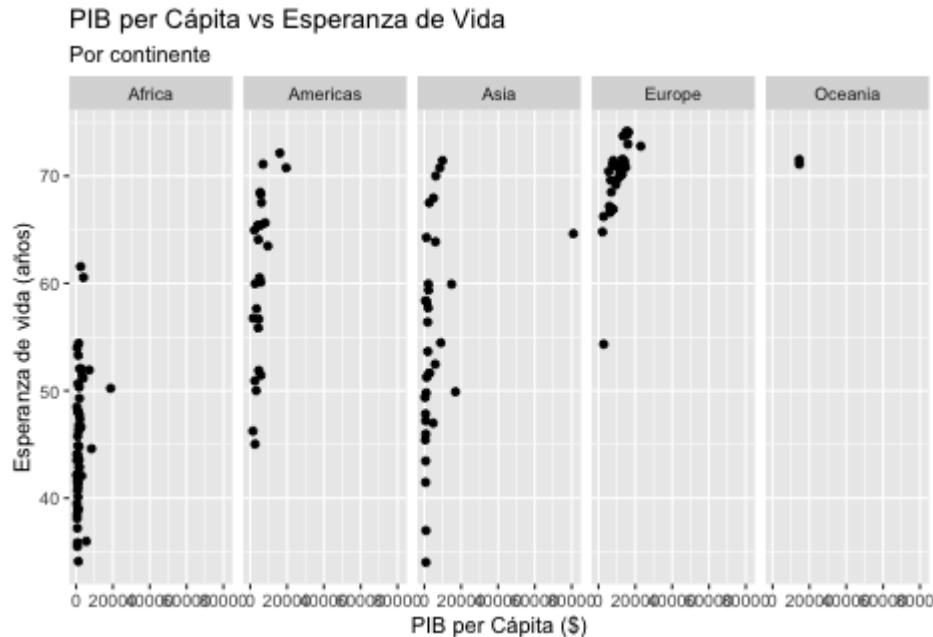
geom_xxx para definir característica del gráfico

Visualización: Faceting

Faceting

Múltiples gráficas en función de variable

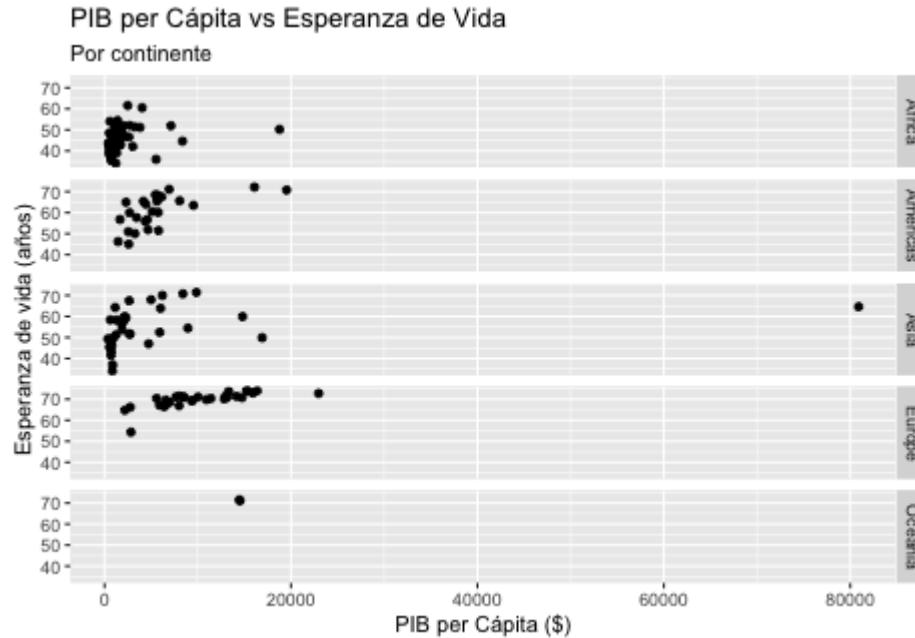
```
ggplot(data = gapminder67, aes(x = gdpPercap, y = lifeExp)) +  
  geom_point() +  
  labs(title = "PIB per Cápita vs Esperanza de Vida",  
       subtitle = "Por continente",  
       x = "PIB per Cápita ($)", y = "Esperanza de vida (años)") +  
  facet_grid(. ~ continent)
```



Faceting

Múltiples gráficas en función de variable

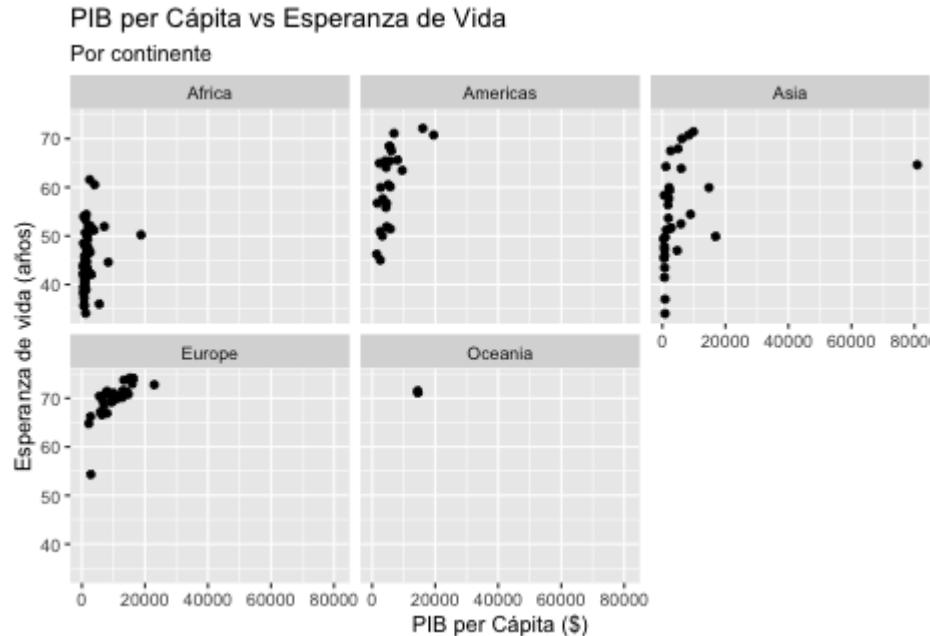
```
ggplot(data = gapminder67, aes(x = gdpPercap, y = lifeExp)) +  
  geom_point() +  
  labs(title = "PIB per Cápita vs Esperanza de Vida",  
       subtitle = "Por continente",  
       x = "PIB per Cápita ($)", y = "Esperanza de vida (años)") +  
  facet_grid(continent ~ .)
```



Faceting

Múltiples gráficas en función de variable

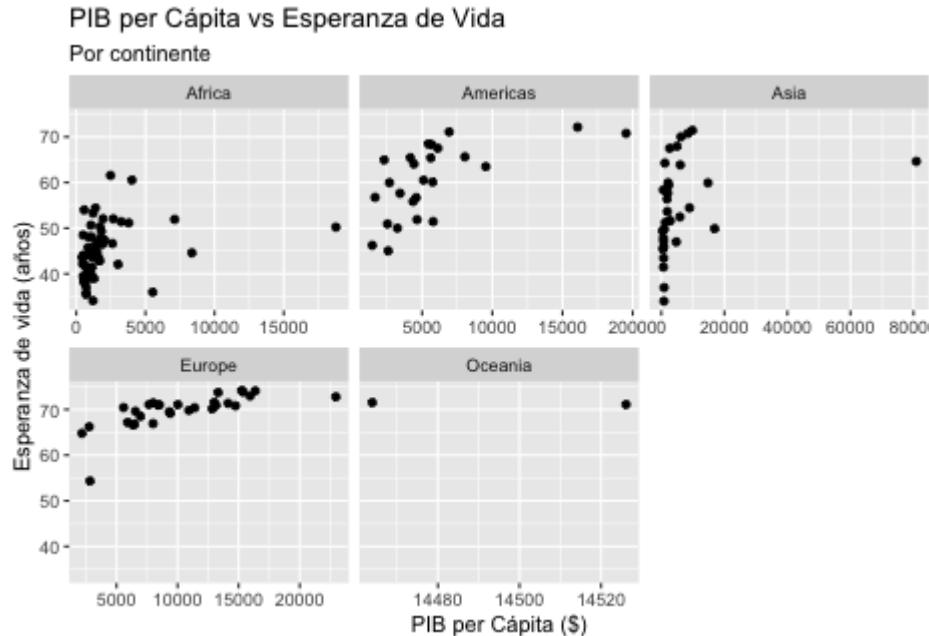
```
ggplot(data = gapminder67, aes(x = gdpPercap, y = lifeExp)) +  
  geom_point() +  
  labs(title = "PIB per Cápita vs Esperanza de Vida",  
       subtitle = "Por continente",  
       x = "PIB per Cápita ($)", y = "Esperanza de vida (años)") +  
  facet_wrap(continent ~ .)
```



Faceting

Múltiples gráficas en función de variable

```
ggplot(data = gapminder67, aes(x = gdpPercap, y = lifeExp)) +  
  geom_point() +  
  labs(title = "PIB per Cápita vs Esperanza de Vida",  
       subtitle = "Por continente",  
       x = "PIB per Cápita ($)", y = "Esperanza de vida (años)") +  
  facet_wrap(continent ~ ., scales = "free_x")
```



Faceting - Resumen

- **facet_grid()**:
 - Red 2d
 - **filas ~ columnas**
 - • para evitar una de ellas
- **facet_wrap()**: 1d
 - fijar escalas con **scales** = ("free_x", "free_y", "free")

Ejercicio 1

Utilizando los datos de la librería **gapminder**:

- Filtra las observaciones correspondientes a los años 1957, 1967, 1977, 1987, 1997, 2007
- Representa la relación entre PIB per cápita y esperanza de vida por continentes y por años en diferentes gráficas usando **faceting**. El tamaño de los puntos ha de ser proporcional a la población. El color debe reflejar el continente.
- ¿Qué conclusiones extraes?

Visualización: distintos tipos de datos

Número de variables a visualizar

- **Análisis de datos univariante:** distribución de una variable

- **Análisis de datos bivariante:** relación entre dos variables

- **Análisis de datos multivariantes:** relación entre varias variables, generalmente enfocándose en dos y condicionando por el resto

Tipos de variables

- **Variables numéricas** pueden ser **contínuas** o **discretas**
 - *altura* es ?
 - *número de cigarros diarios* es ?
 - *distancia* es ?
- **Variables categóricas**, a su vez incluyen **ordinales** si existe un orden natural entre sus niveles o **no ordinales**
 - *color de ojos* es ?
 - *talla de camiseta* es ?

Según tipo de variable tendremos un tipo de representación

Visualización: datos univariantes numéricos

¿Cómo podemos describir datos n^{umericos} univariantes?

- **Forma:**
 - asimetría: simétrico, asimétrico a la derecha, asimétrica a la izquierda
 - número de modas: unimodal, bimodal, multimodal, uniforme
- **Centralidad:** media, mediana, moda
- **Dispersión:** rango, desviación estándar, rango inter-cuartílico
- **Outliers:** existen datos raros

¿Por qué visualizarlos?

```
library(Tmisc)
```

```
##      set  x     y          ##      set  x     y
## 1     I 10 8.04          ## 23    III 10 7.46
## 2     I  8 6.95          ## 24    III  8 6.77
## 3     I 13 7.58          ## 25    III 13 12.74
## 4     I  9 8.81          ## 26    III  9 7.11
## 5     I 11 8.33          ## 27    III 11 7.81
## 6     I 14 9.96          ## 28    III 14 8.84
## 7     I  6 7.24          ## 29    III  6 6.08
## 8     I  4 4.26          ## 30    III  4 5.39
## 9     I 12 10.84         ## 31    III 12 8.15
## 10    I  7 4.82          ## 32    III  7 6.42
## 11    I  5 5.68          ## 33    III  5 5.73
## 12    II 10 9.14         ## 34    IV   8 6.58
## 13    II  8 8.14          ## 35    IV   8 5.76
## 14    II 13 8.74          ## 36    IV   8 7.71
## 15    II  9 8.77          ## 37    IV   8 8.84
```

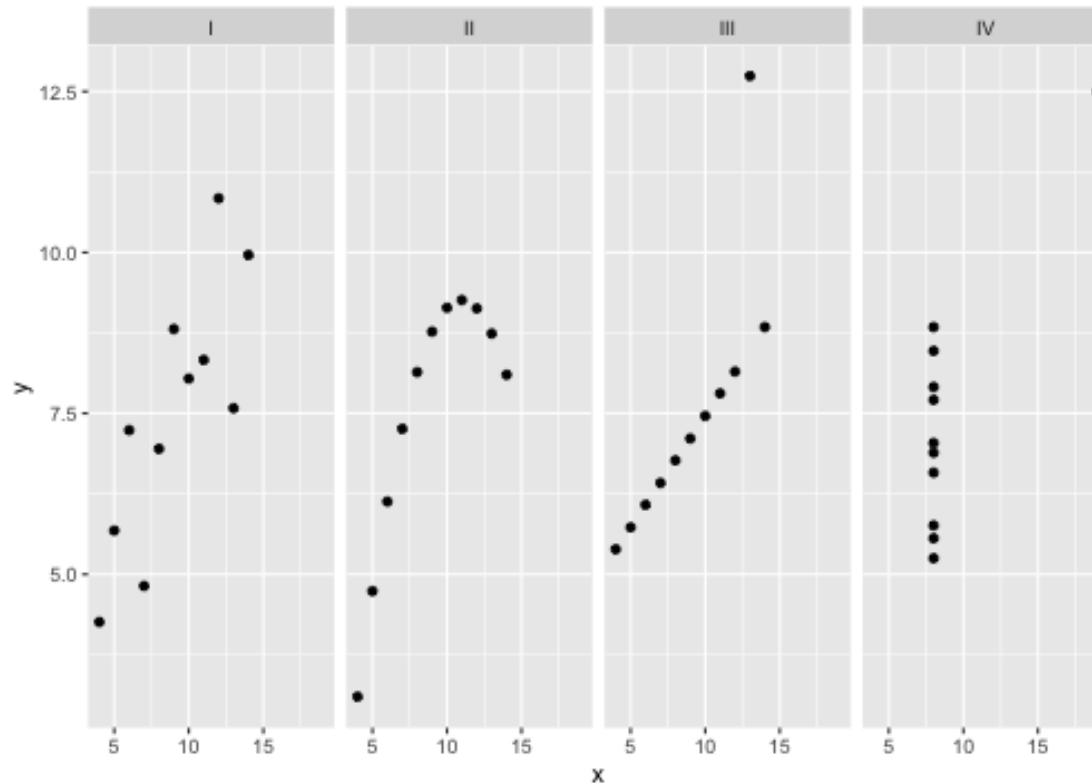
Resumiendo Anscombe's quartet

```
quartet %>%
  group_by(set) %>%
  summarise(
    mean_x = mean(x), mean_y = mean(y),
    sd_x = sd(x), sd_y = sd(y),
    r = cor(x, y)
  )

## # A tibble: 4 × 6
##   set   mean_x  mean_y   sd_x   sd_y      r
##   <fct>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 I       9     7.50    3.32    2.03  0.816
## 2 II      9     7.50    3.32    2.03  0.816
## 3 III     9     7.5     3.32    2.03  0.816
## 4 IV      9     7.50    3.32    2.03  0.817
```

Visualizando Anscombe's quartet

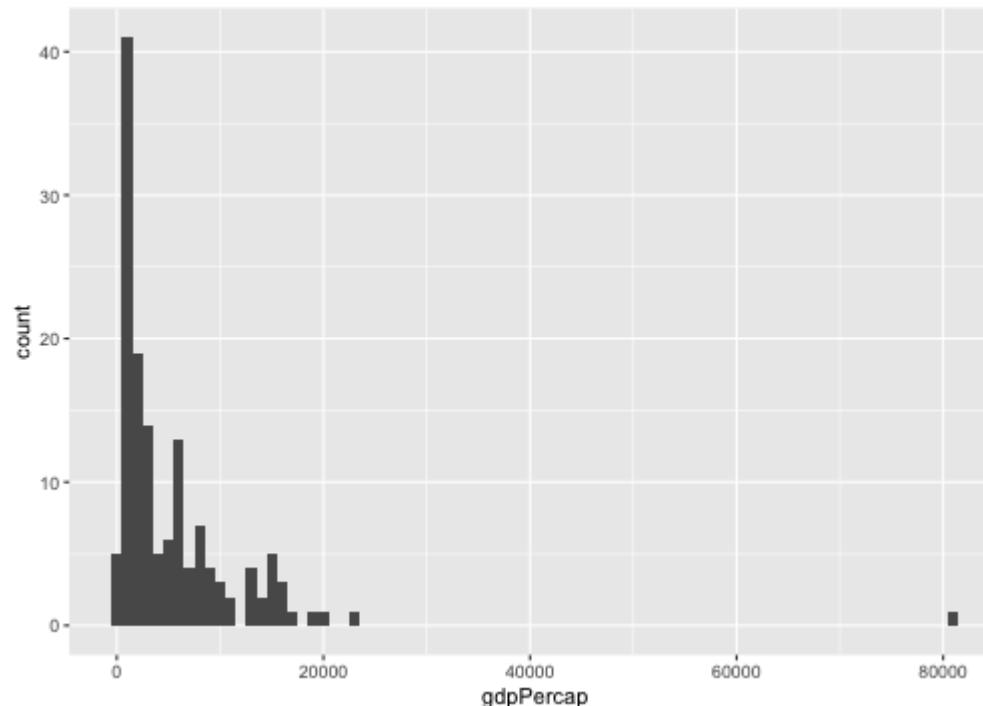
```
ggplot(quartet, aes(x = x, y = y)) +  
  geom_point() +  
  facet_wrap(~ set, ncol = 4)
```



Visualizando datos numéricos univariantes

Histograma - ¿Qué revela este gráfico? (Apuestas...)

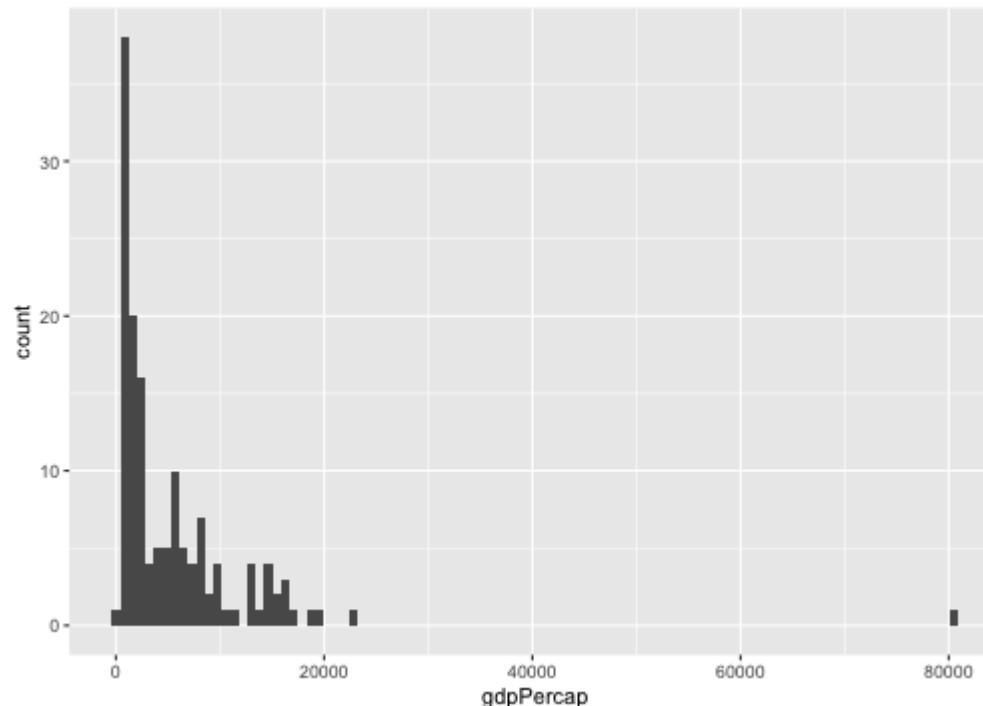
```
ggplot(gapminder67, mapping = aes(x = gdpPercap)) +  
  geom_histogram(binwidth = 1000)
```



Visualizando datos numéricos univariantes

Histograma - ¿Qué revela este gráfico? (Apuestas...)

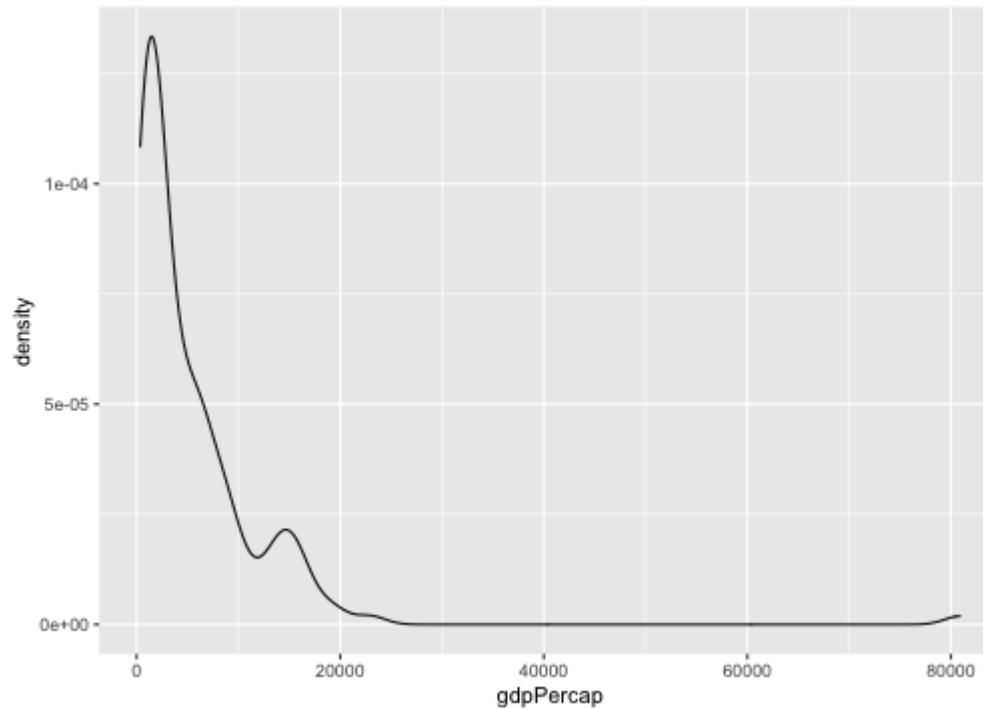
```
ggplot(gapminder67, mapping = aes(x = gdpPercap)) +  
  geom_histogram(bins = 100)
```



Visualizando datos numéricos univariantes

A veces útil

```
ggplot(gapminder67, mapping = aes(x = gdpPercap)) +  
  geom_density()
```

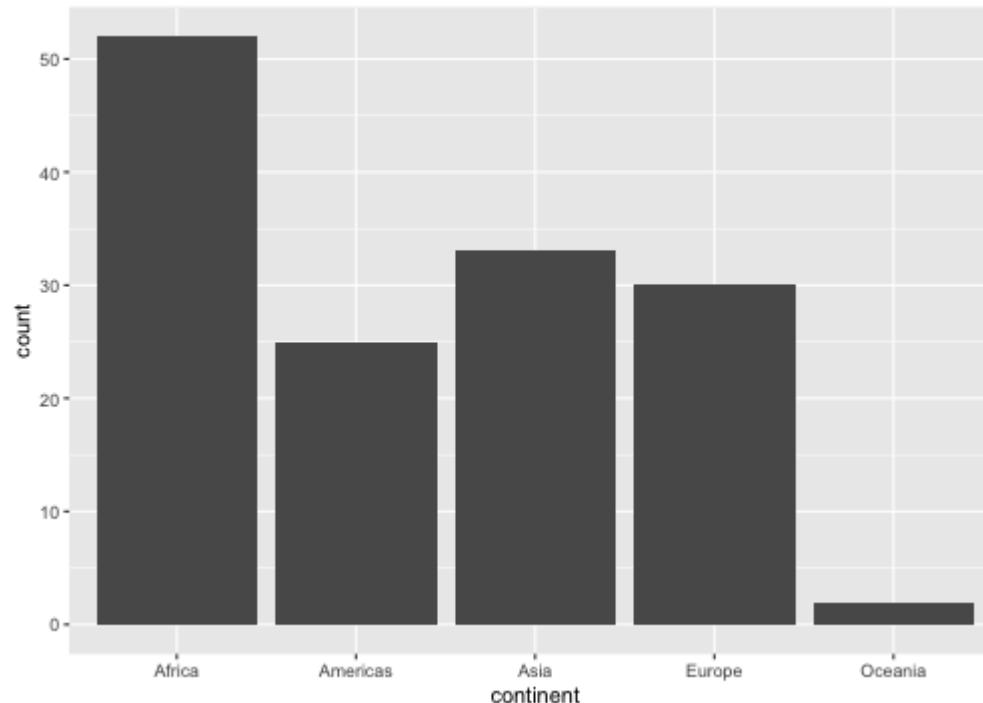


Visualización: datos univariantes categóricos

Visualización de datos univariantes categóricos

Gráfico de barras

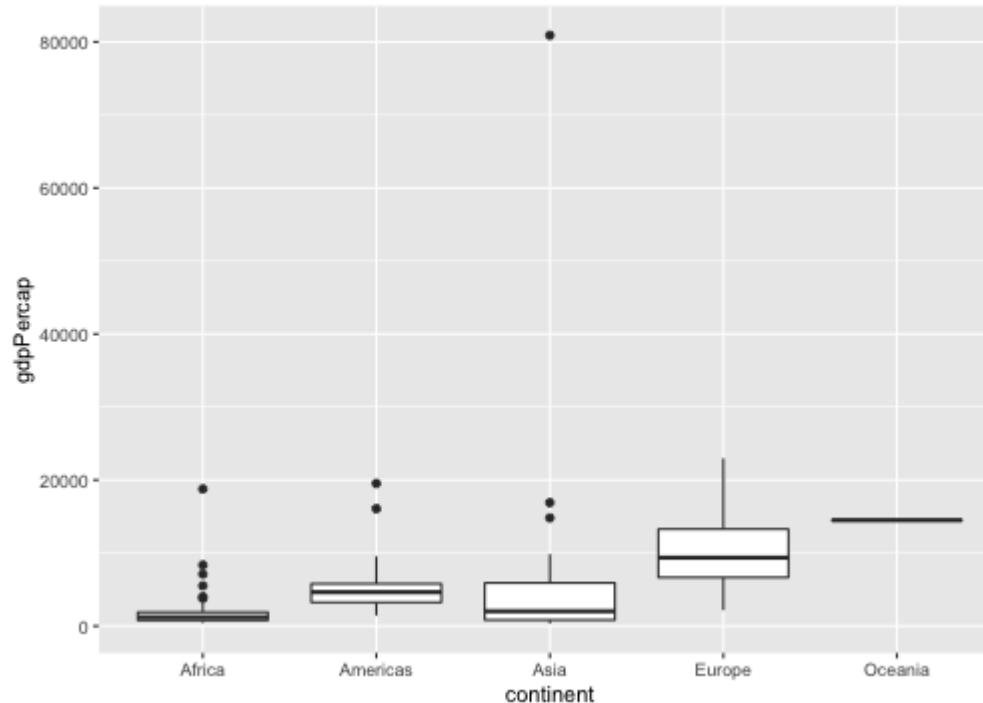
```
ggplot(gapminder67, mapping = aes(x = continent)) +  
  geom_bar()
```



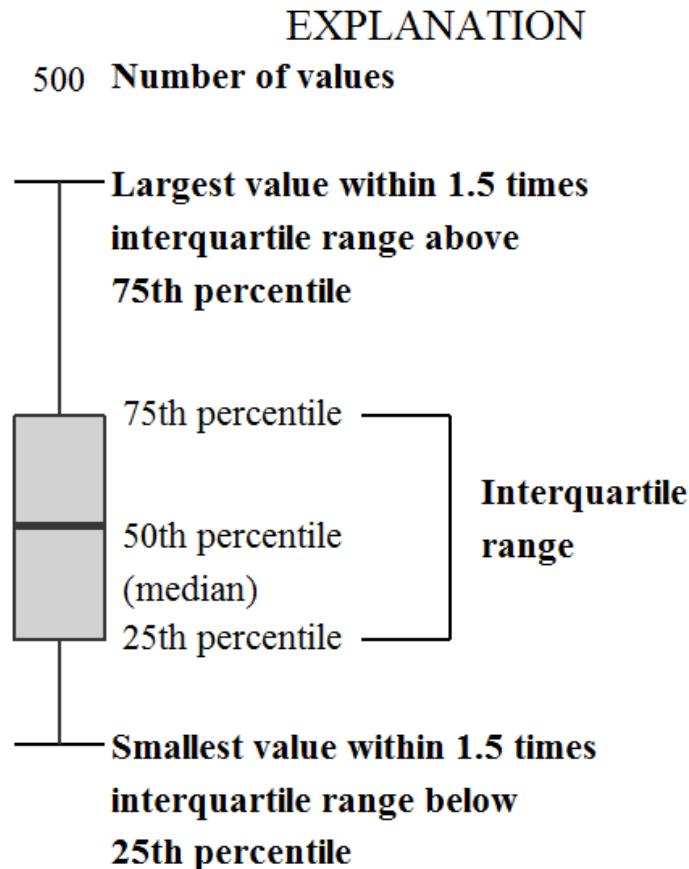
Visualización: datos bivariantes

Visualización: datos bivariantes continuo - categórico

```
ggplot(gapminder67, mapping = aes(x = continent, y = gdpPercap)) +  
  geom_boxplot()
```



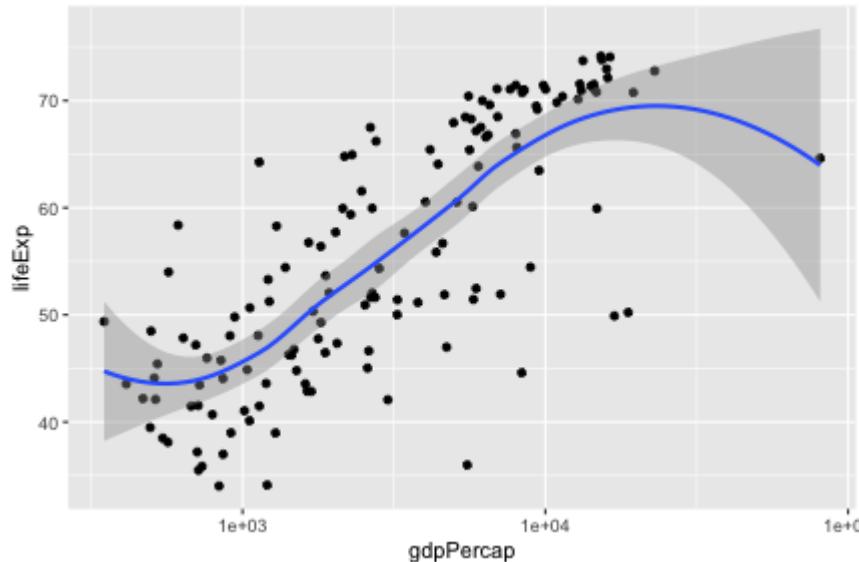
Visualización: datos bivariantes continuo - categórico



Visualización: datos bivariantes continuo - continuo

A veces queda bien añadir un ajuste

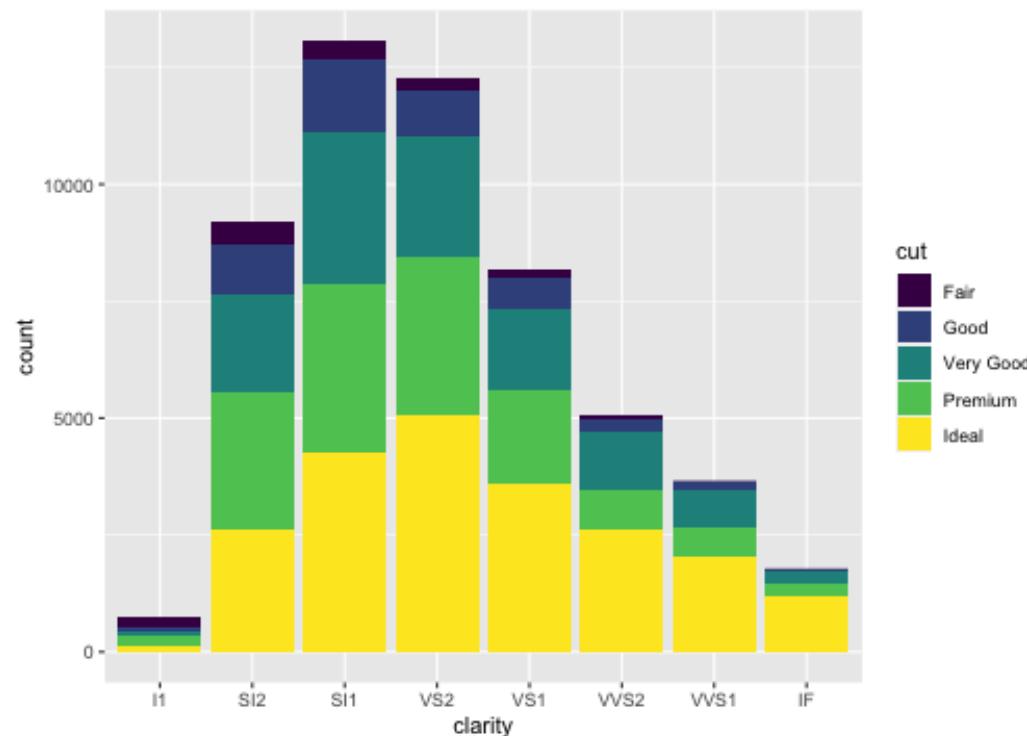
```
ggplot(gapminder67, aes(x = gdpPercap, y = lifeExp) ) +  
  geom_point() +  
  scale_x_log10() +  
  geom_smooth()
```



Visualización: datos bivariantes categorico - categorico

Dataset **diamonds**

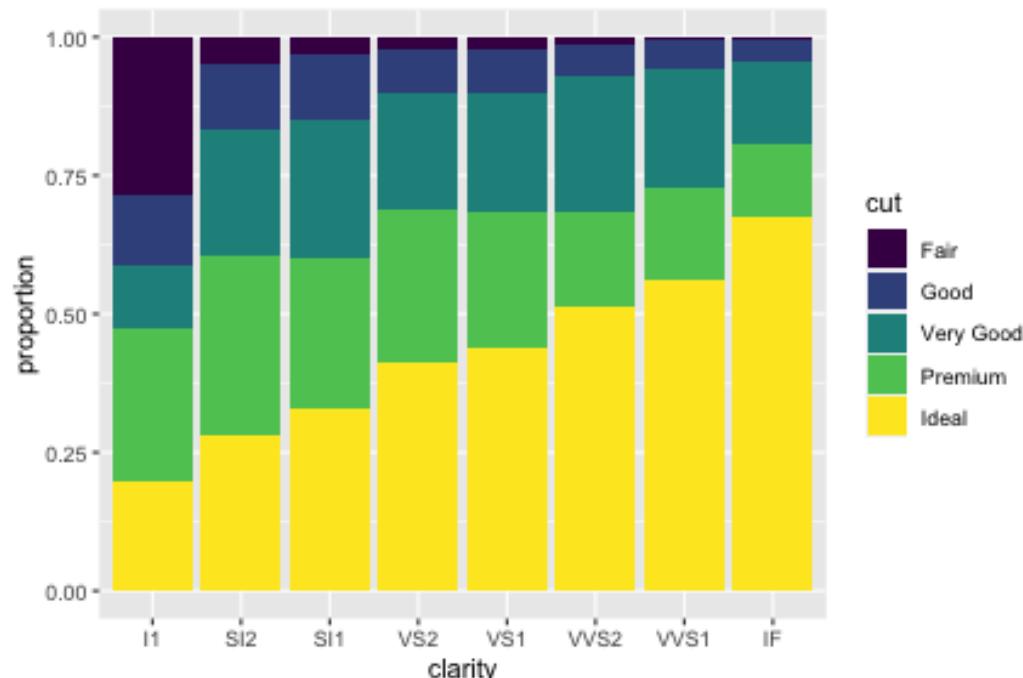
```
ggplot(data = diamonds, mapping = aes(x = clarity, fill = cut)) +  
  geom_bar()
```



Visualización: datos bivariantes categorico - categorico

Dataset **diamonds**

```
ggplot(data = diamonds, mapping = aes(x = clarity, fill = cut)) +  
  geom_bar(position = "fill") +  
  labs(y = "proportion")
```



Visualización: transformaciones estadísticas

Transformaciones estadísticas

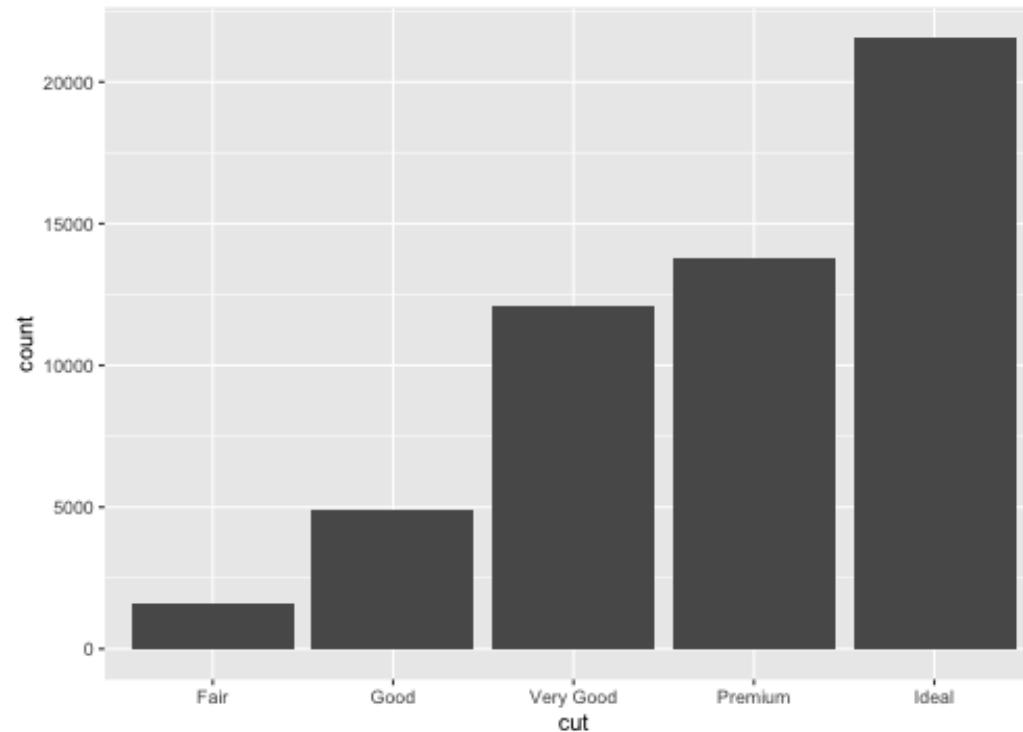
Los gráficos como histogramas, gráficos de barras o boxplots realizan transformaciones a los datos antes de representarlos

- Gráficos de barras e histogramas: parcelan los datos y representan un número por cada parcela
- Smoothers: ajustan un modelo y representan predicciones generadas por el mismo
- Boxplots: calculan un resumen de la distribución y lo pintan en determinado formato

El argumento para especificar el estadístico de resumen es **stat**. Puedes ver su valor por defecto en, e.g. **?geom_bar**

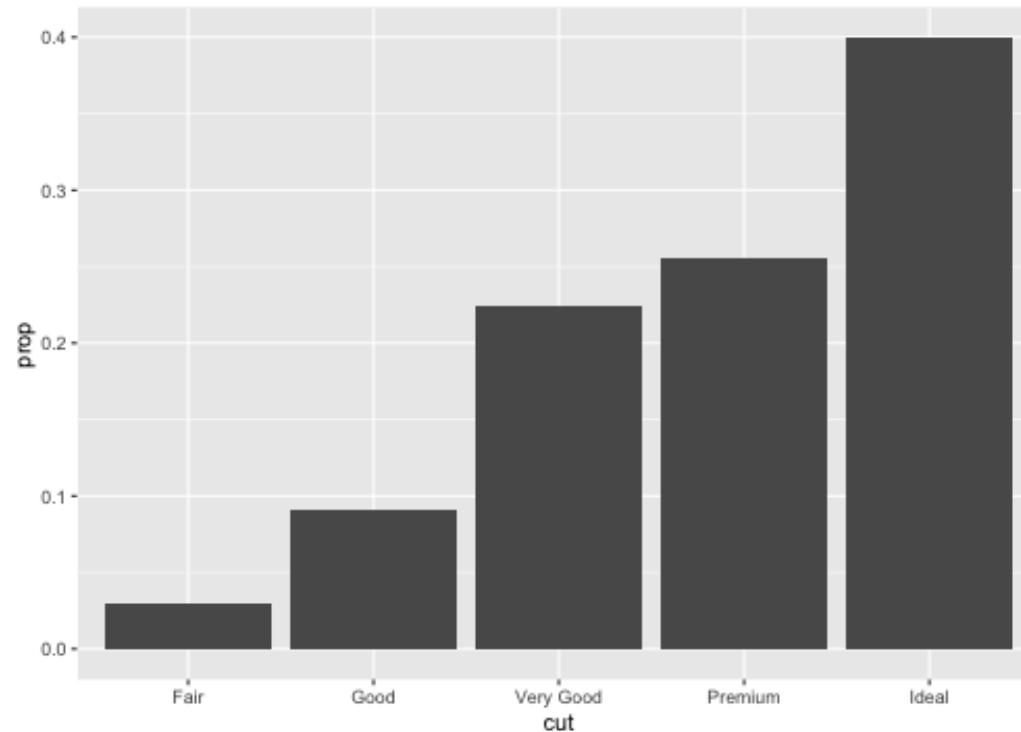
Transformaciones estadísticas

```
ggplot(data = diamonds) +  
  geom_bar(mapping = aes(x = cut))
```



Transformaciones estadísticas

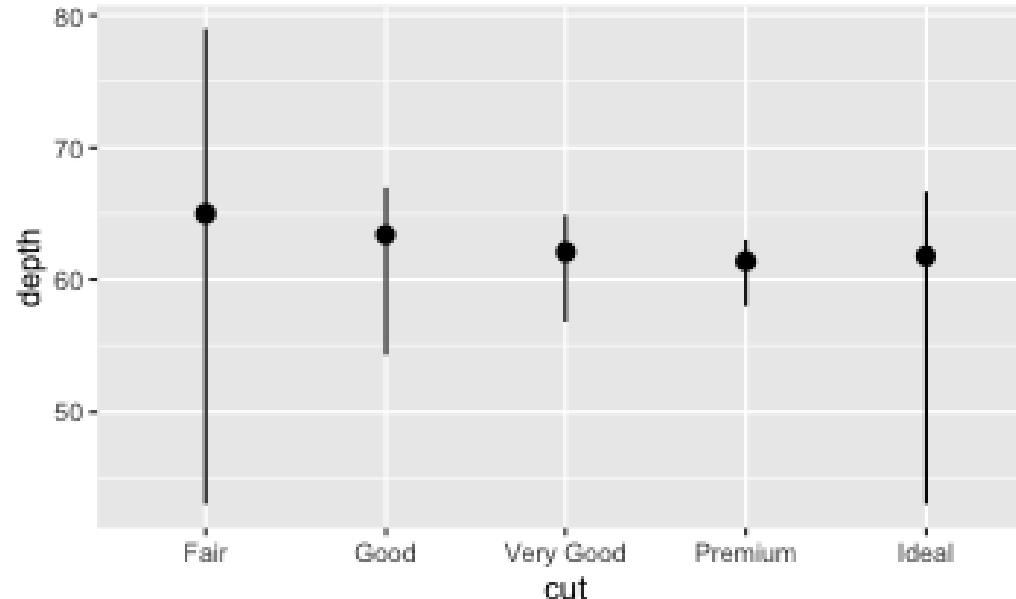
```
ggplot(data = diamonds) +  
  geom_bar(mapping = aes(x = cut, y = stat(prop), group = 1) )
```



Transformaciones estadísticas

Podemos crear estadísticos propios

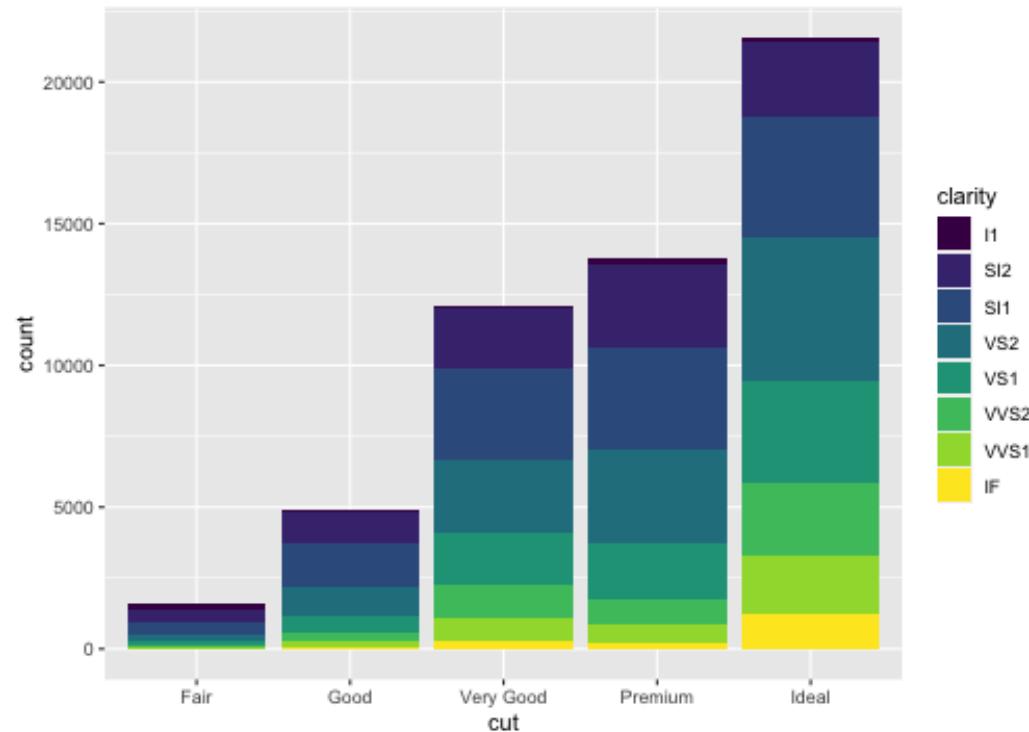
```
ggplot(data = diamonds) +  
  stat_summary(  
    mapping = aes(x = cut, y = depth),  
    fun.min = min,  
    fun.max = max,  
    fun = median  
)
```



Visualización: ajuste de posición

Ajustes de posición

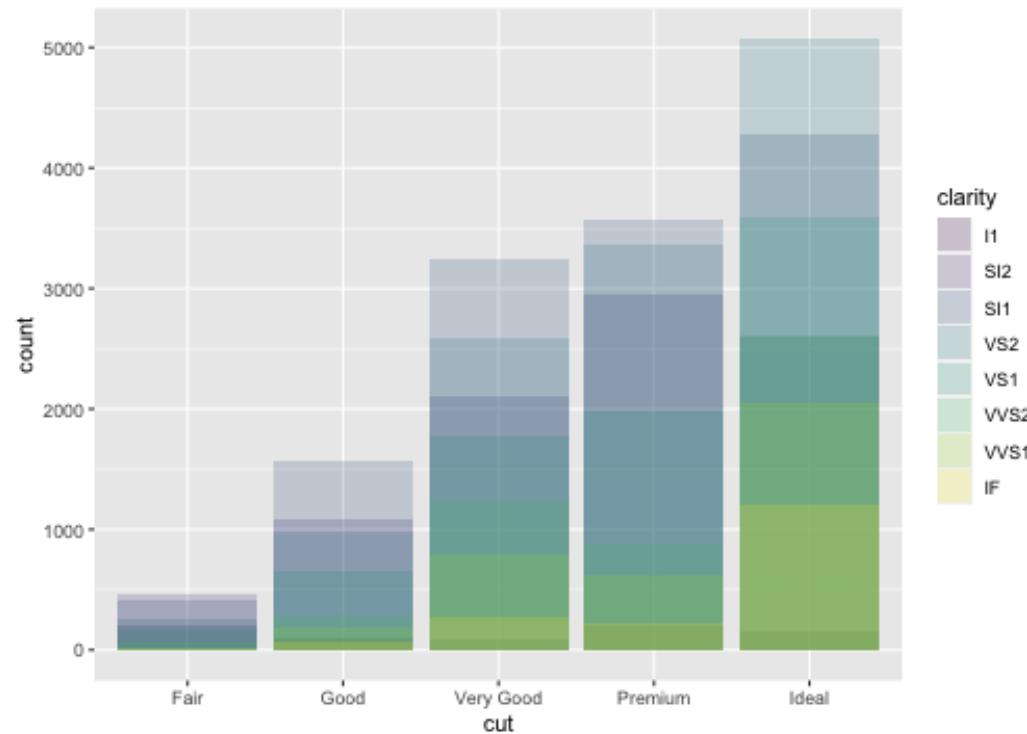
```
ggplot(data = diamonds) +  
  geom_bar(mapping = aes(x = cut, fill = clarity))
```



El apilamiento se realiza automáticamente con el ajuste de posición especificado a través de la variable **position**

Ajustes de posición - "identity"

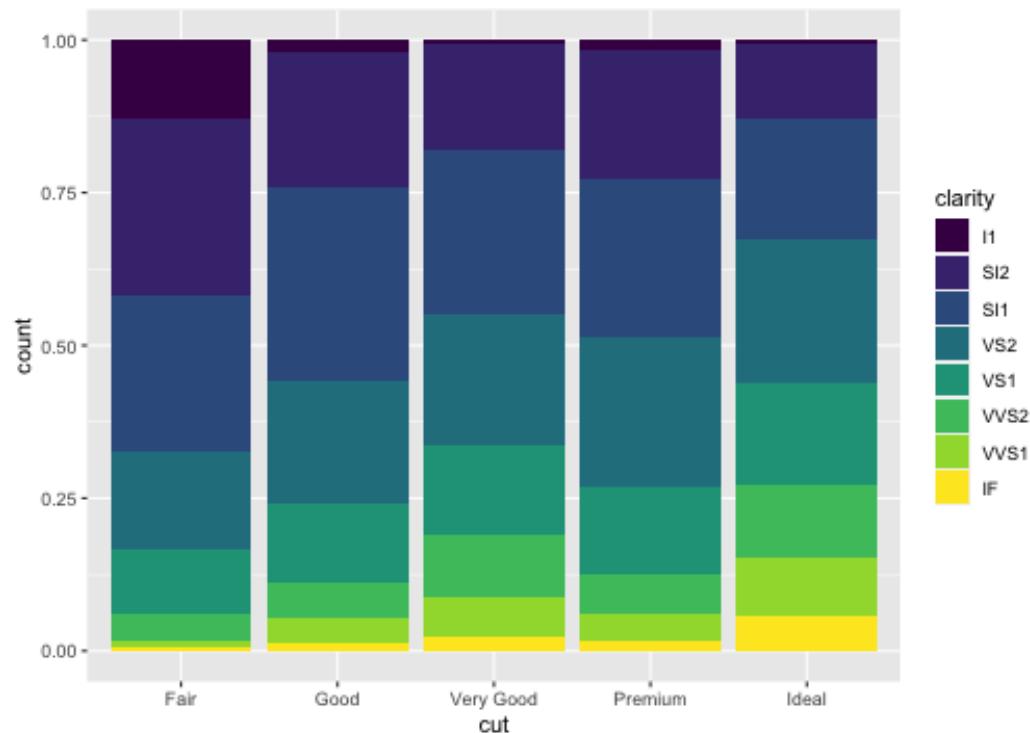
```
ggplot(data = diamonds, mapping = aes(x = cut, fill = clarity)) +  
  geom_bar(alpha = 1/5, position = "identity")
```



Ajustes de posición - "fill"

¿Para qué es bueno este tipo de gráfico?

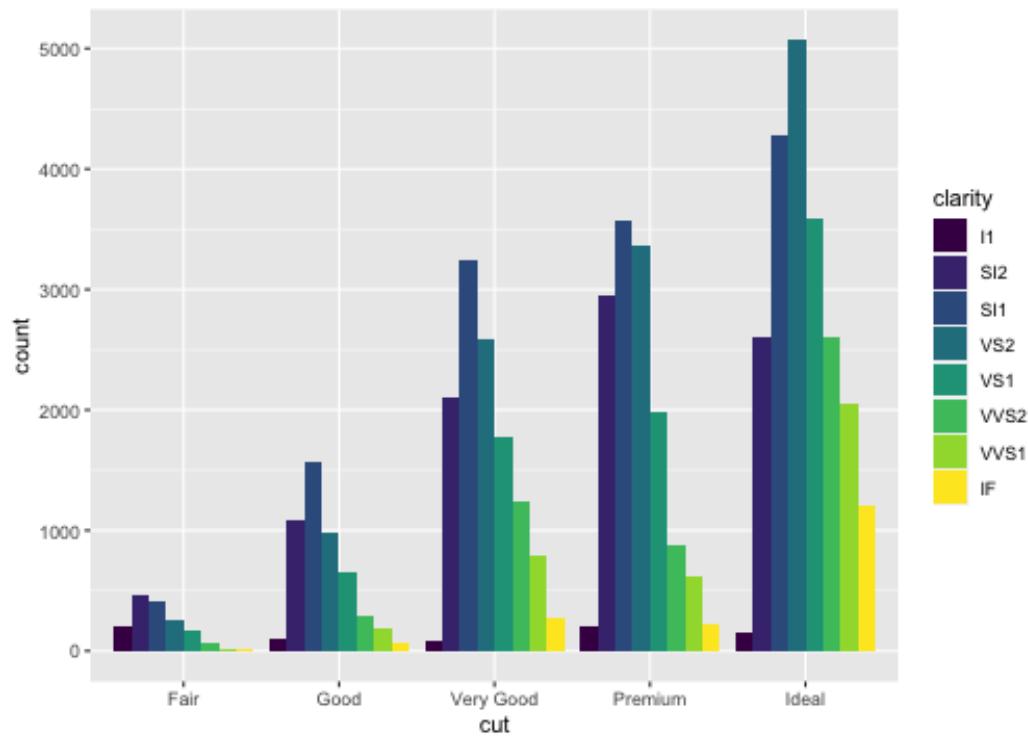
```
ggplot(data = diamonds, mapping = aes(x = cut, fill = clarity)) +  
  geom_bar(position = "fill")
```



Ajustes de posición - "dodge"

¿Y este?

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

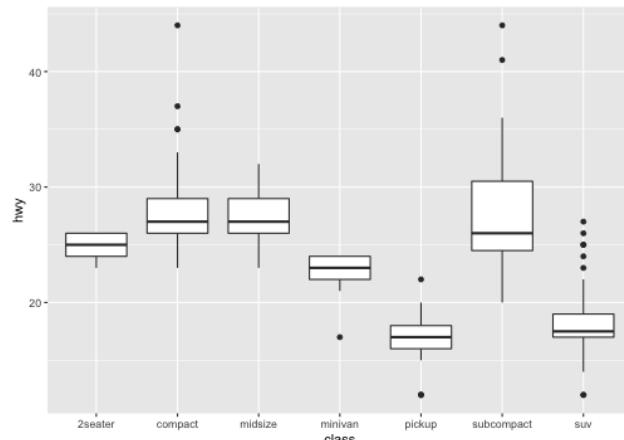


Visualización: coordenadas

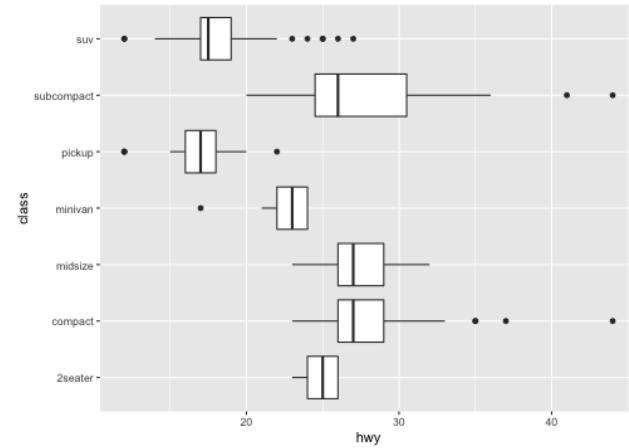
Coordenadas

Por defecto **ggplot2** usa coordenadas Cartesianas. Las coordenadas pueden modificarse con **coord_xxx()**

```
ggplot(data = mpg, mapping = aes(x =  
    geom_boxplot())
```



```
ggplot(data = mpg, mapping = aes(x =  
    geom_boxplot() +  
    coord_flip())
```

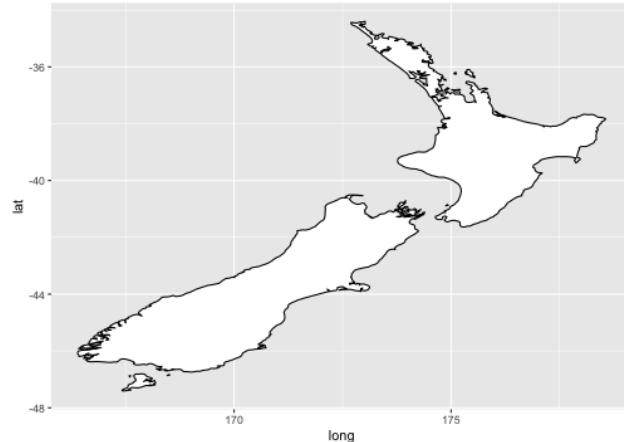


Coordenadas

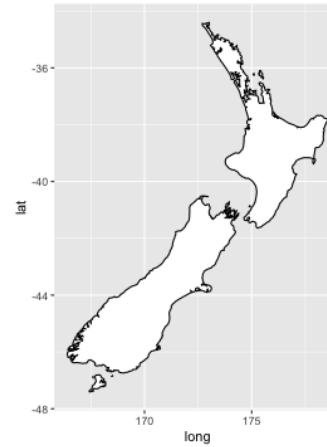
`coord_quickmap()` calcula el ratio correcto para mapas

```
library(maps)
nz <- map_data("nz")

ggplot(nz, aes(long, lat, group = gr
geom_polygon(fill = "white", colou
```



```
ggplot(nz, aes(long, lat, group = gr
geom_polygon(fill = "white", colou
coord_quickmap()
```

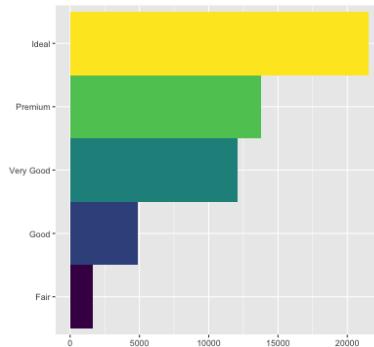


Coordenadas

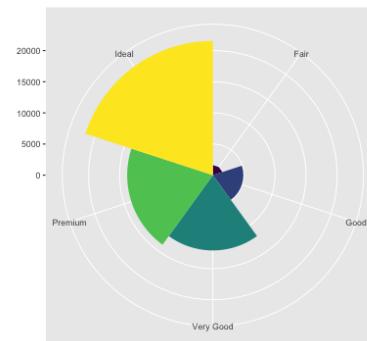
coord_polar()

```
bar <- ggplot(data = diamonds) +  
  geom_bar(  
    mapping = aes(x = cut, fill = cu  
    show.legend = FALSE,  
    width = 1  
  ) +  
  theme(aspect.ratio = 1) +  
  labs(x = NULL, y = NULL)
```

```
bar + coord_flip()
```



```
bar + coord_polar()
```



Resumen

```
ggplot(data = <DATA>) +  
  <GEOM_FUNCTION>(  
    mapping = aes(<MAPPINGS>),  
    stat = <STAT>,  
    position = <POSITION>  
  ) +  
  <COORDINATE_FUNCTION> +  
  <FACET_FUNCTION>
```

Visualización: consejos

Consejos

<https://blog.csgsolutions.com/6-tips-for-creating-effective-data-visualizations>

Bibliografía

- [R for Data Science](#), Wickham and Grolemund (2016)
- [ggplot2.tidyverse.org](#)
- [Data Visualization, A practical introduction](#), Healy (2018)
- [ggplot2.tidyverse.org](#)
- [ggplot2 cheat sheet](#)
- [Top 50 ggplot2 visualizations](#)
- [How the BBC uses ggplot2](#)
- [ggplot2: Elegant Graphics for Data Analysis](#)