Contents

1 Introducción.

Este es un pequeño manual que escribí cuando aprendí a hacer mapas de México en R. En los últimos años he invertido mucho tiempo en aprender a usar varios paquetes de R. Siempre encuentro muy frustrante regresar a lo que algunos meses antes dominaba y encontrar que no recuerdo bien todo lo que hice. Para poder retomar los temas con más facilidad, decidí decidí elaborar algunas notas personales cada vez que aprendo un paquete nuevo, con muchos ejemplos. En mis notas uso pequeñas bases de datos, siempre inventados, para poder concentrarme en lo que hace cada función de R en vez de los datos en sí. Dado que me han sido de utilidad, decidí pulir un poco estas notas y hacerlas una especie de manuales informales, esperando que puedan serle de utilidad a alguien.

Con los mapas me sucedió además que todos los ejemplos eran con mapas de otros lugares. Como esto lo aprendí por mi cuenta, me costó un poco de trabajo navegar la página de INEGI para encontrar los mapas que quería usar y saber exactamente cuales eran los archivos que necesitaba bajar y cargar. Este manual no pretende ser nada muy formal, pero puede ser de utlildad para alguien que no sabe por donde empezar.

Los mapas elaborados en este tutorial son mapas vectoriales. Es importante entender que la información se usa para los mapas no son imágenes en sí, sino bases de datos que contienen coordenadas de puntos en las fronteras de cada estado. Lo que hace R es tomar estos puntos, graficarlos y unirlos con segmentos rectos para dibujar la silueta de cada estado. Esto se explica en varias secciones, incluyendo una introducción a un tipo especial de objeto que tiene un conjunto de coordenadas como elementos.

El objetivo final es poder graficar un mapa de toda la república, o una parte de esta en la cual cada estado esté "pintado" de un color que indique el valor de una variable, ya sea categegórica o numérica.(Lo que en inglés se llama "choropleth maps"). Por ejemplo, se puede tener un mapa en el que cada color indique el partido que gobierna al estado, o que cada color indique un rango de valores en el que se encuentra la población total de cada estado.

En estas notas se incluyen instrucciones de cómo bajar los datos geográficos de México, dónde encontrarlos y cómo añadirle datos para graficar. Está dirigido a personas que saben lo básico de *R*. Es recomendable, más no indispensable saber un poco de *dplyr*. Todo el código se incluye en archivos de *R*.

A los datos geográficos les añadimos datos generados aleatoriamente que no tienen ningún significado, para tener algo que graficar en el mapa. Algunos de los mapas no solamente no son estéticamente agradables, sino que algunos son muy feos, pero su objetivo es resaltar diferentes opciones gráficas, an algunos casos exagerándolas para que resulten obvias.

Se anexan tres archivos de código R.

- mexico.R Código para bajar datos y crear objeto en R que servirá como base del mapa.

- shapes_intro.R. Introducción básica al paquete sf (simple features).
- mapamx.R Código para elaboración de mapas de México. Incluye instrucciones de cómo simplificar los mapas, añadir datos al archivo de mapas y varias formas de graficar esos datos en un mapa.

También se anexa un archivo en formato RData llamado mapasmex_simple.RData que incluye el objeto *sf* con la información de los mapas, ya en formato R. Este archivo se pueden cargar directamente a R, en caso de no poder o no querer bajar la información del sitio de internet de Inegi. Hay otro archivo similar, llamado mapasmex_data.RData que tiene el objecto *sf* con los datos extra generados artificialmente. Estos se pueden usar si se quiere ir directamente a las secciones en las que se grafican los mapas. No se recomienda cargar ambos archivos a R, porque los objetos con y sin datos adicionales se llaman igual.

Para correr estos programas,se necesitan las siguientes librerías:

- dplyr
- ggplot
- tmap
- sf
- rmapshaper
- RColorBrewer

2 Datos espaciales, "simple features" y la librería *sf*.

Los "simple features" es un estándar para manejar objetos geométricos utilizado para sistemas de información geográfica. Ejemplos de objetos básicos geométricos son los puntos, líneas y polígonos. En R, estos objetos se pueden leer crear y manejar en R con la librería *sf*. La clase de estos objetos *sfg*. Se pueden generar a partir de objetos base de R. Un punto se crea con un vector, una línea con una matriz y un polígono con una lista, cuyo primer elemento es una matriz con los vértices, siempre "cerrando" el polígono, es decir, el primer renglón y el último deben ser iguales.

```
# point (punto)
punto <- st_point(c(5, 2))
# línea
linestring_matrix = rbind(c(1, 5), c(4, 4), c(4, 1), c(2, 2), c(3, 2))
linea <-st_linestring(linestring_matrix)
# polígono
polygon_list = list(rbind(c(1, 5), c(2, 2), c(4, 1), c(4, 4), c(1, 5)))
poligono <- st_polygon(polygon_list)

punto
```

```
## POINT (5 2)
```

```
linea
```

```
## LINESTRING (1 5, 4 4, 4 1, 2 2, 3 2)
```

```
poligono
```

```
## POLYGON ((1 5, 2 2, 4 1, 4 4, 1 5))
```

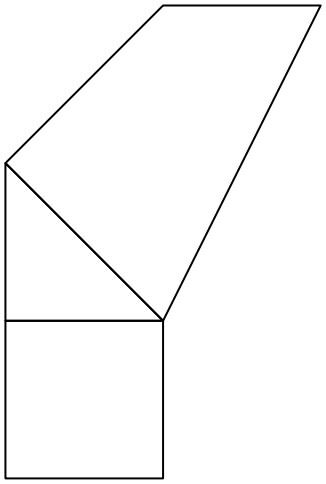
La información para generar los mapas es un objeto de clase *sf* (simple features), que es similar a un data frame, en el cual una de las columnas está conformada por objetos *sfg*, que en el caso de los mapas determinan la silueta de cada estado. La columna de objetos *sfg* es un objeto de clase *sfc*, que no es más que una colección de objetos *sfg*. De hecho, un objeto *sf* está formalmente conformado por una columna *sfc* Y un data frame.

Para entenderlo mejor, veamos un ejemplo sencillo. Primero creamos tres polígonos, cada uno es un objeto *sfg*. Los polígonos se crean a partif de una lista de vectores cuyas coordenadas corresponden a los vértices de los polígonos:

```
#Poligono 1
polygon_list1 = list(rbind(c(0,3), c(0,4), c(1,3), c(0,3)))
polygon1 = st_polygon(polygon_list1)
# Poligono 2
polygon_list2 = list(rbind(c(0, 2), c(1, 2), c(1, 3), c(0, 3), c(0, 2)))
polygon2 = st_polygon(polygon_list2)
# Poligono 3
polygon_list3 = list(rbind(c(1,3), c(0,4), c(1,5), c(2,5), c(1,3)))
polygon3 = st_polygon(polygon_list3)
```

Estos polígonos se pueden agrupar para crear un objeto *sfc* (simple features column). A continuación se muestra una gráfica del objeto sfc, que muestra los tres polígonos.

```
#Columna con dos poligonos (objeto sfc)
#crear
polygon_sfc = st_sfc(polygon1, polygon2, polygon3)

plot(polygon_sfc)
```



Para obtener un objeto *sf*, se necesita añadirle un data frame a la columna *sfc*. En este ejemplo, primero creamos un data frame con datos inventados. El data frame se llama *poly_attrib* y tiene dos columnas, una categórica y una numérica. Para juntar los dos objetos, se usa la función *st-sf*. A continuación se muestra cómo se genera el data frame, cómo se junta con el polígono.

```
# Añadirle otros datos y convertirlo en un objeto sf

#Crear "datos"
poly_attrib <- data.frame(color= c("blue", "green", "red"), valor= c(1, 2, 2))

#juntar el objeto sfc con el data frame para obtener un sf
poly_sf <- st_sf(polygon_sfc, poly_attrib)
```

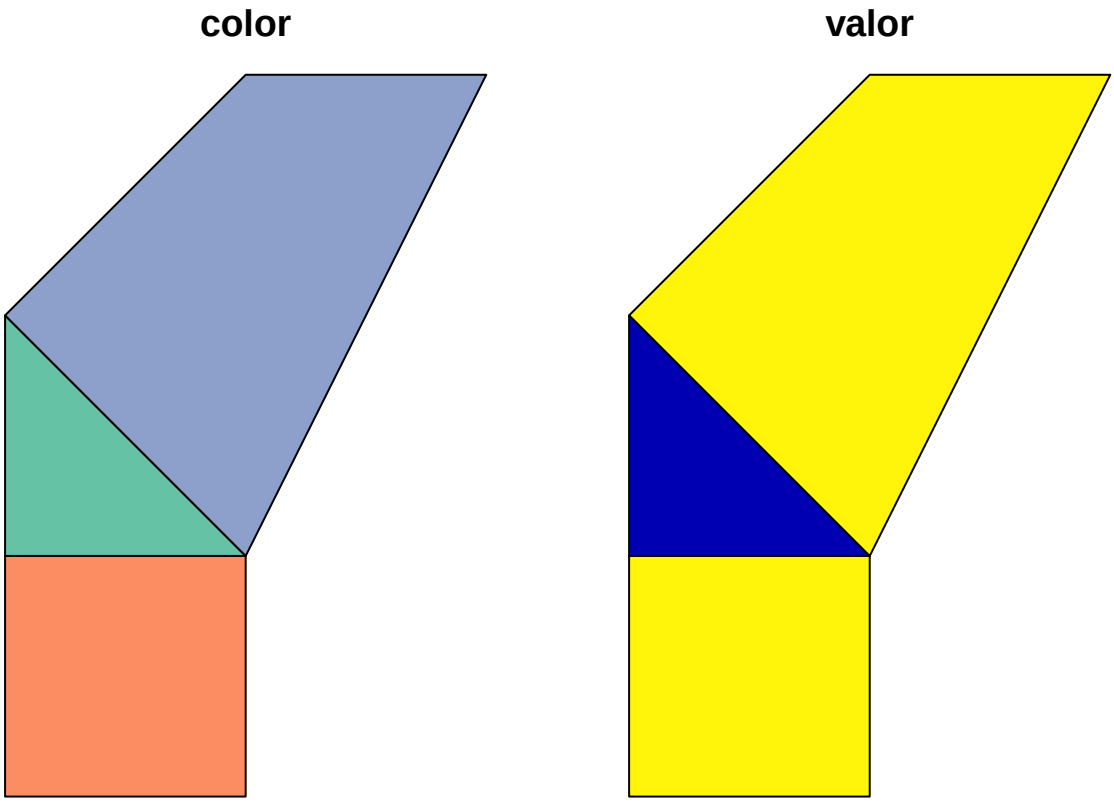
Finalmente mostramos el listado del objeto, con las dos columnas de datos y la columna de objetos *sfg* y una gráfica de los polígonos.

```
poly_sf
```

```
## Simple feature collection with 3 features and 2 fields
## Geometry type: POLYGON
## Dimension:     XY
```

```
## Bounding box:  xmin: 0 ymin: 2 xmax: 2 ymax: 5
## CRS:           NA
##   color valor                    polygon_sfc
## 1  blue     1 POLYGON ((0 3, 0 4, 1 3, 0 3))
## 2 green     2 POLYGON ((0 2, 1 2, 1 3, 0 ...
## 3   red     2 POLYGON ((1 3, 0 4, 1 5, 2 ...
```

```
plot(poly_sf)
```



Al objeto *sf* se le pueden añadir otras columnas de datos de la misma forma en la que se le añadirían a cualquier *data frame*, ya sea con R base o *dplyr*.

```
## al objeto poly_sf se le pueden añadir otros datos.

sabor = c("vainilla", "chocolate", "fresa")

# con dplyr
poly_sf %>% mutate(sabor = sabor)
```

```
## Simple feature collection with 3 features and 3 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: 0 ymin: 2 xmax: 2 ymax: 5
## CRS:           NA
##   color valor                    polygon_sfc     sabor
## 1  blue     1 POLYGON ((0 3, 0 4, 1 3, 0 3))  vainilla
## 2 green     2 POLYGON ((0 2, 1 2, 1 3, 0 ... chocolate
## 3   red     2 POLYGON ((1 3, 0 4, 1 5, 2 ...     fresa
```

```r
# o con rbase

poly_sf$sabor = sabor

cbind(poly_sf, sabor)
```

```
## Simple feature collection with 3 features and 4 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: 0 ymin: 2 xmax: 2 ymax: 5
## CRS:           NA
##   color valor     sabor   sabor.1                    polygon_sfc
## 1  blue     1  vainilla  vainilla POLYGON ((0 3, 0 4, 1 3, 0 3))
## 2 green     2 chocolate chocolate POLYGON ((0 2, 1 2, 1 3, 0 ...
## 3   red     2     fresa     fresa POLYGON ((1 3, 0 4, 1 5, 2 ...
```

Los datos que se utilizarán para elaborar los mapas son objetos *sf* similares a los que se usaron para dibujar los polígonos en esta sección (pero con muchísimos vértices), en los cuales cada renglón correponde a un estado de México. La columna *geometry* es un objeto *sfc*, conformado por "polígonos" georreferenciados. Esto último significa básicamente que estos datos tienen como referencia un origen en común, para que cada estado se grafique en el lugar correcto con respecto al resto de los estados.

Para aprender más sobre este tema, ver liga

3 Bajar los mapas.

Como se explicó anteriormente, los mapas que bajaremos de INEGI no son imágenes, sino una base de datos con coordenadas que *R* utilizará para graficar los mapas, graficando las coordenadas e interpolando líneas rectas entre ellas.

Los mapas usados en este ejemplo son archivos en formato *shp* (shapefile), tomados de la página de INEGI inegi.org.mx En este tutorial se utiliza el mapa más sencillo, que incluye unicamente una columna indicando el nombre de cada entidad y otra columna con los datos geográficos de la entidad. El mapa se encuentra en la página

El archivo *mapamx.R* tiene las instrucciones completas para bajar al mapa y borrar el zip y el directorio. Los archivos que se necesitan (a nivel estado) son *00ent.shp*, *00ent.shx*, *00ent.dbf*, *00ent.prj* y *00ent.shx*. Si se desea, se puede evitar borrar el directorio que se bajó de la página de INEGI, ya que tiene información extra sobre los mapas y otros archivos de mapas (por ejemplo, a nivel municipal) que pueden resultar interesantes. Para evitar borrar el directorio, es necesario "comentar" (poner un signo # antes de la instrucción *unlink(temp)*). De forma alternativa, se puede bajar directamente el archivo comprimido de la liga, abrirlo y simplemente leer el archivo *00ent.shp* que buscamos usando la instrucción *st_read*. Tomar en cuenta que tiene que estar el "path" al archivo completo o cambiar el directorio al cual se encuentra el archivo.

```r
mapamex <- st_read("00ent.shp")
```

Si corrió correctamente, se debe haber cargado un objeto llamado *mapamex* en el ambiente *R*. Este objeto es de clase *sf*. Este objeto se encuentra en el repositorio en formato RData. El nombre del archivo es *mapamex.RData*. Se puede cargar directamente a R con la instrucción load:

```r
load(mapamex.RData)
```

4 Graficar los mapas.

En esta sección se explica a detalle cómo simplificar los mapas, añadirles datos y graficarlos utilizando el paquete *tmap*. Todo el código documentado se encuentra en el archivo *mapamex.R*

4.1 Simplificar el mapa

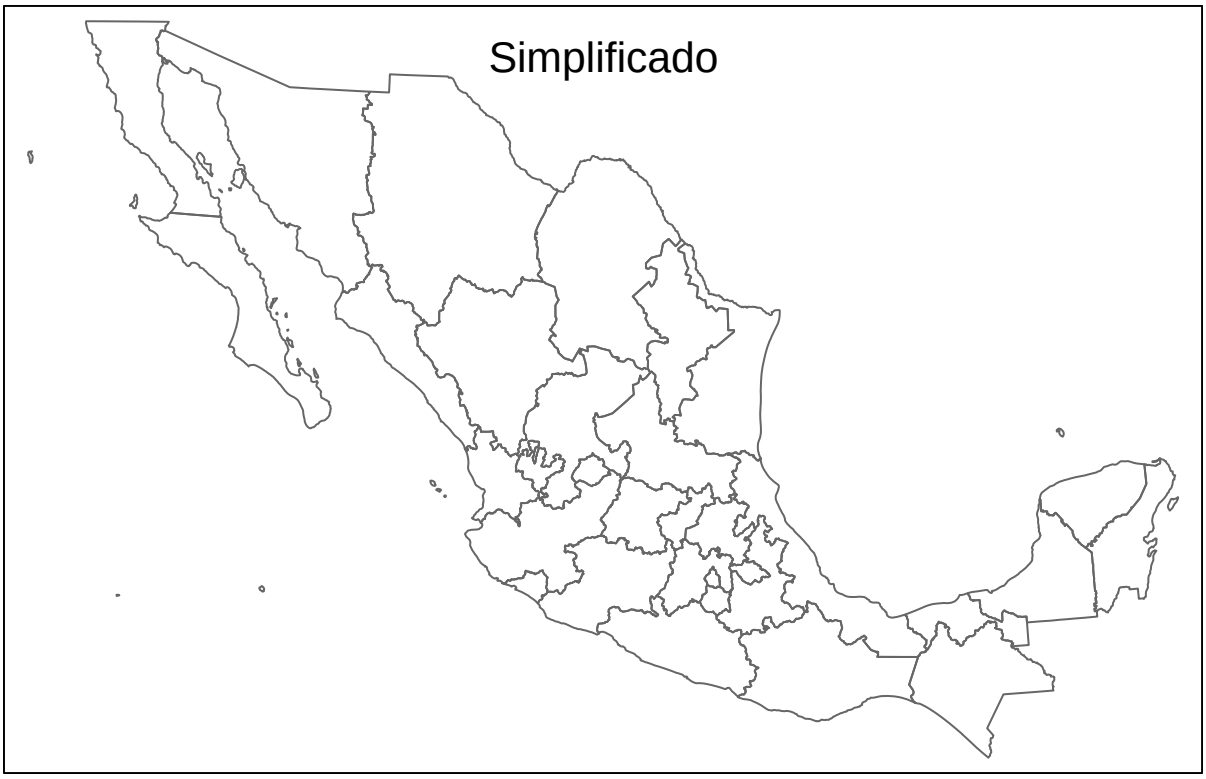
Como se explicó anteriormente, cada estado se grafica a partir de coordenadas que forman vértices unidos por líneas rectas. Cada estado tiene una enorme cantidad de vértices para poder dibujarlo con mucho detalle o en formato muy grande. Para muchas aplicadiones, algo tan detallado no es necesario. Los mapas se pueden simplificar facilmente, quitando algunos de los vértices. Simplificarlos tiene las ventajas de que los archivos se vuelven más pequeños y se grafican mucho más rápido. La librería *sf*, que es una de los que utilizaremos para graficar, hace esta simplificación con la función *st_simplify*. El parámetro dTolerance determina que tanto se simplifica y mientras mayor sea, más vértices quita y más se simplifica.

```
#Simplificar mapa
mex_simple <- st_simplify(mapamex, dTolerance=2000,preserveTopology = FALSE )

mex_simple %>%tm_shape() +
    tm_borders()
```

El problema con *st_simplify()* es que simplifica los elementos de la columna de coordenadas individualmente, lo cual puede hacer que las fronteras ya no coincidan. La función *ms_simplify()* del paquete *rmapshaper* resuelve este problema, ya que simplifica toda la columna y fuerza a que las fronteras entre los estados coincidan. Con esta función se especifica la proporción de vértices que se quieren mantener, a través del parámetro *keep*. El valor de este parámetro dependerá de la escala a la que se quiera graficar. En el ejemplos siguientes vemos que aún manteniendo menos del 1% de los vértices es perfectamente posible distinguir la forma del país y de los estados. En algunos casos, un mapa muy simplificado, casi "caricaturizado" puede ser lo que estemos buscando.

```
#mapa simplificado
mapamex_simple <-  ms_simplify(mapamex, keep = 0.01, keep_shapes = TRUE)

#mapa muy simplificado
mapamex_muysimple <- ms_simplify(mapamex, keep = 0.002, keep_shapes = TRUE)
```

```
mapamex_simple %>%tm_shape() +
                  tm_borders() +
                  tm_layout(title = "Simplificado",
                            title.position = c('center', 'top'))
```




```
mapamex_muysimple %>% tm_shape() +
    tm_borders() +
    tm_layout(title = "Muy simplificado",
              title.position = c('center', 'top'))
```

4.2 Añadir datos al objeto *sf*.

En esta sección se utiliará la versión simplificada generada en la sección anterior, el objeto *mapamex_simple*. Este objeto también se puede encontrar en el repositorio con el nombre de mapamex_simple.RData. Para añadirle datos para graficar al objeto *sf*, basta con tratarlo como si fuera un *data.frame*. Como con cualquier *data.frame*, existen varias formas de añadirle una columna.

Para mostrar como se hace, primero generamos unos datos falsos. Para la variable categórica, generamos 32 datos aleatorios de tres opciones(gato, perro y conejo). Para las variables categóricas son simplmente muestras de dos distribuciones. Entonces primero se generan tres vectores de la forma descrita anteriormente, llamados *categorica*, *numerica1* y *numerica2*.

```
set.seed(100)
categorica <- sample(c("gato", "perro", "conejo"), 32, replace = T)
numerica <- round(runif(32,30,80),2)
numerica2 <- round(rnorm(32, 40, 35))^2
```

Ahora, hay que "pegarle" los datos al objeto *sf* (mapamex_simple). La manera más fácil de añadirlos es con R base:

```
mapamex_simple$categorica  <- categorica
mapamex_simple$numerica <- numerica
```

Los datos también se pueden añadir con *dplyr*, usando *mutate*.

```
mapamex_simple <-
    mapamex_simple %>%
    mutate(categorica = categorica, numerica = numerica, numerica2 = numerica2)
```

Existen muchas otras formas de añadirle una columna de datos (por ejemplo con *rbind*), o si los datos vienen de otro *data frame*, se pueden usar alguna de las varias formas de juntar los *data frame* (por ejemplo *merge* o *join*).

El objeto *sf* con las columnas añadidas, termina siendo de esta forma:

```
head(mapamex_simple)
```

```
## Simple feature collection with 6 features and 6 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 911292 ymin: 692158.7 xmax: 3859532 ymax: 2349605
## Projected CRS: MEXICO_ITRF_2008_LCC
##   CVEGEO CVE_ENT               NOMGEO                       geometry categorica
## 1     01      01       Aguascalientes MULTIPOLYGON (((2492248 114...      perro
## 2     02      02      Baja California MULTIPOLYGON (((1208688 187...     conejo
## 3     03      03  Baja California Sur MULTIPOLYGON (((1711892 136...      perro
## 4     04      04             Campeche MULTIPOLYGON (((3709416 998...     conejo
## 5     05      05 Coahuila de Zaragoza MULTIPOLYGON (((2712006 174...       gato
## 6     06      06               Colima MULTIPOLYGON (((1157633 768...      perro
##   numerica numerica2
## 1    69.02      5476
## 2    74.21      1296
## 3    40.39      7921
## 4    45.35       484
## 5    46.53      3844
## 6    39.93       484
```

4.3 Graficar los mapas.

Finalmente llegamos a la sección en la que dibujamos los mapas. Es una introducción sencilla, con nuestros ridículos datos falsos, pero servirá bien como introducción para elaborar mapas con contenido más serio. Para graficar los datos se utiliza la librería *tmap*. La sintáxis de *tmap* es muy similar a la de *ggplot*. En toda esta sección se utilizará el objeto mapamex_simple con los datos añadidos. Este objeto (las coordenadas con los datos falsos añadidos) se puede cargar con el archivo maps_with_data.RData.

4.3.1 Dibujar las fronteras de los estados

El mapa más sencillo que se puede hacer es uno de todo el país con las siluetas de los estados. La librería *tmap* le va añadiendo información al mapa con funciones añadidas secuencialmente con un signo "+". Cada una de estas funciones añade información gráfica, a las que se les llama capas (layers). La función *tm_shape* define qué objeto se va a graficar. La primera capa que añadimos es la de las fronteras de los estados, la cual se añade con la función *tm_borders*:

```
# Llamar al objeto sf para graficar
tm_shape(mex_simple) +
    #graficar los datos geográficos, las fronteras de los estados
    tm_borders()
```

4.3.2 Graficar en el mapa una variable categórica.

Ahora ya podemos empezar a añadir más información en el mapa. La segunda capa que añadiremos es una que grafique otras variables en el mapa. Una forma de hacer esto es iluminando el interior de cada estado con un color que indique a qué categoría pertenece ese estado o en qué rango de valores de una variable numérica se encuentra. Para eso se utiliza la función *tm_fill*. En este primer ejemplo, el mapa se rellenará con la variable categórica, un color distinto para cada categoría. Para esto, los parámetros que usa la función *tm_fill* son el nombre de la variable (en este caso *col= "categorica*) y la opción *style ="cat"*, que indica que es una variable categórica.

```
tm_shape(mapamex_simple) +
    #dibujar fronteras
    tm_borders() +
    #rellenar con variable categorica, un color distinto por cada categoria
    tm_fill(col ="categorica", style = "cat")
```

También se puede usar la instrucción *tm_polygons()* que es equivalente a *tm_borders() + tm_fill*.

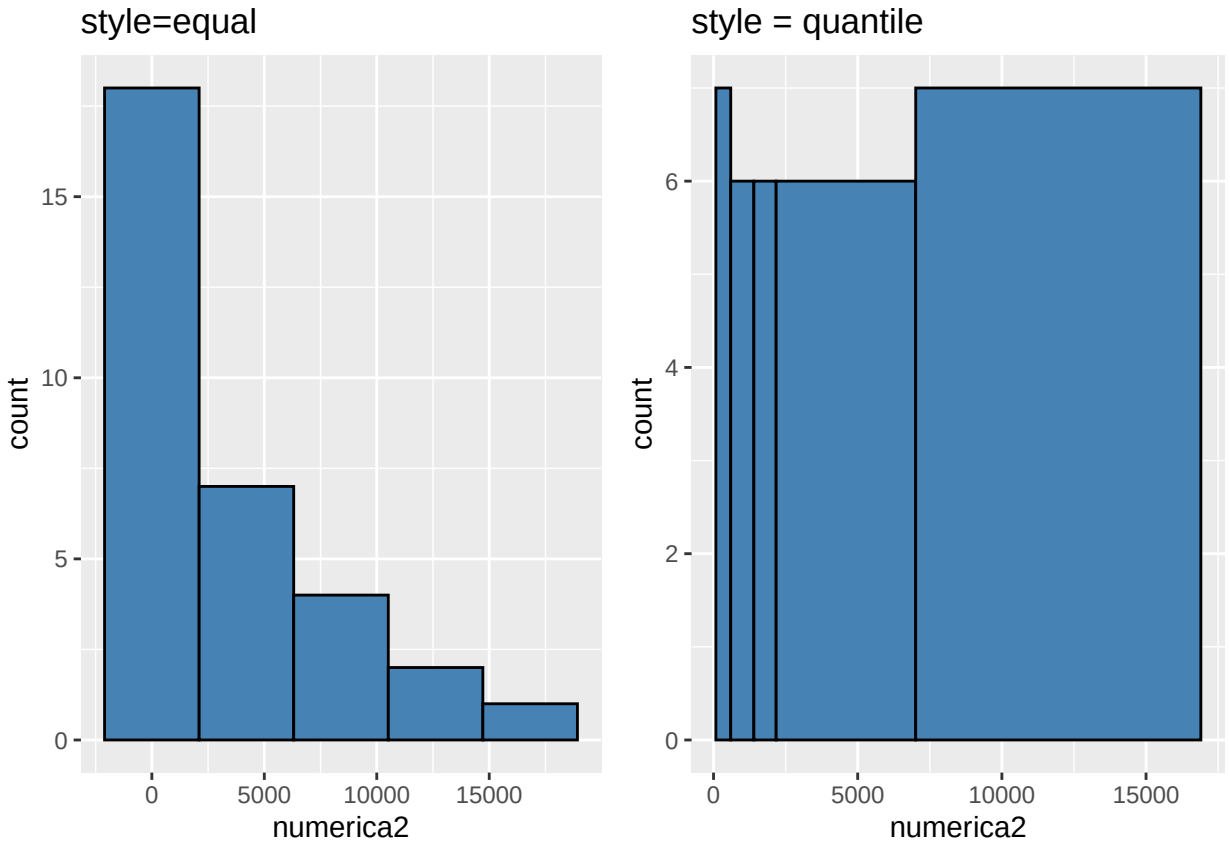
```
tm_shape(mapamex_simple) +
tm_polygons(col ="categorica", style = "cat")
```

4.4 Graficar en el mapa una variable numérica.

Para graficar una variable numérica, se pueden dividir los datos en rangos y graficarla como categórica (un tono de color para cada intervalo). Existen varias formas de determinar los intervalos, las más importantes son: dividir el rango en intervalos de igual longitud y dividir el rango en grupos con el mismo número de observaciones, aunque los intervalos no sean iguales. Para el primer caso, se usa la opción *style =“equal”* y para la segunda, *style = “quantile”*. Además también se define el número de intervalos entre los que se dividirá el rango. Si los datos no están muy sesgados, los resultados de ambos criterios para dividir el rango serán casi iguales. Si los datos están muy sesgados, la opción “quantile” puede ayudar a una mejor distribución de los colores.

Para entender mejor la diferencia entre las dos formas de dividir el rango, se muestran dos histogramas de la misma variable, una dividida en 6 grupos por tamaño de intervalo y otra dividido por número de observaciones.

En el caso de dividir el rango en intervalos y volverlo “discreto”, hay dos parámetros importantes que utiliza la función *tm_fill*: *n*, que determina en número de intervalos y *style* que determina el criterio para dividir el rango. Los más usuales son *eauql*, que divide el rango en intervalos de igual longitud y *quantile*, que divide el rango en 5 intervalos con el mismo número de observaciones. Se recomienda esta última opción si los datos están muy sesgados.

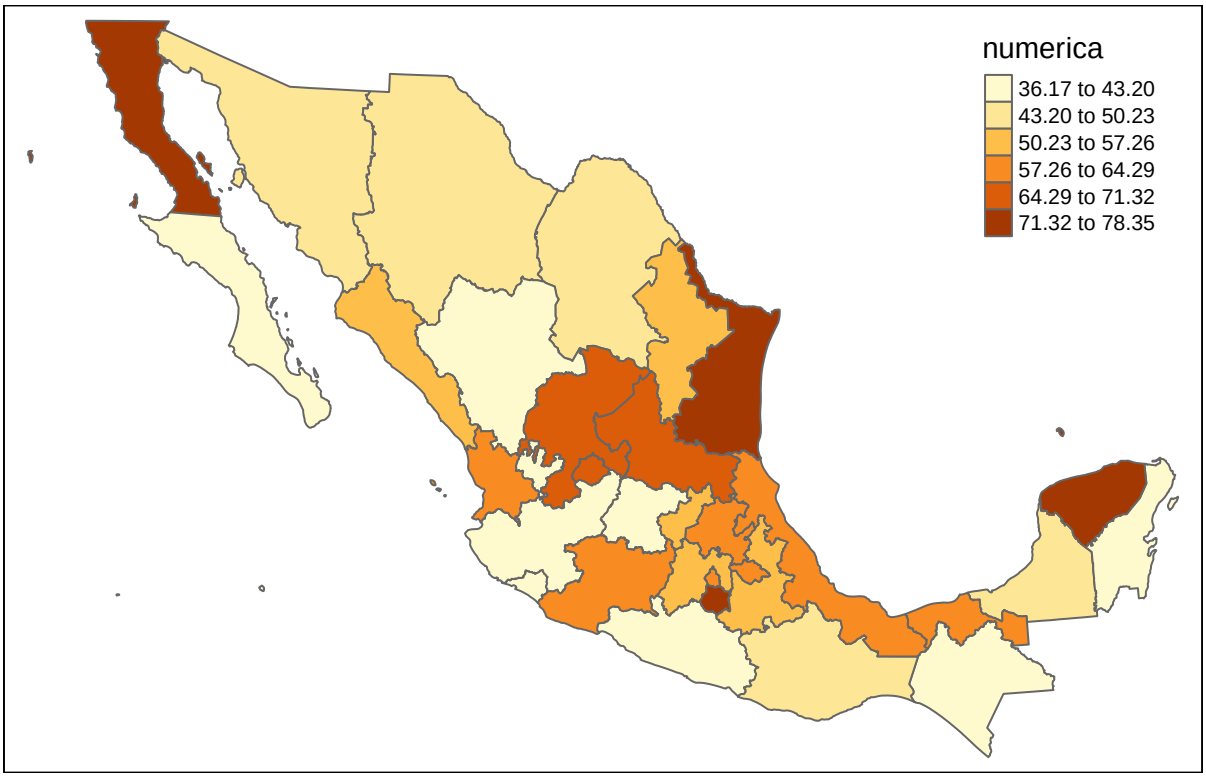


Aunque para todo efecto práctico dividir el rango en intervalos sea equivalente a discretizar la variable, las paletas de colores utilizadas para iluminar variables contínuas se compone de distintos tonos del mismo color, para indicar un aumento o disminución de un valor. En el caso de una variable intrínsecamente discreta, lo que se busca es usar colores muy distintos para crear contraste.

A continuación se muestran algunos ejemplos de estas opciones (*equal* y *quantile*).

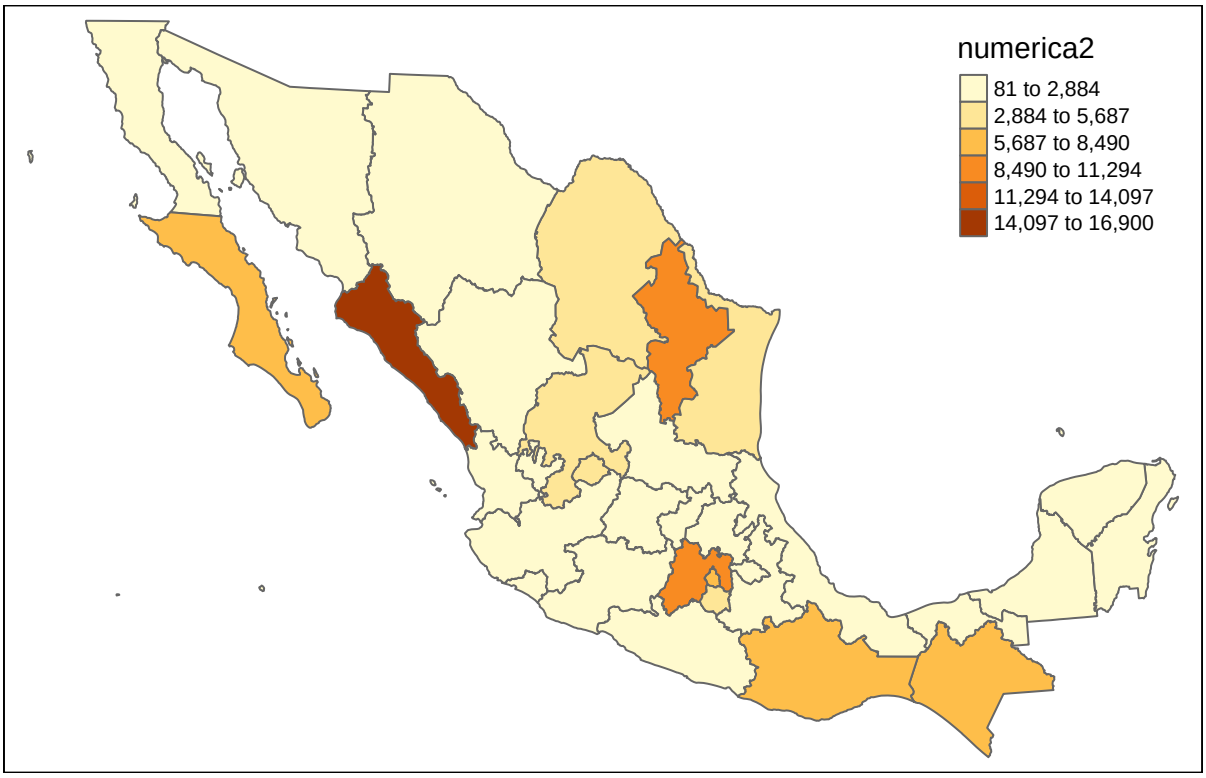
En el siguiente mapa se grafica la variable *numérica*, dividiendo el rango en 6 intervalos de la misma longitud, usando la opción *style = "equal*. Como estos datos están distribuidos uniformemente, esta opción funciona bien.

```
#variable numérica con rango dividido en 6 intervalos iguales


tm_shape(mapamex_simple) +
    #dibujar las fronteras de los estados
    tm_borders() +
    #rellenar los estados con la variable "numerica" dividida en 6 intervalos iguales
    tm_fill(col ="numerica", n=6, style = "equal")
```



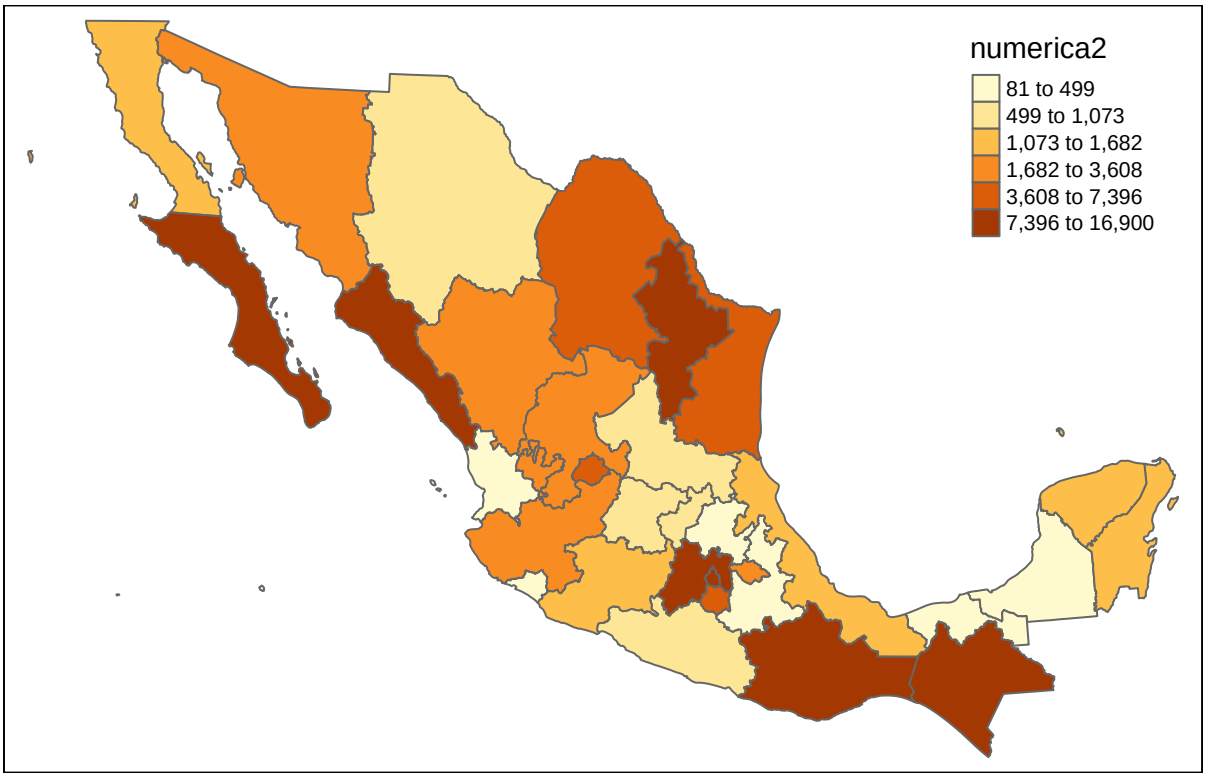
Los datos de la variable *categorica2* están muy sesgados, por lo que separarlos en rangos uniformes no resalta bien las diferencias.

```
tm_shape(mapamex_simple) +
    tm_borders() +
    tm_fill(col ="numerica2", n=6, style = "equal")
```



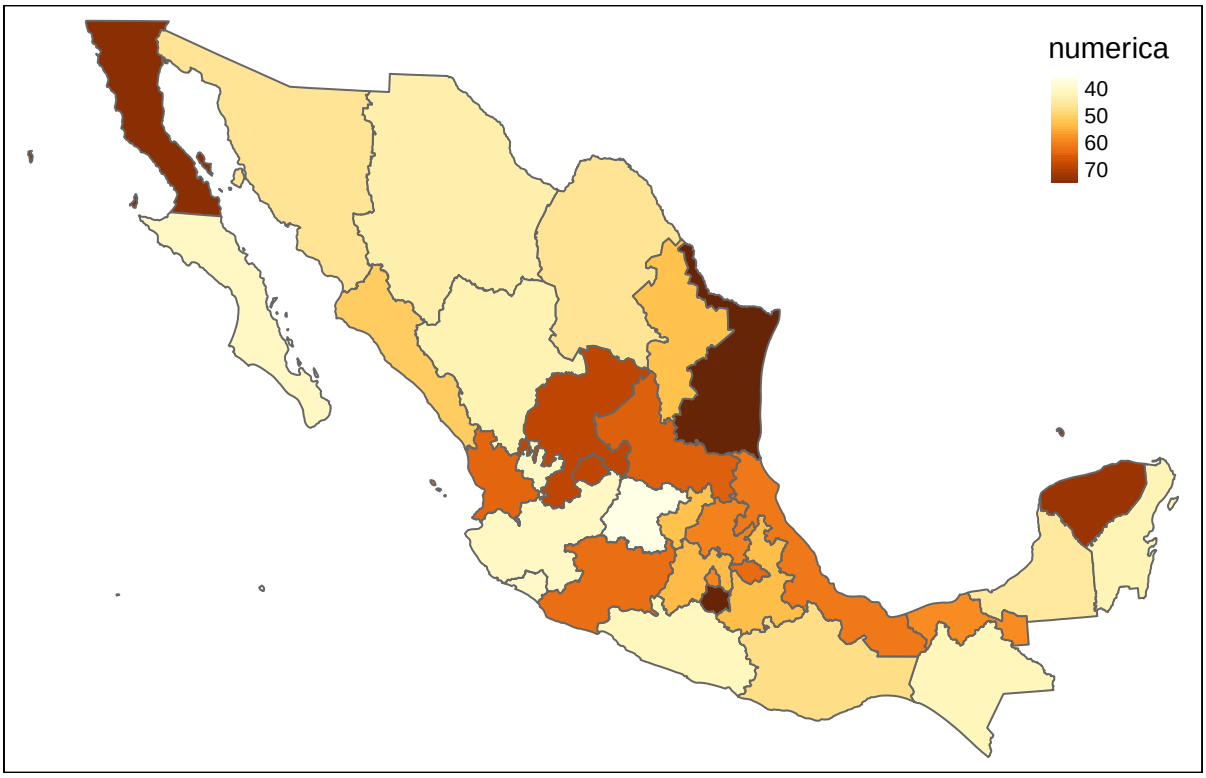
Repitiendo el mapa anterior, pero con el rango dividido en 6 intervalos con el mismo número de observaciones (opción *quantile*), tenemos mejores resultados:

```
#variable numérica con rango dividido en 4 intervalos con mismo número de datos


tm_shape(mapamex_simple) +
    tm_borders() +
    tm_fill(col ="numerica2", n=6, style = "quantile")
```



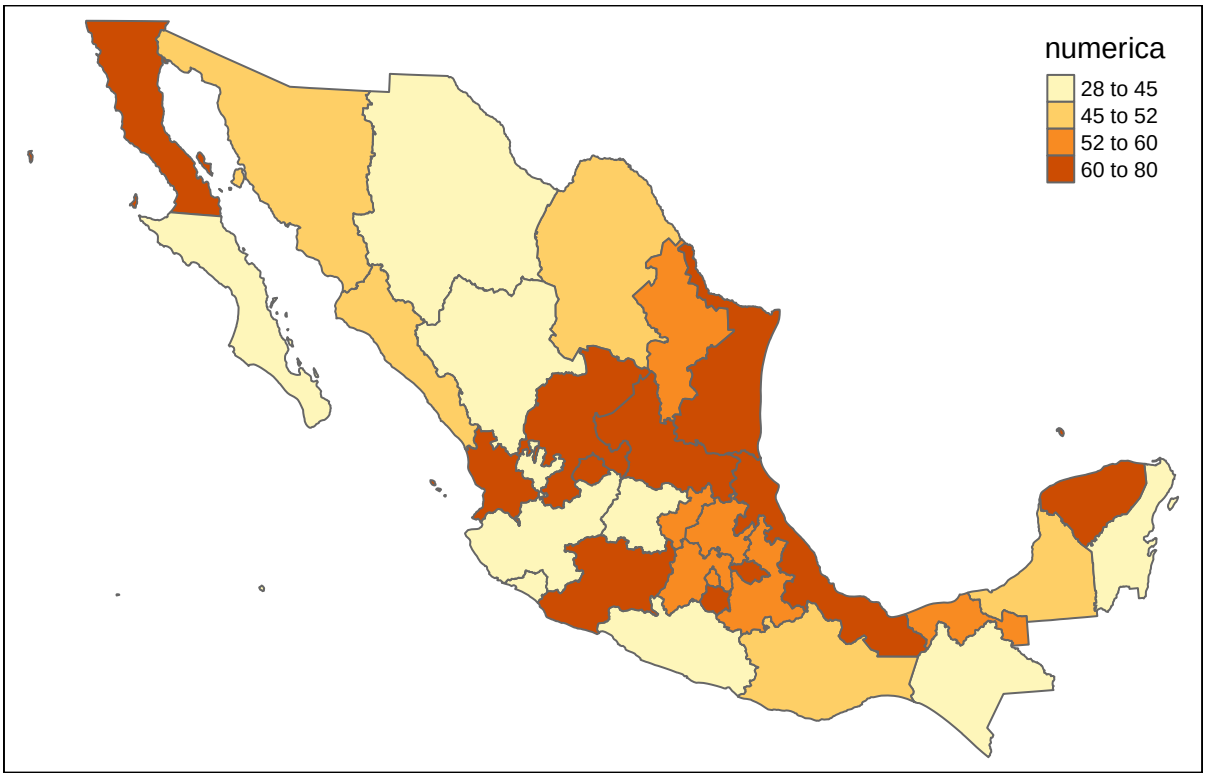
Otra forma de graficar una variable numérica es usando la opción “cont” (de continuous). El valor de la variable se toma de un gradiente “contínuo” de tonos.

```
tm_shape(mapamex_simple) +
    tm_borders() +
    tm_fill(col ="numerica",  style = "cont")
```



El rango también se puede dividir arbitrariamente, escogiendo manualmente los puntos de corte. Para hacerlo de esa forma, primero se define un vector con los puntos donde se quiere que se hagan las divisiones.

```
#Definir puntos donde se divide el rango
my_breaks = c(28, 45, 52, 60, 80)

tm_shape(mapamex_simple) +
    tm_borders() +
    tm_fill(col ="numerica",  breaks = my_breaks)
```



Para más opciones, ver https://geocompr.robinlovelace.net/adv-map.html

4.5 Elegir colores.

Los mapas que hemos hecho hasta ahora, los mapas se iluminaron con colores default que utiliza R. Estos colores también pueden ser elegidos por el usuario. Se pueden elegir manualmente uno por uno, o usando paletas profesionales.

4.5.1 Elegir colores de forma manual.

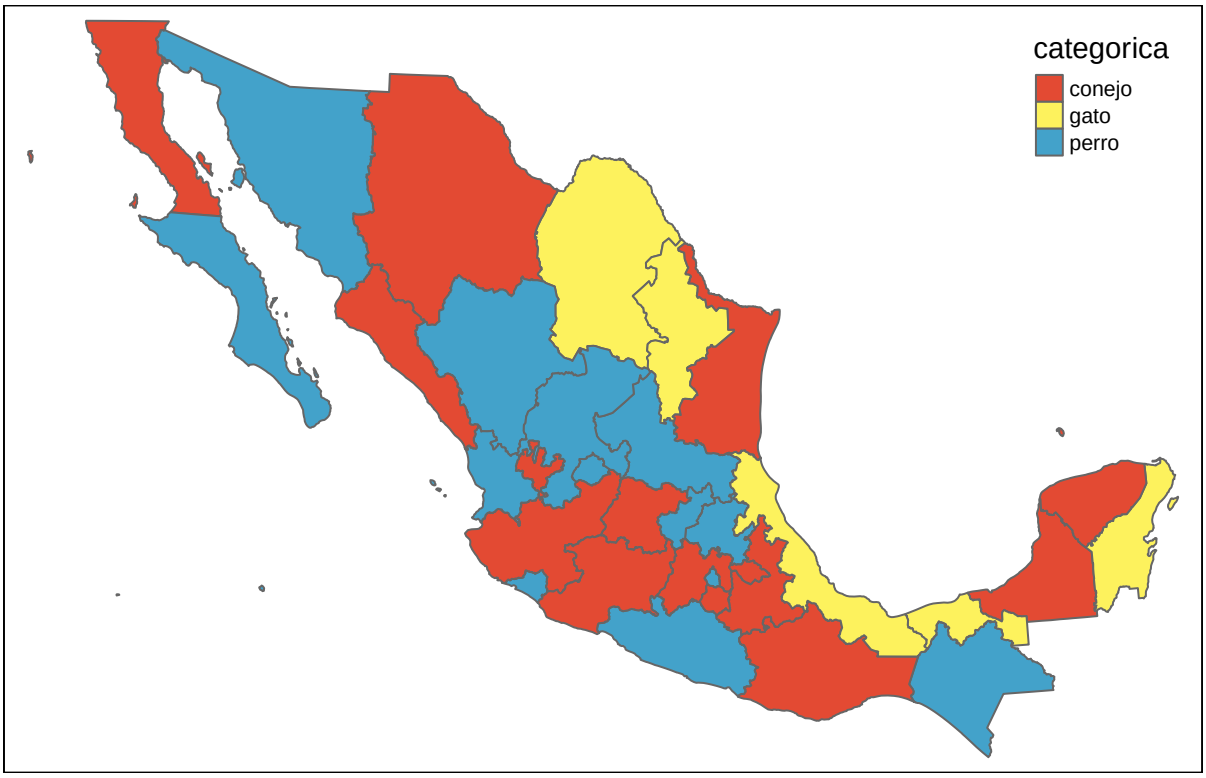
Los colores para el mapa se pueden elegir a través de paletas o manualmente. La forma más fácil de añadir colores es creando un vector con los nombres ó códigos hexadecimales de los colores que se quieren usar y pasarla como opción en tm_fill. En este sitio se puede encontrar el nombre de cada tono de color, así como las paletas de *RColorBrewster* de las que hablaremos en la siguiente sección. Los códigos hexadecimales se pueden encontrar en el aquí . En general se recomienda que para variables numéricas se utilicen paletas secuenciales, es decir, una secuencia de tonos del mismo color o de colores que cambien gradualmente, y para categóricas se recomiendan colores que contrasten.

Para iluminar el mapa de los colores elegidos por el usuario, primero se crea un vector con los nombres de los colores o los valores hexadecimales, y después se incluye este vector como opción en la función *tm_fill*, usando la opción *pal*. En el siguiente ejemplo se repite el mapa de la variable categórica, haciendo primero una paleta de colores, elegidos manualmente usando sus "nombres":

```
# Se definen los colores que se quieren usar
mypal <- c("plum2", "springgreen3", "steelblue2")

tm_shape(mapamex_simple) +
    tm_borders() +
# Se añade el vector con los colores con la opción pal
    tm_fill(col ="categorica", style = "cat", pal = mypal)
```

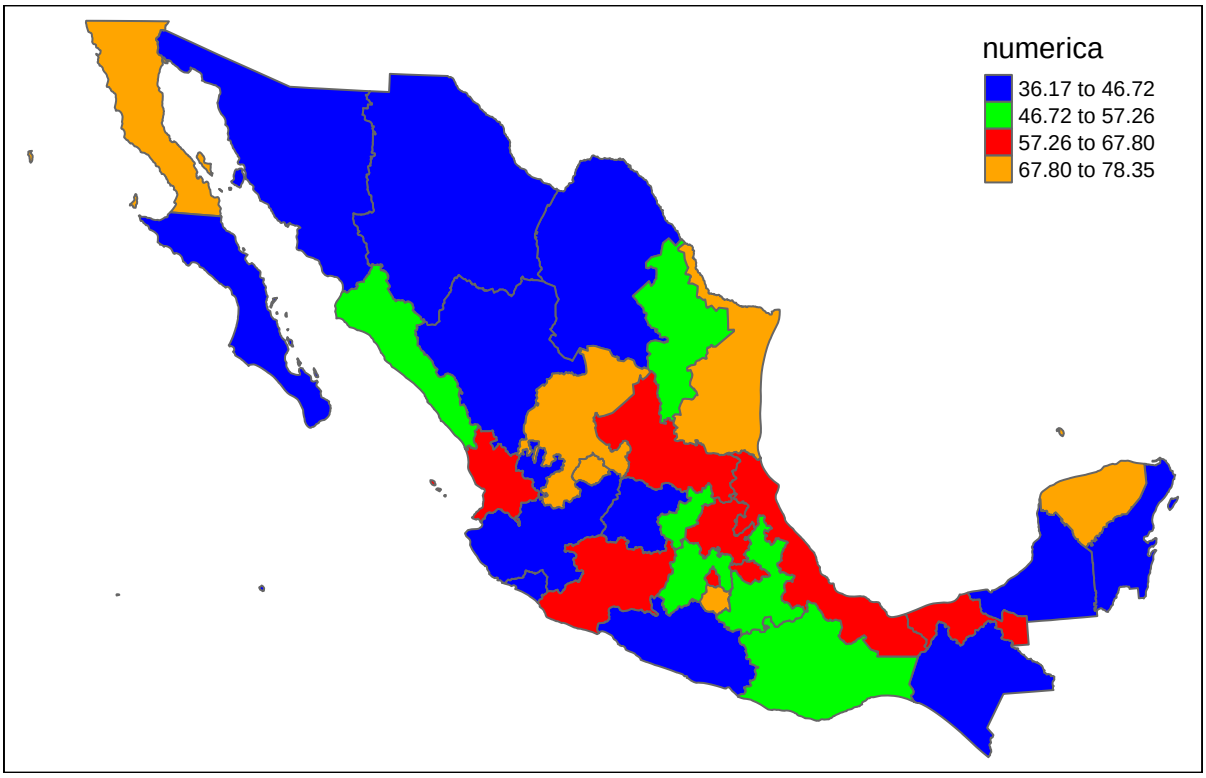
También se puede asignar un color específico para cada categoría:

```
#Cada categoría de la variable tiene asignado un color distinto
mypal <- c("conejo" = "#e34a33", "gato" = "#fdf25d", "perro"= "#43a2ca")
mapamex_simple %>%
    tm_shape() +
    tm_borders() +
    tm_fill(col ="categorica", style = "cat", pal = mypal)
```



También se puede elegir manualmente los colores de las variables numéricas. Se crea un vector con el número de colore igual al número de intervalos usados para dividir el rango. Los colores que se escogieron para este ejemplo no son los ideales para una variable numérica, son solamente para ilustrar el ejemplo. Se recomienda para variables numéricas una paleta secuencial. En la siguiente sección aprenderemos sobre tipos de paletas.

```
#Vector con 4 colores
mypal <-  c("blue", "green", "red", "orange")

tm_shape(mapamex_simple) +
    tm_borders() +
    #rango dividido en 4 intervalos
    tm_fill(col ="numerica", n=4, style = "equal", pal=mypal)
```

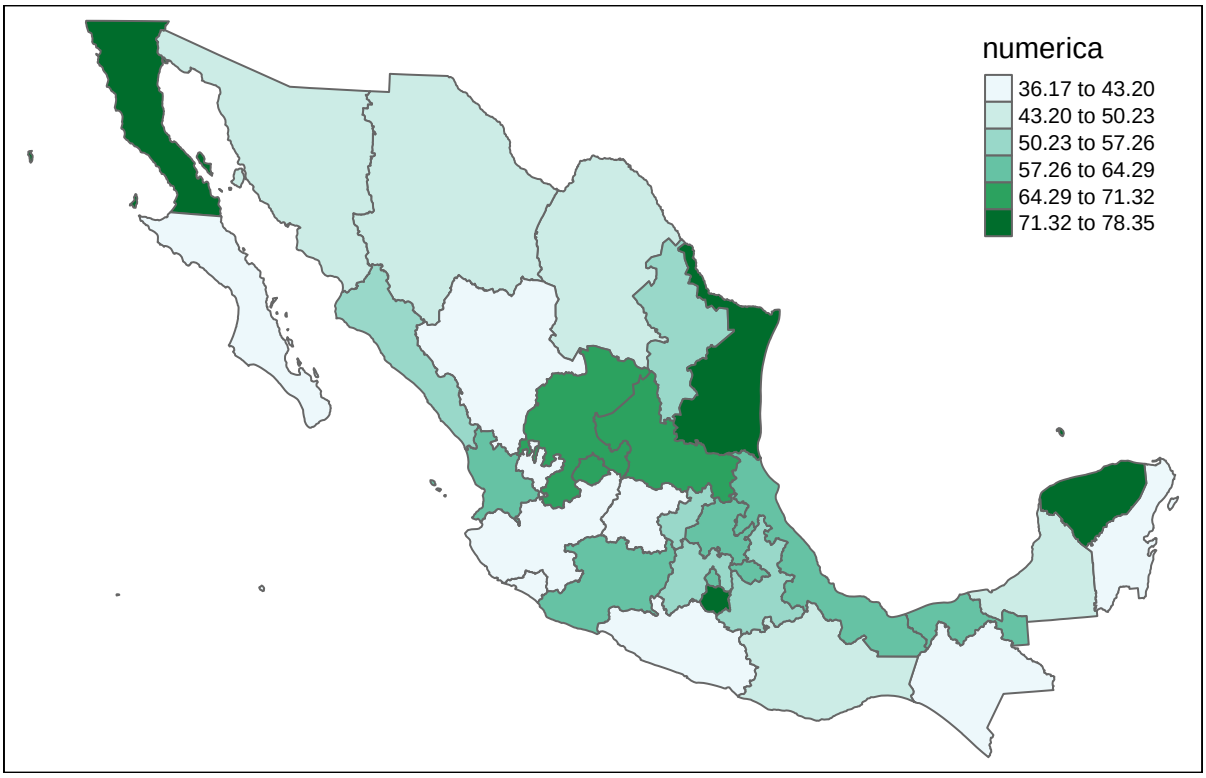


4.5.2 Especificar colores con Color Brewer.

Aunque se pueda especificar manualmente los colores, en la mayoría de los casos es mejor escoger los colores con paletas elaboradas profesionalmente. Existen varios sitios en internet en donde se pueden encontrar paletas elaboradas específicamente para comunicar mejor cada tipo de información, mismas que se pueden usar en gráficas o en mapas. Con la librería *RColorBrewer* se pueden usar las paletas Color Brewer, utilizadas en cartografía. Para ver estas paletas en mapas, ir a esta liga.

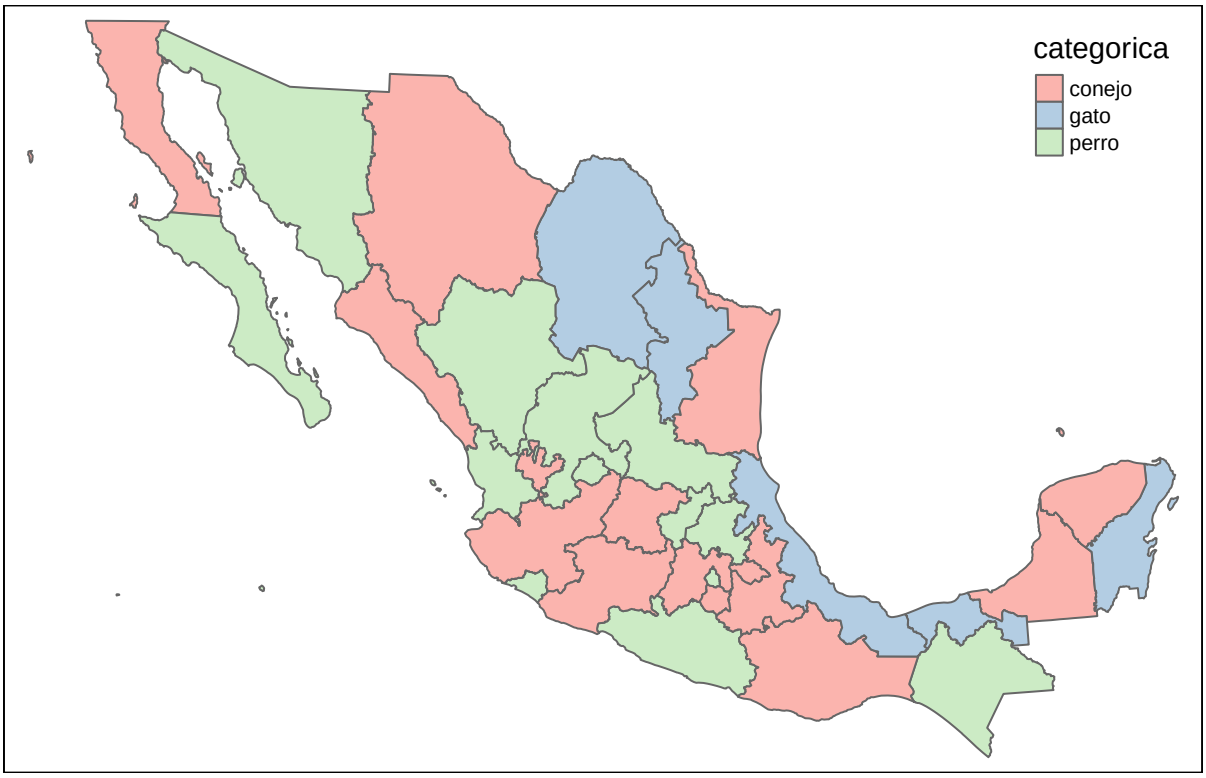
Para las variables numéricas se recomienda usar las paletas secuenciales. Estas contienen secuencias de tonos del mismo color o de varios colores que van de un tono más claro a uno más obscuro, dando al usuario información inmediata sobre si se trata de un valor mayor o menor sin tener que consultar la leyenda. En el ejemplo usamos una paleta secuencial en tonos de tonos de verde. La paleta se define con la función *brewer.pal*. Los nombres de las paletas se pueden ver aquí.

```
# Definir la paleta con color brewer, con 6 tonos.
mypal <- brewer.pal(6,"BuGn")


tm_shape(mapamex_simple) +
    tm_borders() +
    tm_fill(col ="numerica",  n=6, style = "equal", pal = mypal)
```



Para una variable categórica, usamos colores que contrasten:

```
#Paleta con 3 tonos contrastantes

mypal <- brewer.pal(3, "Pastel1")

tm_shape(mapamex_simple) +
    tm_borders() +
    tm_fill("categorica", pal = mypal,   style ="cat")
```



Para ver más sobre colores en R, visitar esta liga

4.5.3 Títulos, leyndas y demás texto.

Existe una gran cantidad de opciones para la personalización de las leyendas y títulos en los mapas. En esta sección se dará una brefe introducción de cómo añadir títulos y personalizarlos, junto con las leyendas, cambiando la posición, el tamaño color y el tipo de letra.

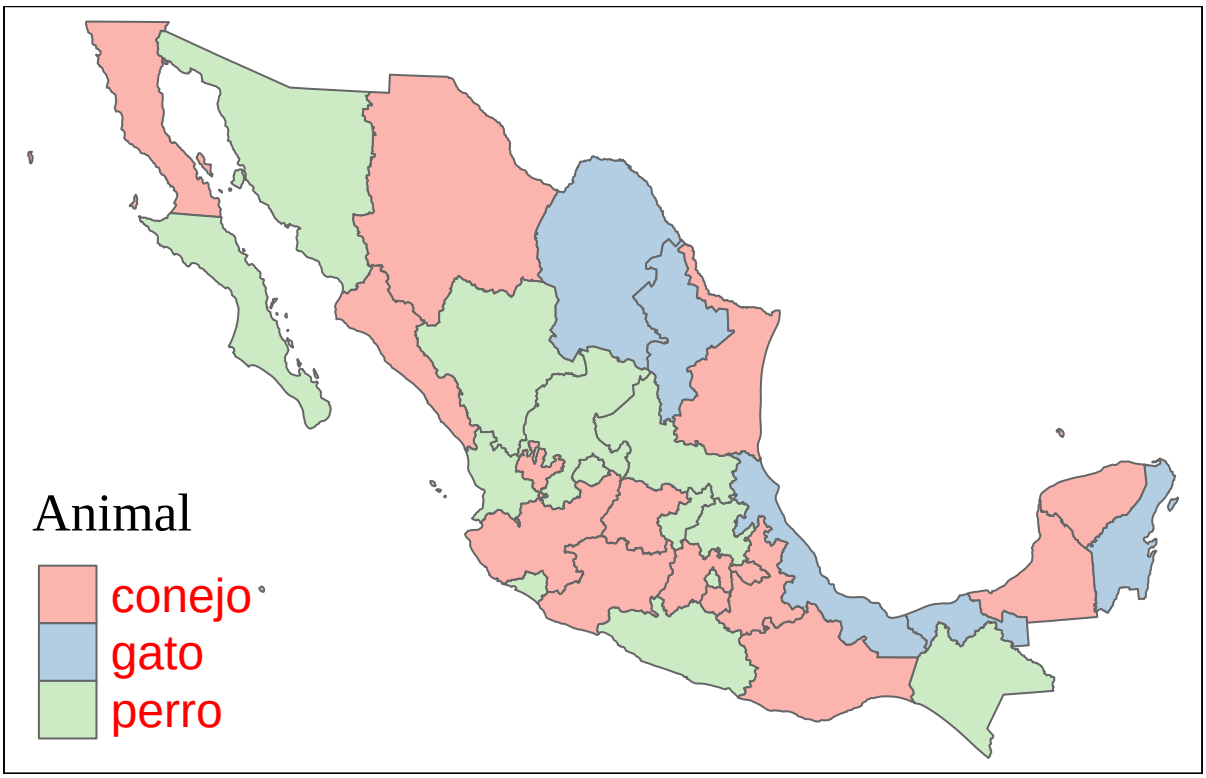
Cuando se hace un mapa como los que hemos estado haciendo, se genera una leyenda que indica el nombre de la variabl como título de la leyenda y que valor de dicha variable indica dada color. Esta leyenda nos indica a qué categoría pertenece cada estado o a que rango de valores. Cuando se genera la leyenda, el título de la leyenda es el nombre de la variable (columna). Este título se puede cambiar a algo que comunique mejor lo que representa dicha variable. En nuestro ejemplo, la variable categórica se llama "categorica", pero los valores son nombre de animales. Para cambiar el título de la leyenda, se pone como opción en *tm_fill(..., title = "Animal)* En este cambiaremos el título de la leyenda a"Animal".

También se puede cambiar de posición, tamaño de letra, color de letra etc. Esto se hace añadiendo otra capa con una función llamada *tm_legend*. Algunos aspectos de la leyenda que se pueden cambiar son:

- La posición de la leyenda. Usando un vector con 2 componentes entre 0 y 1, se indica el lugar dentro del panel del mapa donde se quiere poner la leyenda, siendo *(0,0)* la esquina inferior izquierda y *(1,1)* la esquina superior derecha. Para esto se usa la opción *position*. También se puede usar un vector que tenga como el primer valor "left", "center" o "right" y como segundo valor "top" o "bottom".
- El tamaño de la letra para el título de la leyenda se determina con *title.size*
- El tamaño del texto que indica las categorías se indica con *text.size*
- Los colores de los textos se indican con las opciones *text.color* y *title.color*.

```
mypal <- brewer.pal(3, "Pastel1")

tm_shape(mapamex_simple) +
    tm_borders() +
```

```
    tm_fill("categorica",, pal = mypal,   style ="cat", title = "Animal") +

    tm_legend(position = c("left", 'bottom'),
              title.size = 2,
              text.size = 1.5,
              text.color = "red",
              title.fontfamily = "serif")
```



El título del mapa se puede definir en la función *tm_layout*. Al igual que sucede con *ggplot* existen varias maneras de hacer lo mismo. Por ejemplo, las leyendas también se podrían manejar desde la función *tm_layout*, pero por el momento no lo haremos así para no crear más confusión.

En el siguiente ejemplo se dibuja un mapa en el que se incluye un título en el que se personaliza el tamaño de letra, el color y la posición (centrado). Usando la función *tm_layout* con la opción *main.title = "Título del mapa* se crea un título afuera del panel del mapa.

```
tm_shape(mapamex_simple) +
    tm_borders() +
    tm_fill("categorica", pal = mypal,   style ="cat", title = "Animal") +
    # personalizar la leyenda
    tm_legend(position = c(0.1,0.1),
              title.size = 2,
              text.size = 1.5,
              text.color = "red",
              title.fontfamily = "serif") +
    #personalizar el título
    tm_layout(main.title = "Mascotas en México",
              main.title.size = 3,
              main.title.color = "green",
              main.title.position = "center")
```

El título también se puede poner adentro del panel del mapa. En este case se usa la opción *title* y el lugar donde se pone el título se determina con un vector, de la misma forma que el la leyenda. Tomar en cuenta que el vector (ya sea con coordenadas o usando "top" etc) determina el lugar donde empieza el título.

```
tm_shape(mapamex_simple) +
    tm_borders() +
    tm_fill("categorica", palette = mypal,   style ="cat", title = "Animal") +
    tm_legend(position = c(0.01,0.1),
              title.size = 2,
              text.size = 1.5,
              text.color = "red",
              title.fontfamily = "serif") +
    tm_layout(title = "Mascotas en México",
              title.size = 2,
              title.color = "blue",
              title.position = c(0.2, 0.95)
    )
```

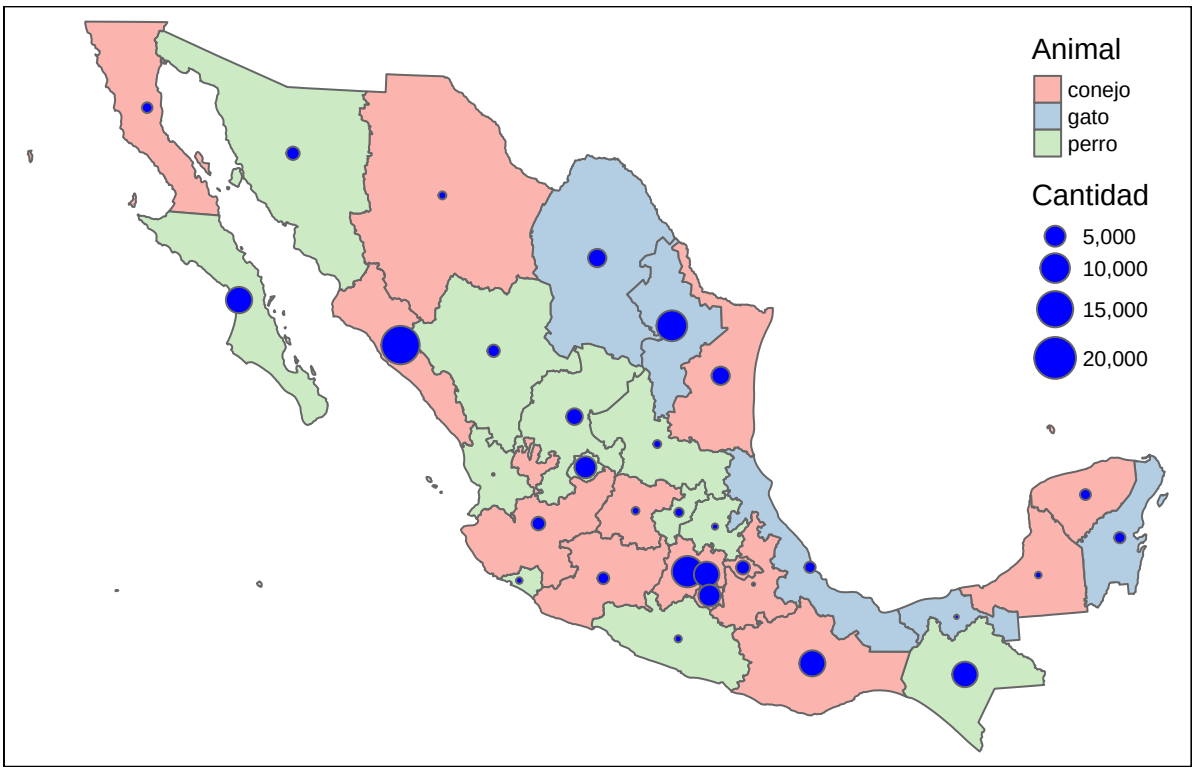
4.5.4 Graficar variables en el mapa con otros elementos gráficos.

Además de iluminar cada polígono (en este caso, estado) para indicar los valores de una variable, se pueden también usar símbolos en los que el color o el tamaño del símbolo indiquen el valor de la variable. De esta forma, se puede graficar más de una columna de datos en un mismo mapa.

En el mapa siguiente se grafican dos variables. La variable categórica se grafica rellenando cada estado de un color diferente y una variable numérica se grafica añadiendo un círculo cuyo tamaño varía con el valor de la variable. Para esto se añade otra capa con la función *tm_bubbles*. Las opciones para *tm_bubbles* son:

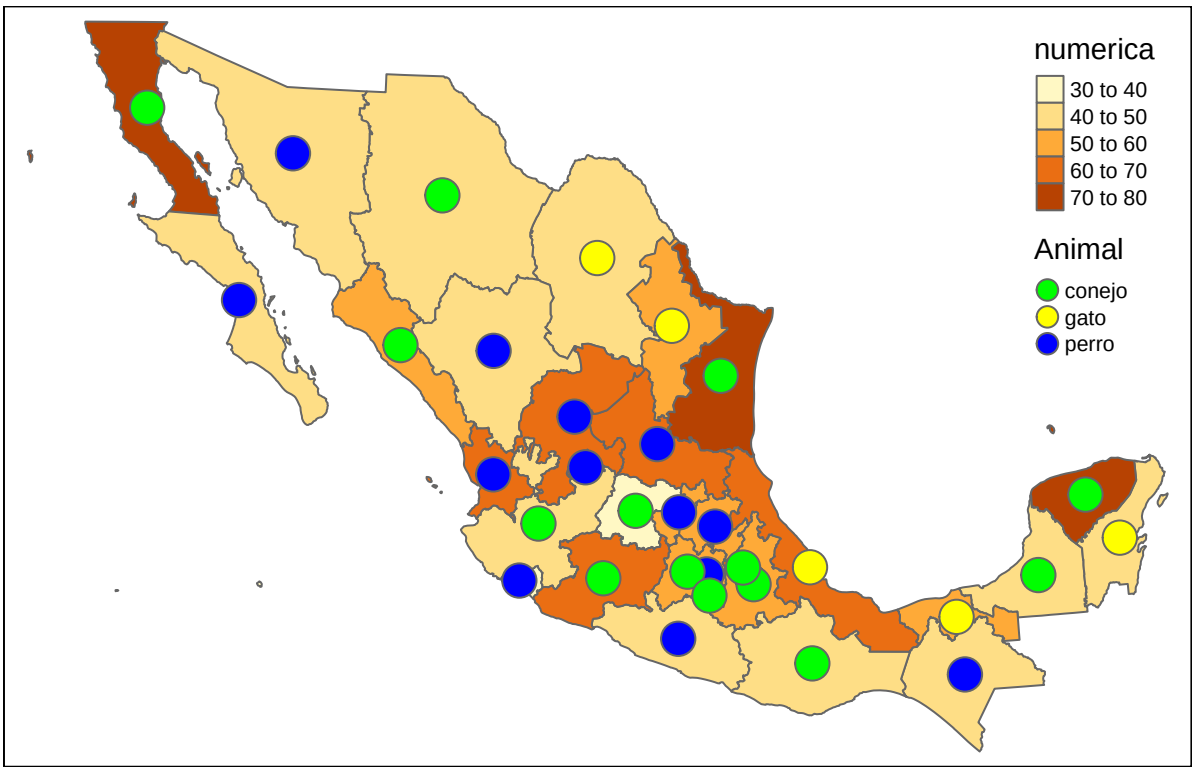
- *size* Si se se desea que el tamaño varíe con una variable, esta función se iguala al nombre de la columna.
- *title.size* Título de la leyenda, cuando lo que se varía es el tamaño
- *col* Color. Se le puede asignar un color determinado.
- *legend.size.is.portrait* Cuando se grafica el círculo proporcional a la variable, por default coloca la leyenda en forma horizontal. En este caso añadimos esta opción para que quede vertical, igual que la leyenda de la otra variable, pero es opcional.

```
tm_shape(mapamex_simple) +
    tm_borders() +
    tm_fill(col ="categorica",
            palette = mypal,
            style ="cat",
            title = "Animal") +

    tm_bubbles(size= "numerica2",   #círculo grafica variable numérica
               legend.size.is.portrait = TRUE, #leyenda vertical
               title.size = "Cantidad",   #título de la leyenda
               col="blue") +              #color de los círculos.

    tm_layout(main.title = "Mascotas en México")
```



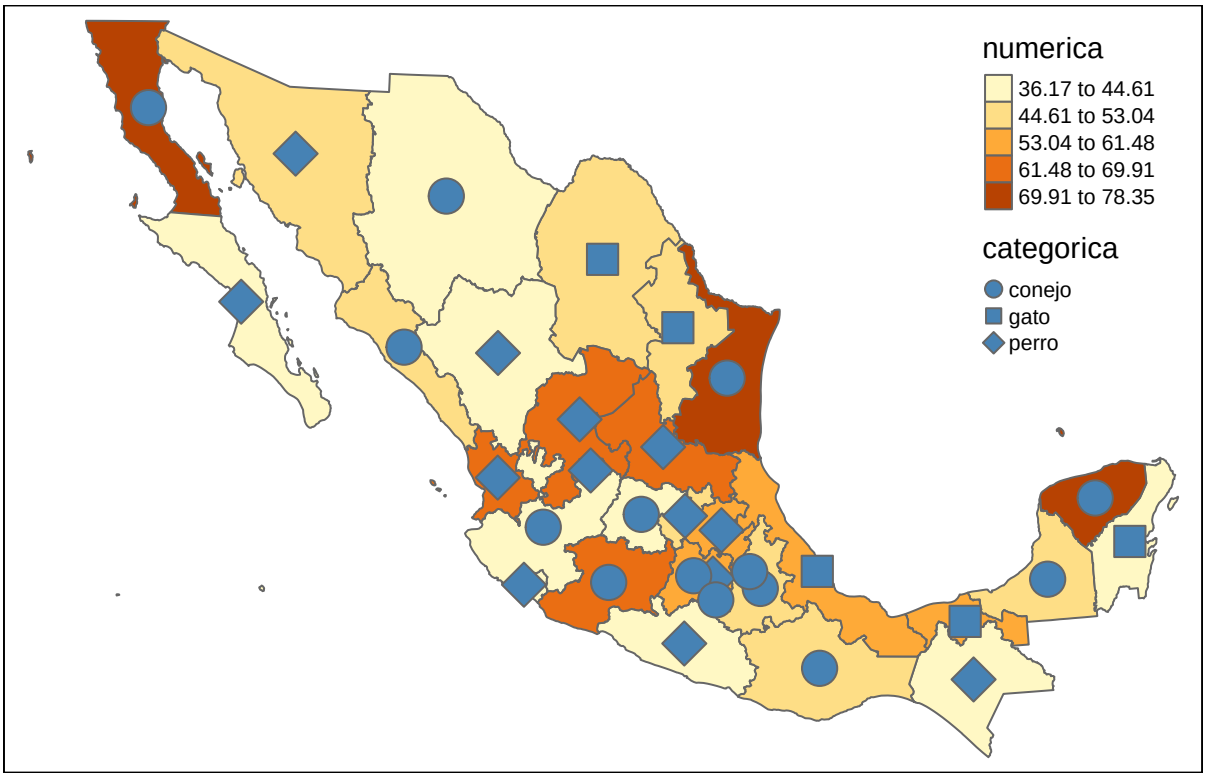
También se pueden usar los círculos para variables categóricas, usando un color diferente para indicar cada catergoría. También se utiliza la función *tm_bubbles*. En este caso las opciones son *col*= (color) igual al nombre de la variable, *title.col* para el título de la leyenda. También se puede usar una paleta elegida por el usuario y personalizar el tamaño (dado que en este caso, los círculos son iguales).

```
mypal = c("green", "yellow", "blue")

tm_shape(mapamex_simple) +
    tm_borders() +
    tm_fill(col ="numerica") +
    tm_bubbles(col= "categorica",      # El color lo determina la variable categórica
               title.col = "Animal",   #Título de la leyenda
               pal = mypal,
               size=0.8) +             #Se puede elegir el tamaño
    tm_layout(main.title = "Mascotas en México")
```



Y en este último ejemplo se muestra otra forma de graficar una variable categórica. En este caso, lo que cambia con la categoría es la forma de el símbolo, usando la opción *shape*. Aunque cambia la forma, el usuario puede elegir el tamaño y el color.

```
tm_shape(mapamex_simple) +
    tm_borders() +
    tm_fill(col ="numerica",
            style ="equal" ) +
    tm_symbols(shape= "categorica",
               legend.size.is.portrait = TRUE,
               col = "steelblue",
               size = 1.5    )
```



4.6 Salvar los mapas

Después de crear los mapas, se pueden salvar en varios formatos. Lo primero que se tiene que hacer es crear un objeto con el mapa. Esto se hace simplemente asignando el mapa a un objeto antes de crearlo. En este ejemplo lo llamamos *mimapa* y lo asignamos añadiendo la instrucción *mimapa* <- antes de crear el mapa:

```
mimapa <-
    tm_shape(mapamex_simple) +
    tm_borders() +
    tm_fill(col ="categorica",
            palette = mypal,
            style ="cat",
            title = "Animal") +

    tm_bubbles(size= "numerica2",    #círculo grafica variable numérica
               legend.size.is.portrait = TRUE, #leyenda vertical
               title.size = "Cantidad",    #título de la leyenda
               col="blue") +               #color de los círculos.

    tm_layout(main.title = "Mascotas en México")
```

Para salvarlo se usa la función *tmap_save*. La forma más básica de crear una imagen (jpg o png) es usando la función *tmap_save* con dos argumentos: el nombre del objeto (mimapa en este caso) y el nombre del archivo que se va a crear, entre comillas, pudiendo incluir toda la ruta ("path", nombre del subdirectorio).

```
tm_save(mipama, "mi_mapa.jpg")
tm_save(mipama, "mi_mapa.png")
```

Se puede variar el tamaño final de muchas formas. La más fácil es elegir el ancho o largo de la imagen, en número de pixeles. La otra dimensión se ajusta automáticamente, manteniendo las proporciones:

```
tmap_save(mi_mapa, "mimapa2.jpg",width=4000)
```

5 Para aprender más.

Esta es una pequeña muestra de lo que se puede hacer con los mapas.

Existen muchos tutoriales para aprender más de *sf*, *tmap* y mapas en R. Una introducción fácil, con muchos ejemplos, se puede encontrar aquí y aquí. Una guía muy completa se encuentra en este libro.