Topic 3: Choropleth Mapping and Analysis

Dr. Kam Tin Seong
Assoc. Professor of Information Systems

School of Computing and Information Systems,
Singapore Management University

2019/11/25 (updated: 2021-05-11)

Content

- Choropleth Mapping with in R
  - Classification of maps
  - Data Classification
  -
- Analytical Mapping Techniques
  - Percentile map
  - Box map
  - Standard deviation map
  - Mapping rates

Choropleth Map

A choropleth map is a type of thematic map in which areas are shaded or patterned in proportion to a statistical variable that represents an aggregate summary of a geographic characteristic within each area, such as population or per-capita income.

Basic Choropleth Mapping using tmap

- In this hands-on exercise, you will learn how to plot choropleth maps by using **tmap** package.

- By the end of this hands-on exercise, you will be able:

    - to import an aspatial data in R by using **readr** pakage,
    - to import geospatial data (ESRI shapefile) into R as simple feature objects using *sf* package,
    - to perform data wrangling using *dplyr* and *tidyr* packages,
    - to plot choropleth maps using *tmap* package.

Getting Started

For this in-class exercise, the following R packages are needed:

- sf,
- tmap, and
- tidyverse

Write a code chunk to install and launch these packages in R.

The code chunk should look similar to below:

```r
packages = c('sf', 'tmap', 'tidyverse')
for (p in packages){
  if(!require(p, character.only = T)){
    install.packages(p)
  }
  library(p,character.only = T)
}
```

Importing geospatial data

In this exercise, URA's Master Plan 2014 Subzone Boundary (i.e. MP14_SUBZONE_WEB_PL) will be used.

- Show the code chunk used to import the geospatial data into R as an sf data.frame. Name the sf data.frame *mpsz*.

The code chunk

```
mpsz <- st_read(dsn = "data/geospatial",
                layer = "MP14_SUBZONE_WEB_PL")
```

```
## Reading layer `MP14_SUBZONE_WEB_PL' from data sour
## Simple feature collection with 323 features and 15
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 2667.538 ymin: 15748.72 xmax:
## Projected CRS: SVY21
```

Importing Attribute Data into R

In this exercise, you are required to:

- download the latest *Singapotre Residents by Planning Area/Subzone, Age Group, Sex and Type of Dwelling, June 2011-2020* data from Department of Statistics, Singapore home page.

- parse the downloaded data into R as a tibble data.frame. Call the tibble data.frame *popdata*.

The code chunk is:

```
popdata <- read_csv("data/aspatial/respopagesex
```

Data Prepation

You are required to prepare a data table with year 2020 values. The data table should include the variables PA, SZ, YOUNG, ECONOMY ACTIVE, AGED, TOTAL, DEPENDENCY.

- YOUNG: age group 0 to 4 until age groyup 20 to 24,
- ECONOMY ACTIVE: age group 25-29 until age group 60-64,
- AGED: age group 65 and above,
- TOTAL: all age group, and
- DEPENDENCY: the ratio between young and aged against economy active group.

The code chunk is:

```
popdata2020 <- popdata %>%
  filter(Time == 2020) %>%
  group_by(PA, SZ, AG) %>%
  summarise(`POP` = sum(`Pop`)) %>%
  ungroup()%>%
  spread(AG, POP) %>%
  mutate(YOUNG = rowSums(.[3:6])
         +rowSums(.[12])) %>%
mutate(`ECONOMY ACTIVE` = rowSums(.[7:11])+
rowSums(.[13:15]))%>%
mutate(`AGED`=rowSums(.[16:21])) %>%
mutate(`TOTAL`=rowSums(.[3:21])) %>%
mutate(`DEPENDENCY` = (`YOUNG` + `AGED`)
/`ECONOMY ACTIVE`) %>%
  select(`PA`, `SZ`, `YOUNG`,
       `ECONOMY ACTIVE`, `AGED`,
       `TOTAL`, `DEPENDENCY`)
```

Georelation Join

Perform georelation join by using planning sub-zone name as the unique identifier.

The code chunk is:

```
popdata2020 <- popdata2020 %>%
  mutate_at(.vars = vars(PA, SZ),
          .funs = funs(toupper)) %>%
  filter(ECONOMY ACTIVE > 0)
mpszpop2020 <- left_join(mpsz, popdata2020,
               by = c("SUBZONE_N" = "SZ"))
```

Joining the attribute data and geospatial data

Next, *left_join()* of **dplyr** is used to join the geographical data and attribute table using planning subzone name e.g. *SUBZONE_N* and *SZ* as the common identifier.

```
mpsz_popdata2020 <- left_join(mpsz,
                              popdata2020,
                              by = c("SUBZONE_N" = "SZ"))
```

Plotting a choropleth map quickly by using *qtm()*

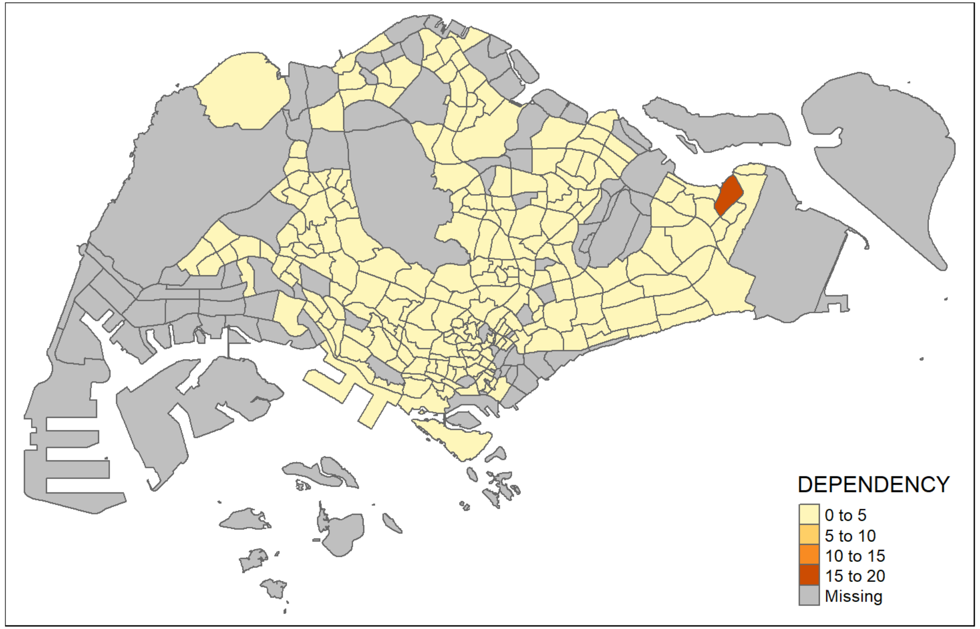
The easiest and quickest to draw a choropleth map using **tmap** is using *qtm()*. It is concise and provides a good default visualisation in many cases.

The code chunk below will draw a cartographic standard choropleth map as shown below.

```
tmap_mode("plot")
qtm(mpsz_popdata2020,
    fill = "DEPENDENCY")
```

Things to learn from the code chunk above:

- *tmap_mode()* with "plot" option is used to produce a static map.
- *fill* argument is used to map the attribute (i.e. DEPENDENCY)



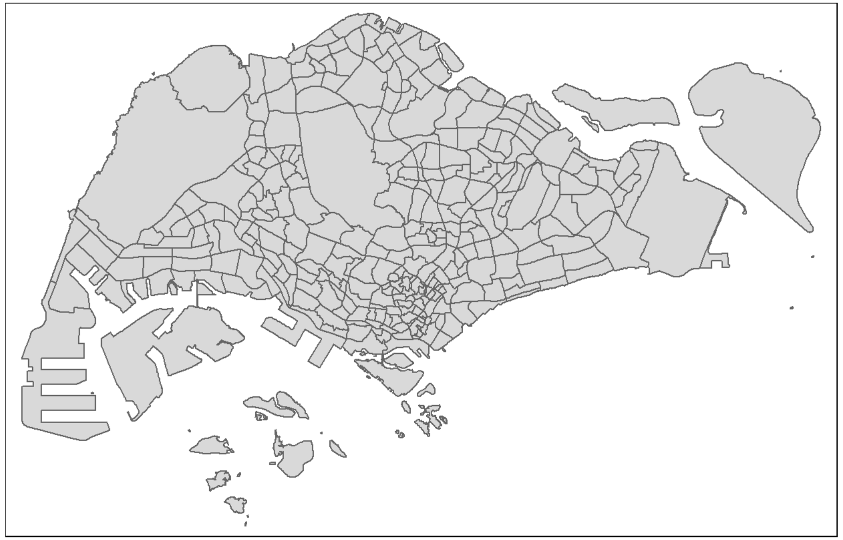
Drawing a base map

The basic building block of **tmap** is *tm_shape()* followed by one or more layer elemments such as *tm_fill()* and *tm_polygons()*.

In the code chunk below, *tm_shpae()* is used to define the input data (i.e *mpsz_agmale2018*) and *tm_polygons()* is used draw the planning subzone polygons

```
tm_shape(mpsz_popdata2020) +
  tm_polygons()
```

Be warned: The "+" sign should be place at the end of a code line and not at the front of a code line.



But why the figure is not sharp?

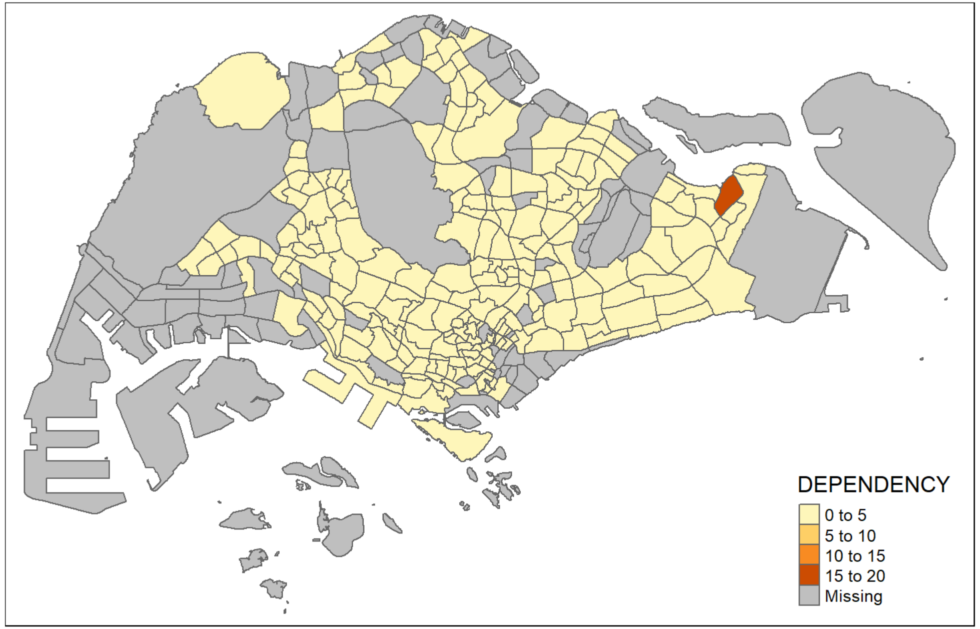
Drawing a choropleth map using *tm_polygons()*

To draw a choropleth map showing the geographical distribution of a selected variable by planning subzone, we just need to assign the target variable such as *DEPENDENCY* to *tm_polygons()*.

```
tm_shape(mpsz_popdata2020)+
  tm_polygons("DEPENDENCY")
```

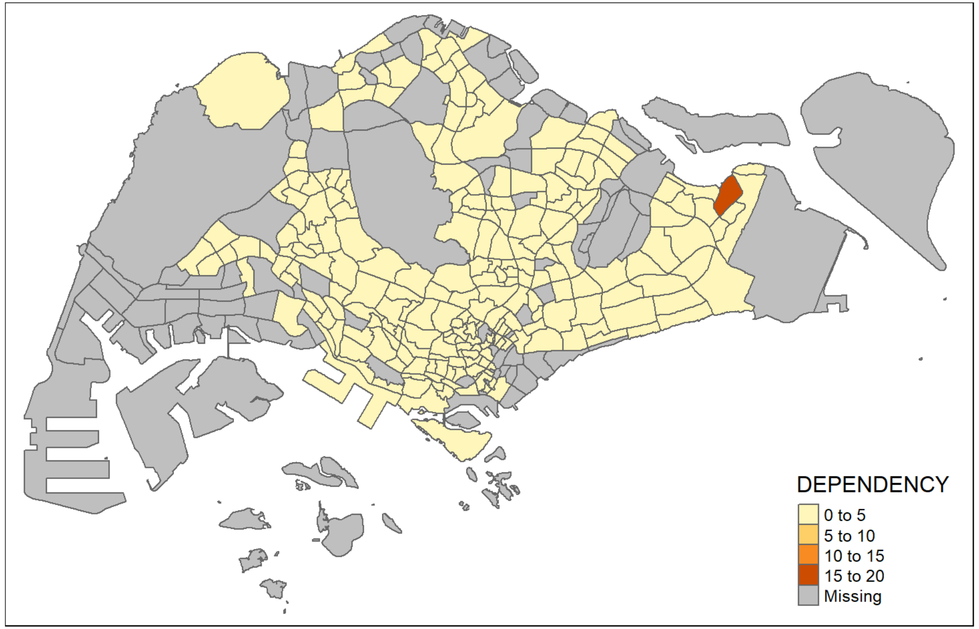
Things to learn from *tm_polygons()*:

- By default, 5 bins will be used.
- The default data classification method used is called "pretty".
- The default colour scheme used is "YlOrRd" of ColorBrewer. You will learn more about the color palette later.
- By default, Missing value will be shaded in gray.



Now it looks shaper

> {r, echo=TRUE, eval=FALSE, message=FALSE, warning=FALSE, fig.retina=3}



- The default dpi = 72. The default fig.retina = 2. That means the dpi is 72 x 2 = 144

- That means the dpi is 72 x 3 = 216

- out.width = fig.width * dpi / fig.retina

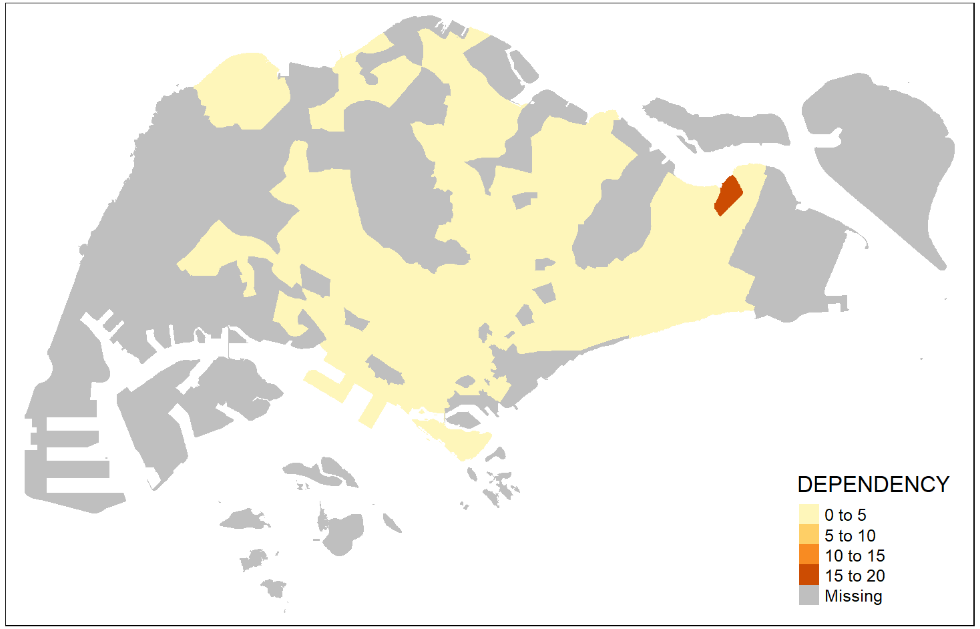
Drawing a choropleth map using *tm_fill()* and *tm_border()**

Actually, *tm_polygons()* is a wrapper of *tm_fill()* and *tm_border()*. *tm_fill()* shades the polygons by using the default colour scheme and *tm_borders()* adds the borders of the shapefile onto the choropleth map.

The code chunk below draw a choropleth map by using *tm_fill()* alone.

```
tm_shape(mpsz_popdata2020)+
  tm_fill("DEPENDENCY")
```

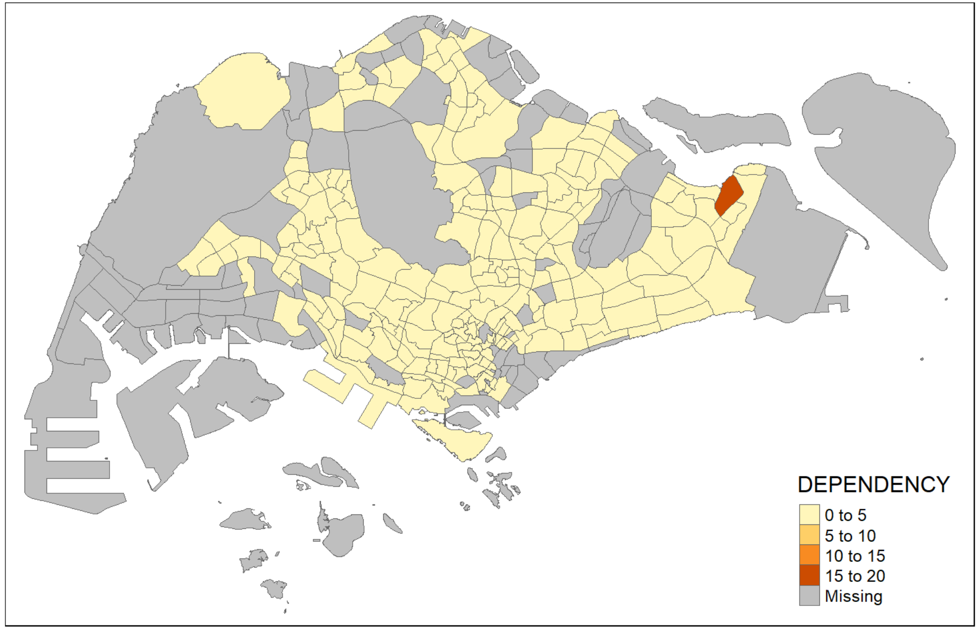
Notice that the planning subzones are shared according to the respective dependecy values



Drawing a choropleth map using *tm_border()**

To add the boundary of the planning subzones, *tm_border*s will be used as shown in the code chunk below.

```
tm_shape(mpsz_popdata2020)+
  tm_fill("DEPENDENCY") +
  tm_borders(lwd = 0.1,
             alpha = 1)
```



Notice that light-gray border lines have been added on the choropleth map.

- *lwd* = border line width. The default is 1,
- *alpha* = transparency number between 0 (totally transparent) and 1 (not transparent). By default, the alpha value of the col is used (normally 1),
- *col* = border colour, and
- *lty* = border line type. The default is "solid".

Data classification methods of tmap

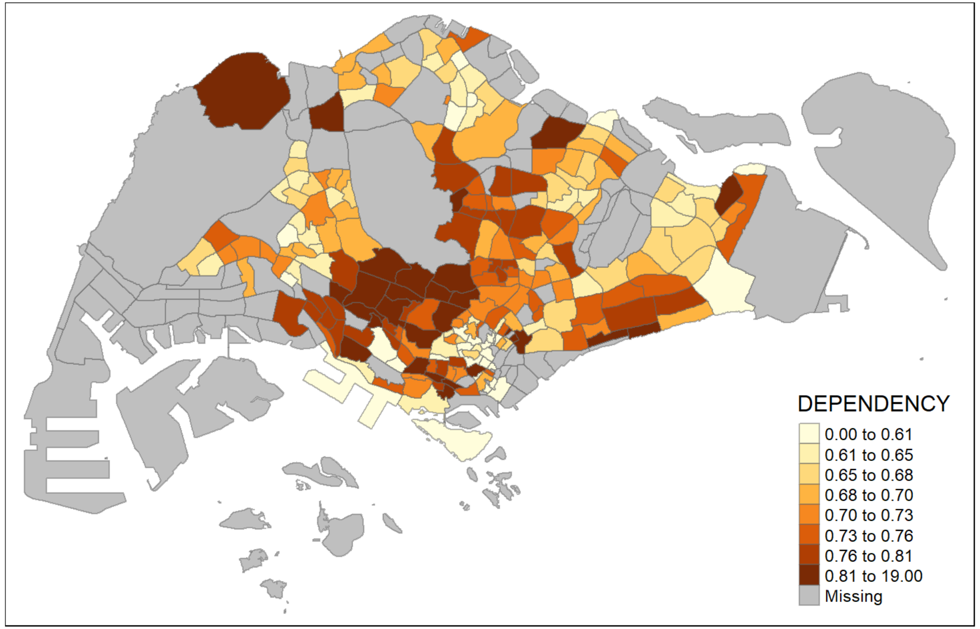
Most choropleth maps employ some method of data classification. The point of classification is to take a large number of observations and group them into data ranges or classes.

tmap provides a total ten data classification methods, namely: *fixed*, *sd*, *equal*, *pretty* (default), *quantile*, *kmeans*, *hclust*, *bclust*, *fisher*, and *jenks*.

To define a data classification method, the *style* argument of *tm_fill()* or *tm_polygons()* will be used.

The code chunk below shows a quantile data classification with 8 classes are used.

```
tm_shape(mpsz_popdata2020)+
  tm_fill("DEPENDENCY",
          n = 8,
          style = "quantile") +
  tm_borders(alpha = 0.5)
```



Comparing Quantile and Equal Interval

In the code chunk below, *quantile* and *equal* data classification methods are used.

Notice that the distribution of quantile data classification method are more evenly distributed then equal data classification method.

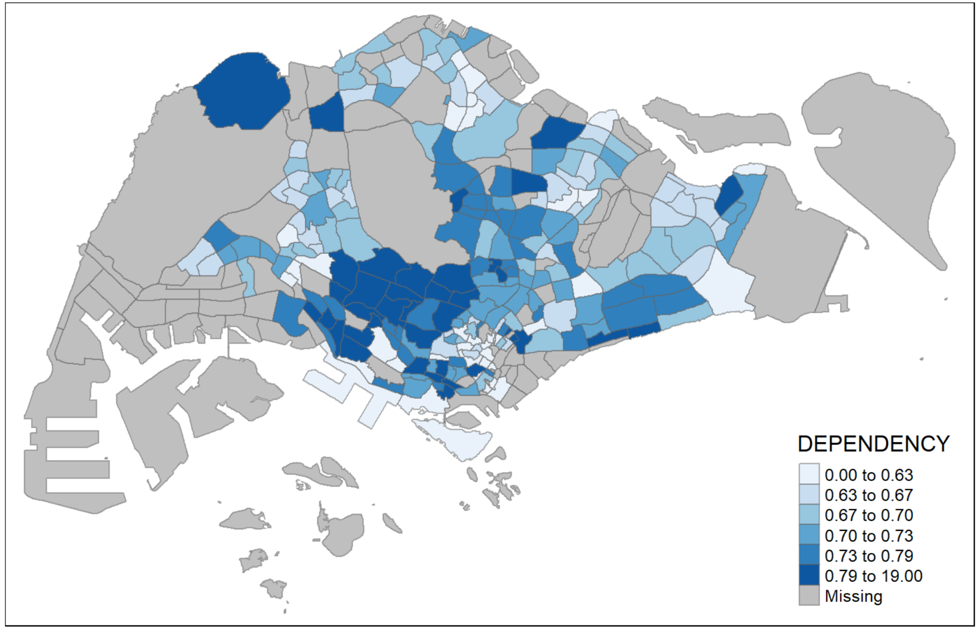
Colour Scheme

tmap supports colour ramps either defined by the user or a set of predefined colour ramps from the **RColorBrewer** package.

To change the colour, we assign the prefered colour to *palette* argument of *tm_fill()* as shown in the code chunk below.

```
tm_shape(mpsz_popdata2020)+
  tm_fill("DEPENDENCY",
          n = 6,
          style = "quantile",
          palette = "Blues") +
  tm_borders(alpha = 0.5)
```

Notice that the choropleth map is shaded in blue.

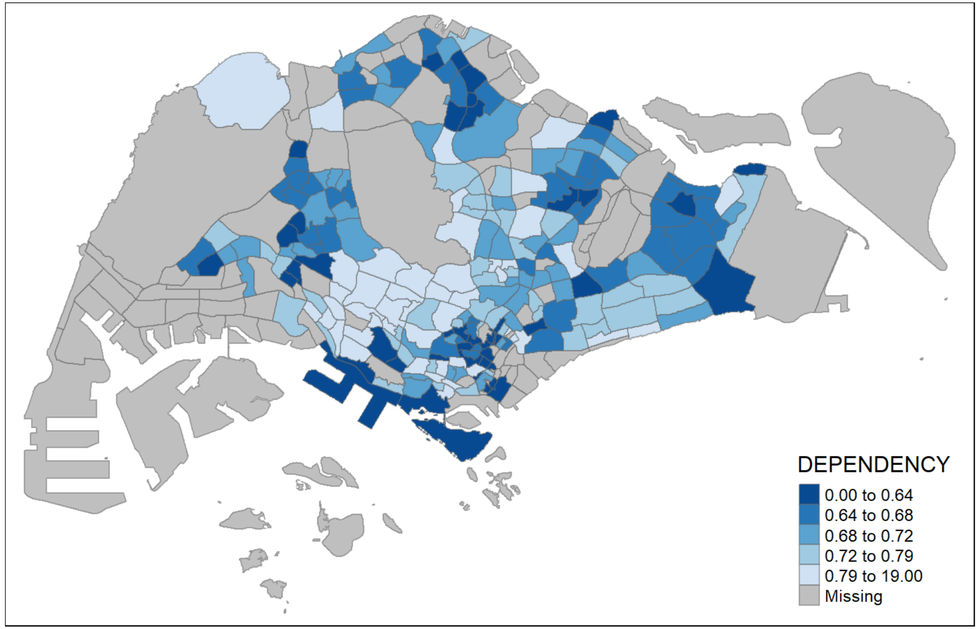


More about colour

To reverse the colour shading, add a "-" prefix.

```
tm_shape(mpsz_popdata2020)+
  tm_fill("DEPENDENCY",
          style = "quantile",
          palette = "-Blues") +
  tm_borders(alpha = 0.5)
```

Notice that the colour scheme has been reversed.



Map Layouts

Map layout refers to the combination of all map elements into a cohensive map. Map elements include among others the objects to be mapped, the title, the scale bar, the compass, margins and aspects ratios, while the colour settings and data classification methods covered in the previous section relate to the palette and break-points used to affect how the map looks.

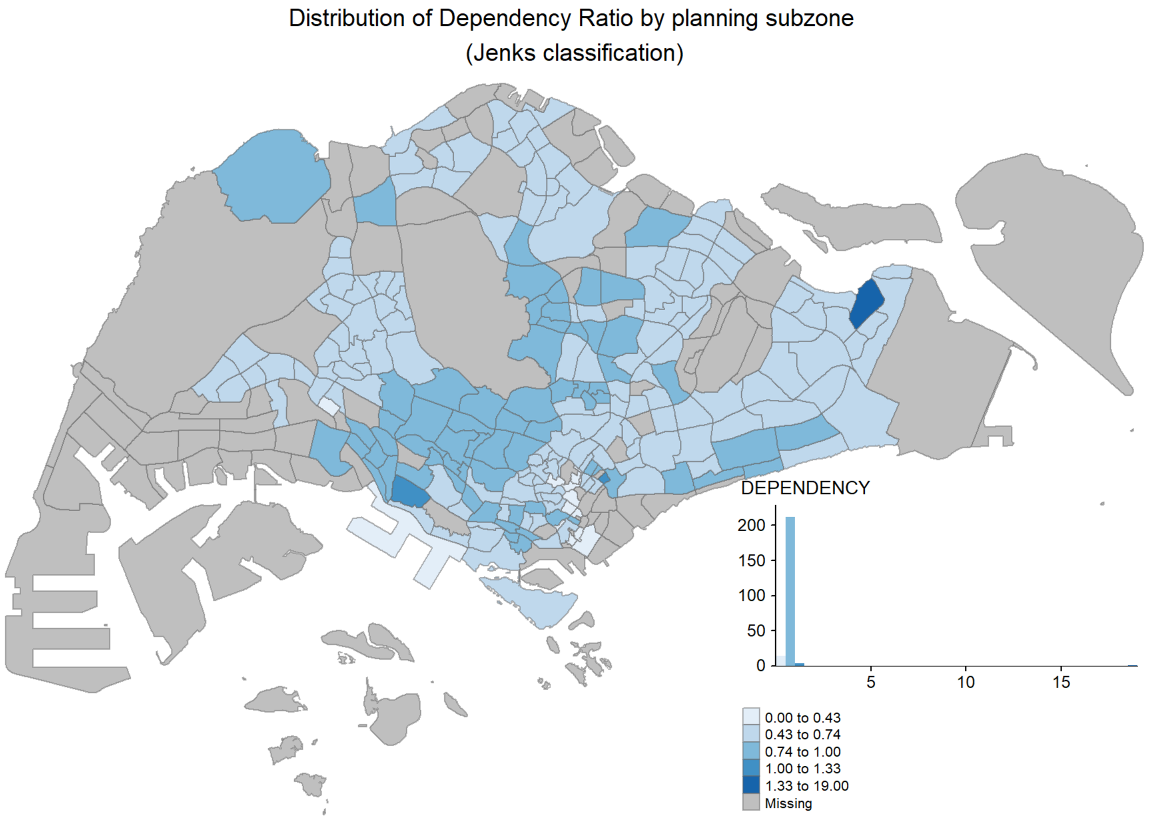
Map Legend

In **tmap**, several *legend* options are provided to change the placement, format and appearance of the legend.

```
tm_shape(mpsz_popdata2020)+
  tm_fill("DEPENDENCY",
          style = "jenks",
          palette = "Blues",
          legend.hist = TRUE,
          legend.is.portrait = TRUE,
          legend.hist.z = 0.1) +
  tm_layout(main.title = "Distribution of Dependency Ratio by planning subzone \n(Jenks classificat
            main.title.position = "center",
            main.title.size = 1,
            legend.height = 0.45,
            legend.width = 0.35,
            legend.outside = FALSE,
            legend.position = c("right", "bottom"),
            frame = FALSE) +
  tm_borders(alpha = 0.5)
```

Map Legend

The output map



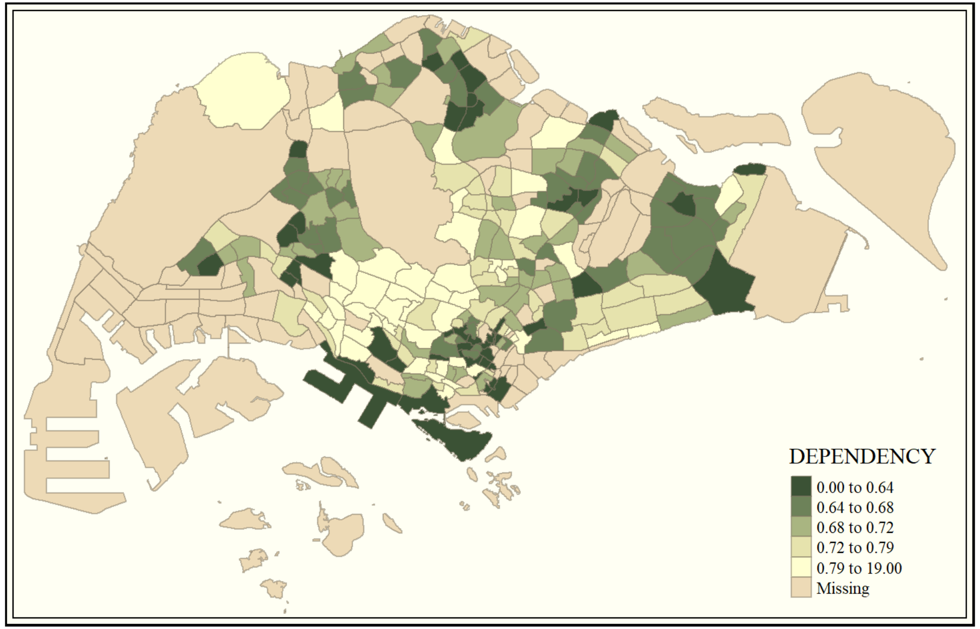
```
## [1] "#E3EEF8" "#BFD8EC" "#7FB9DA" "#4090C5" "#1664AB"
```

Map style

tmap allows a wide variety of layout settings to be changed. They can be called by using *tmap_style()*.

The code chunk below shows the *classic* style is used.

```
tm_shape(mpsz_popdata2020)+
  tm_fill("DEPENDENCY",
          style = "quantile",
          palette = "-Greens") +
  tm_borders(alpha = 0.5) +
  tmap_style("classic")
```



Cartographic Furniture

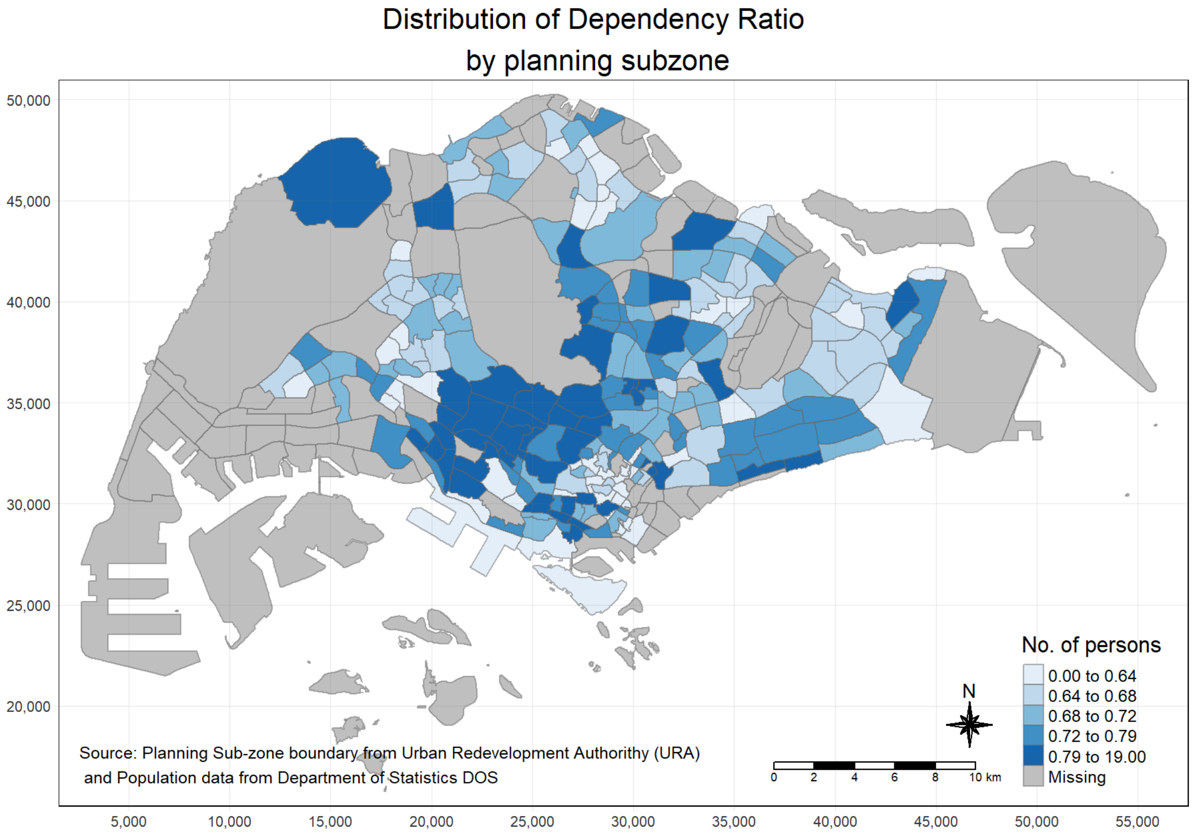
Beside map style, **tmap** also also provides arguments to draw other map furniture such as compass, scale bar and grid lines.

In the code chunk below, *tm_compass()*, *tm_scale_bar()* and *tm_grid()* are used to add compass, scale bar and grid lines onto the choropleth map.

```
tm_shape(mpsz_popdata2020)+
  tm_fill("DEPENDENCY",
          style = "quantile",
          palette = "Blues",
          title = "No. of persons") +
  tm_layout(main.title = "Distribution of Dependency Ratio \nby planning subzone",
            main.title.position = "center",
            main.title.size = 1.2,
            legend.height = 0.45,
            legend.width = 0.35,
            frame = TRUE) +
  tm_borders(alpha = 0.5) +
  tm_compass(type="8star", size = 2) +
  tm_scale_bar(width = 0.15) +
  tm_grid(lwd = 0.1, alpha = 0.2) +
  tm_credits("Source: Planning Sub-zone boundary from Urban Redevelopment Authorithy (URA)\n and Po
             position = c("left", "bottom"))
```

Cartographic Furniture

The output plot



To reset the default style, the code chunk use the code chunk below.

Drawing Small Multiple Choropleth Maps

Small multiple maps, also refered to facet maps, are composed of many maps arrange side-by-side, and sometimes stacked vertically. Small multiple maps enable the visualisation of how spatial relationships change with respect to another variable, such as time.

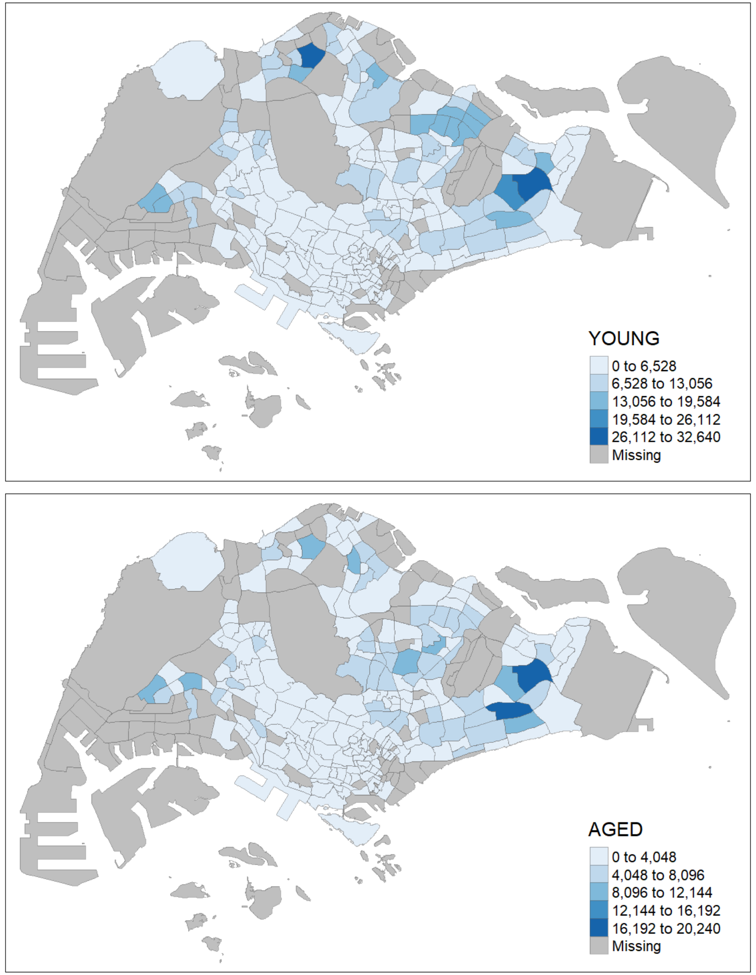
In **tmap**, small multiple maps can be plotted in three ways:

- by assigning multiple values to at least one of the asthetic arguments,
- by defining a group-by variable in *tm_facets()*, and
- by creating multiple stand-alone maps with *tmap_arrange()*.

By assigning multiple values to at least one of the aesthetic arguments

In this example, small multiple choropleth maps are created by defining ***ncols*** in **tm_fill()**

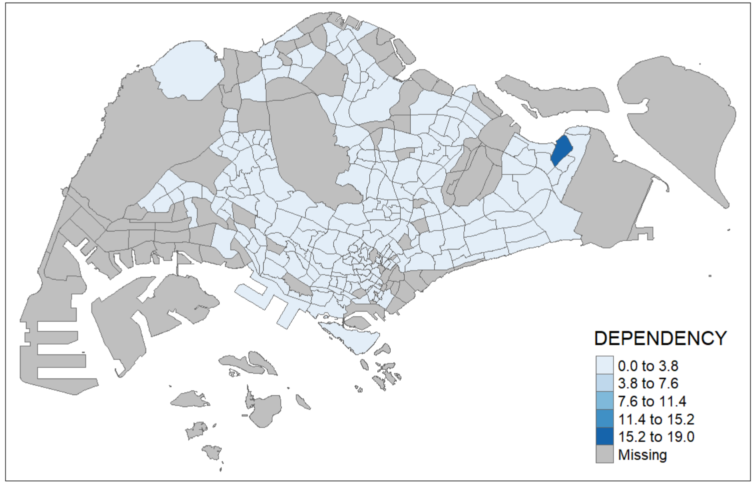
```
tm_shape(mpsz_popdata2020)+
  tm_fill(c("YOUNG", "AGED"),
          style = "equal",
          palette = "Blues") +
  tm_layout(legend.position = c("right",
                                "bottom")) +
  tm_borders(alpha = 0.5) +
  tmap_style("white")
```

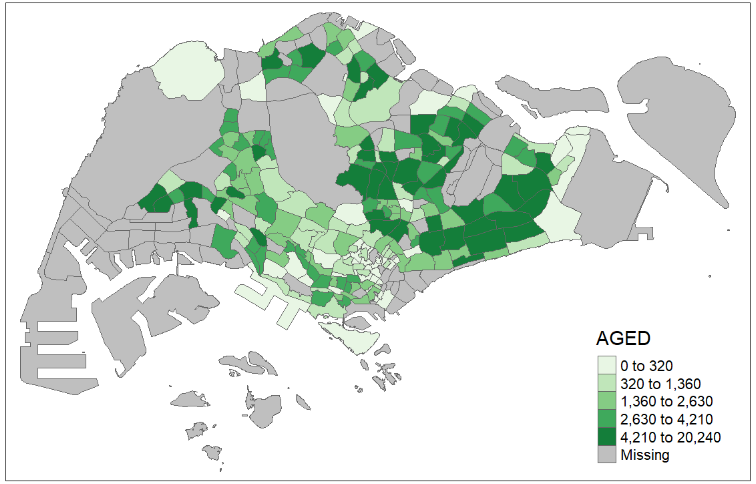


By assigning multiple values to at least one of the aesthetic arguments

In this example, small multiple choropleth maps are created by assigning multiple values to at least one of the aesthetic arguments

```
tm_shape(mpsz_popdata2020)+
  tm_polygons(c("DEPENDENCY","AGED"),
          style = c("equal", "quantile"),
          palette = list("Blues","Greens")) +
  tm_layout(legend.position = c("right",
                                "bottom"))
```

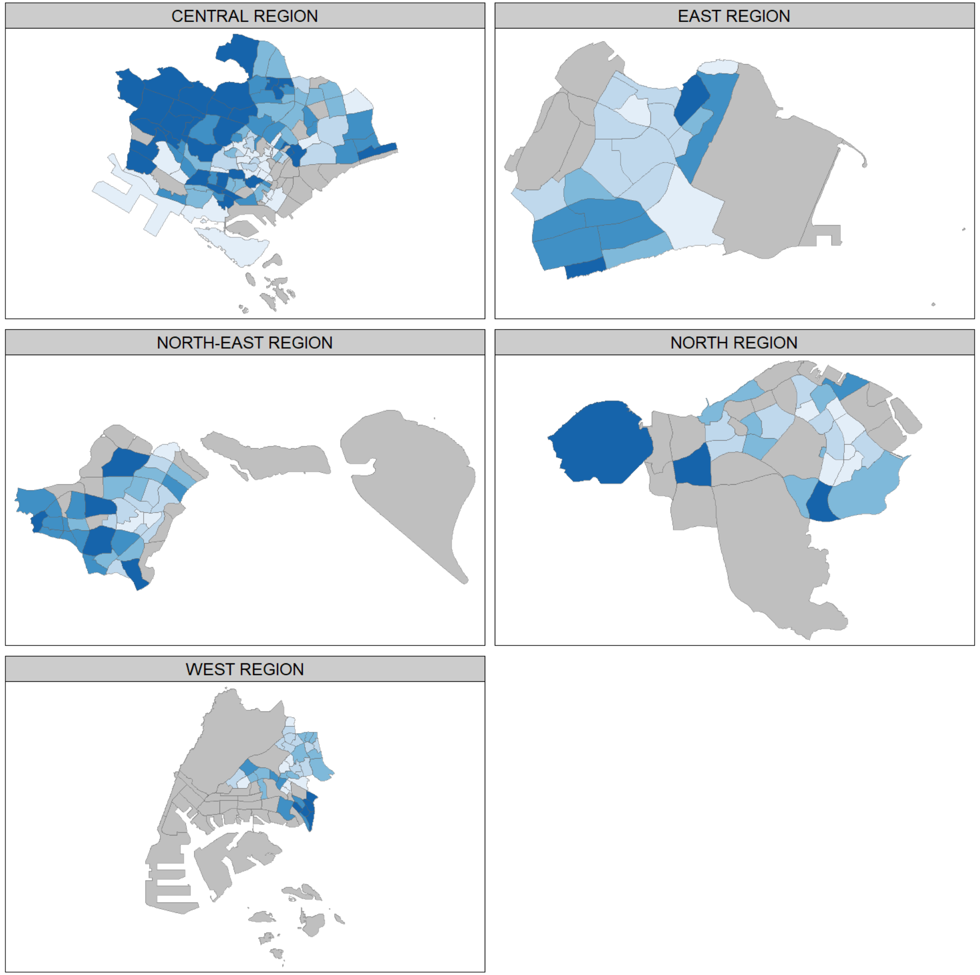




By defining a group-by variable in *tm_facets()*

In this example, multiple small choropleth maps are created by using **tm_facets()**.

```
tm_shape(mpsz_popdata2020) +
  tm_fill("DEPENDENCY",
          style = "quantile",
          palette = "Blues",
          thres.poly = 0) +
  tm_facets(by="REGION_N",
            free.coords=TRUE,
            drop.shapes=TRUE) +
  tm_layout(legend.show = FALSE,
            title.position = c("center",
                               "center"),
            title.size = 20) +
  tm_borders(alpha = 0.5)
```



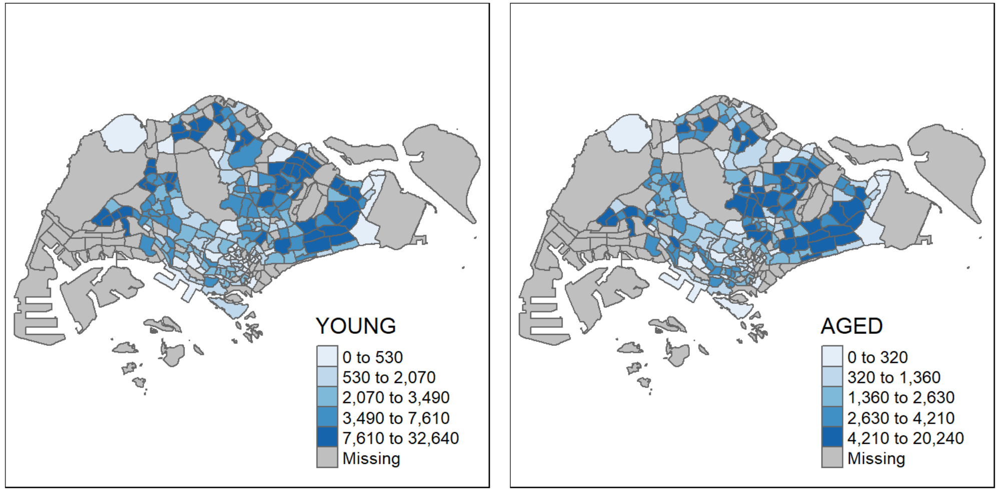
By creating multiple stand-alone maps with *tmap_arrange()*

In this example, multiple small choropleth maps are created by creating multiple stand-alone maps with **tmap_arrange()**.

```
youngmap <- tm_shape(mpsz_popdata2020)+
  tm_polygons("YOUNG",
              style = "quantile",
              palette = "Blues")

agedmap <- tm_shape(mpsz_popdata2020)+
  tm_polygons("AGED",
              style = "quantile",
              palette = "Blues")

tmap_arrange(youngmap, agedmap, asp=1, ncol=2)
```



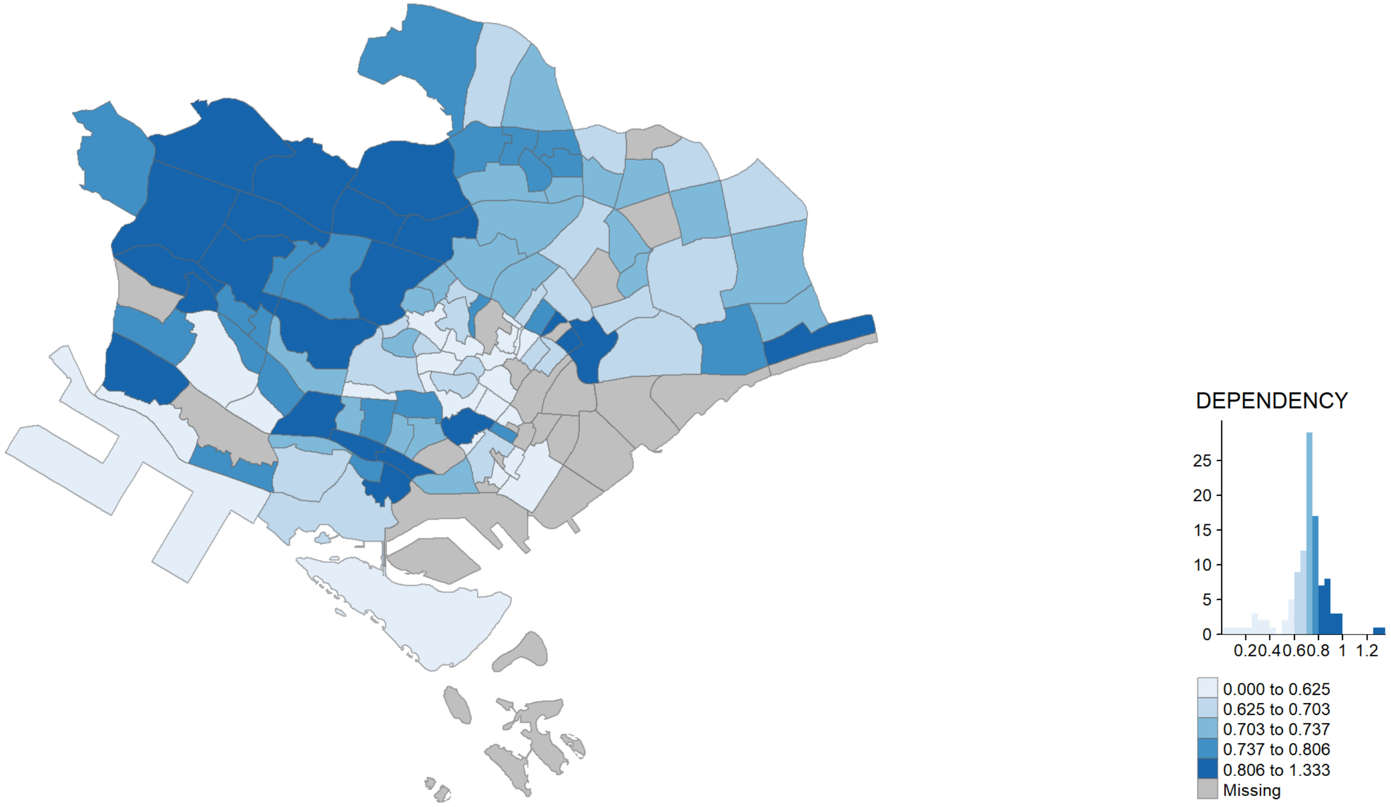
Mappping Spatial Object Meeting a Selection Criterion

Instead of creating small multiple choropleth map, you can also use selection function to map spatial objects meeting the selection criterion.

```
tm_shape(mpsz_popdata2020[mpsz_popdata2020$REGION_N=="CENTRAL REGION", ]) +
  tm_fill("DEPENDENCY",
          style = "quantile",
          palette = "Blues",
          legend.hist = TRUE,
          legend.is.portrait = TRUE,
          legend.hist.z = 0.1) +
  tm_layout(legend.outside = TRUE,
            legend.height = 0.45,
            legend.width = 5.0,
            legend.position = c("right", "bottom"),
            frame = FALSE) +
  tm_borders(alpha = 0.5)
```

Mappping Spatial Object Meeting a Selection Criterion

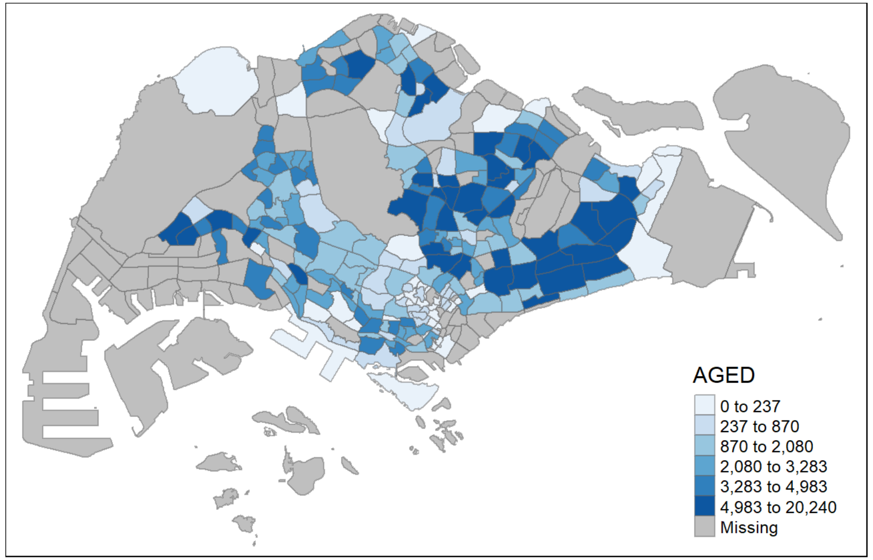
The output choropleth maps.

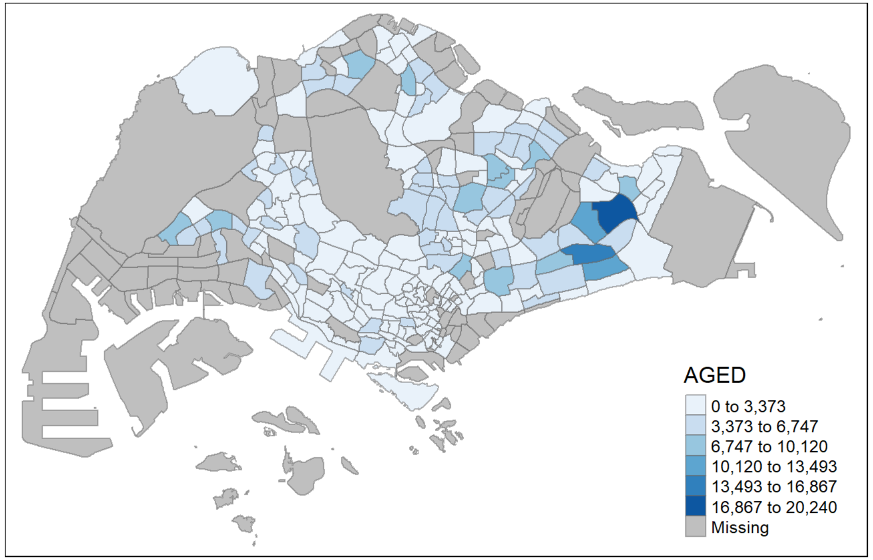


```
## [1] "#E3EEF8" "#BFD8EC" "#7FB9DA" "#4090C5" "#1664AB"
```

Limitation of Statistical Map: Maps lie!

Although both choropleth maps were created using the same variable (i.e. aged population) but the choropleth maps produced look very different. This is because the choropleth maps were created using two different data classification methods. For the choropleth map on the left, quantile classification method was used and for choropleth on the right, equal interval classification method was used.

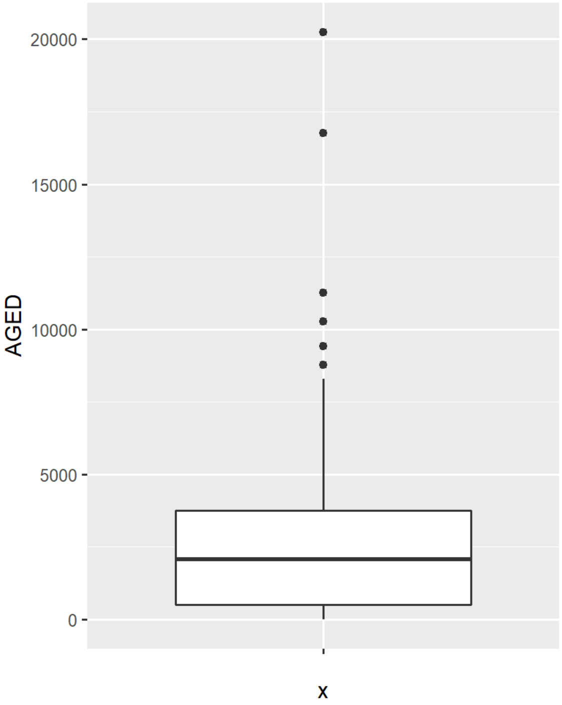




> **Challenge:**
>
> - How to identify planning subzones with extreme high or low numbers of aged population?

Visualising Extreme Values

- **Boxplot** is one one the popular Exploratory Data Analysis (EDA) technique used to show the statistics and distribution of data values.



The code chunk is:

```
ggplot(data=mpszpop2020,
       aes(x = "",
           y = AGED)) +
  geom_boxplot()
```

Despite its usefulness, boxplot is not able to reveal the spatial distribution of these outliers.

Extreme Value Maps

- Extreme value maps are variations of common choropleth maps where the classification is designed to highlight extreme values at the lower and upper end of the scale, with the goal of identifying outliers.

- These maps were developed in the spirit of spatializing EDA, i.e., adding spatial features to commonly used approaches in non-spatial EDA (Anselin 1994).

Box map

- Displaying summary statistics on a choropleth map by using the basic priciples of boxplot.

- In essence, a box map is an augmented quartile map, with an additional lower and upper category. When there are lower outliers, then the starting point for the breaks is the minimum value, and the second break is the lower fence. In contrast, when there are no lower outliers, then the starting point for the breaks will be the lower fence, and the second break is the minimum value (there will be no observations that fall in the interval between the lower fence and the minimum value).

- To create a box map, a custom breaks specification will be used. However, there is a complication. The break points for the box map vary depending on whether lower or upper outliers are present.

Why Writing Functions?

Writing a function has three big advantages over using copy-and-paste:

- You can give a function an evocative name that makes your code easier to understand.
- As requirements change, you only need to update code in one place, instead of many.
- You eliminate the chance of making incidental mistakes when you copy and paste (i.e. updating a variable name in one place, but not in another).

Source: Chapter 19: Functions of **R for Data Science**.

Creating the boxbreaks function

```r
boxbreaks <- function(v,mult=1.5) {
  qv <- unname(quantile(v))
  iqr <- qv[4] - qv[2]
  upfence <- qv[4] + mult * iqr
  lofence <- qv[2] - mult * iqr
  # initialize break points vector
  bb <- vector(mode="numeric",length=7)
  # logic for lower and upper fences
  if (lofence < qv[1]) {  # no lower outliers
    bb[1] <- lofence
    bb[2] <- floor(qv[1])
  } else {
    bb[2] <- lofence
    bb[1] <- qv[1]
  }
  if (upfence > qv[5]) { # no upper outliers
    bb[7] <- upfence
    bb[6] <- ceiling(qv[5])
  } else {
    bb[6] <- upfence
    bb[7] <- qv[5]
  }
  bb[3:5] <- qv[2:4]
  return(bb)
```

The code chunk on the left is an R function that creating break points for a box map.

- arguments:
  - v: vector with observations
  - mult: multiplier for IQR (default 1.5)
- returns:
  - bb: vector with 7 break points compute quartile and fences

Creating the get.var function

```r
get.var <- function(vname,df) {
  v <- df[vname] %>% st_set_geometry(NULL)
  v <- unname(v[,1])
  return(v)
}
```

The code chunk on the left is an R function to extract a variable as a vector out of an sf data frame.

- arguments:
  - vname: variable name (as character, in quotes)
  - df: name of sf data frame
- returns:
  - v: vector with values (without a column name)

Test drive the newly created function

Let's test the newly created function

```
var <- get.var("AGED", mpszpop2020)
boxbreaks(var)
```

Let's exclude AGED = NA by using the code chunk below.

```
mpszpop2020a <- mpszpop2020 %>%
  filter(AGED>=0)
var <- get.var("AGED", mpszpop2020a)
boxbreaks(var)
```

```
## [1] -4330     0   515  2080  3745  8590 20240
```

Boxmap function

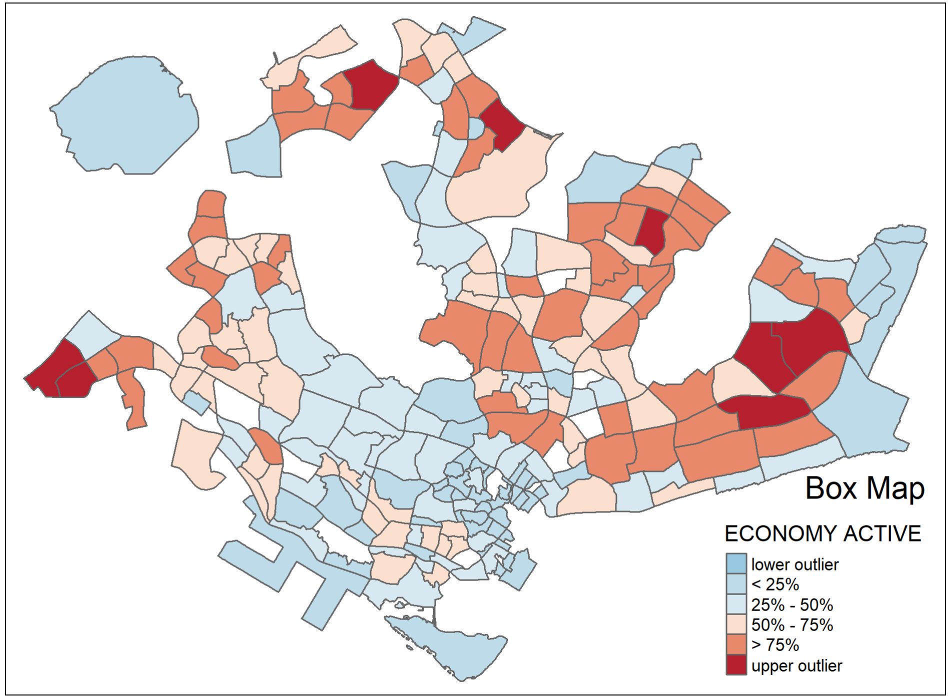
```r
boxmap <- function(vnam, df,
                   legtitle=NA,
                   mtitle="Box Map",
                   mult=1.5){
  var <- get.var(vnam,df)
  bb <- boxbreaks(var)
  tm_shape(df) +
     tm_fill(vnam,title=legtitle,
             breaks=bb,
             palette="-RdBu",
          labels = c("lower outlier",
                     "< 25%",
                     "25% - 50%",
                     "50% - 75%",
                     "> 75%",
                     "upper outlier"))  +
  tm_borders() +
  tm_layout(title = mtitle,
            title.position = c("right",
                               "bottom"))
}
```

The code chunk on the left is an R function to create a box map.

- arguments:
  - vnam: variable name (as character, in quotes)
  - df: simple features polygon layer
  - legtitle: legend title
  - mtitle: map title
  - mult: multiplier for IQR
- returns:
  - a tmap-element (plots a map)

The box map of AGED population

```
boxmap("ECONOMY ACTIVE", mpszpop2020a)
```

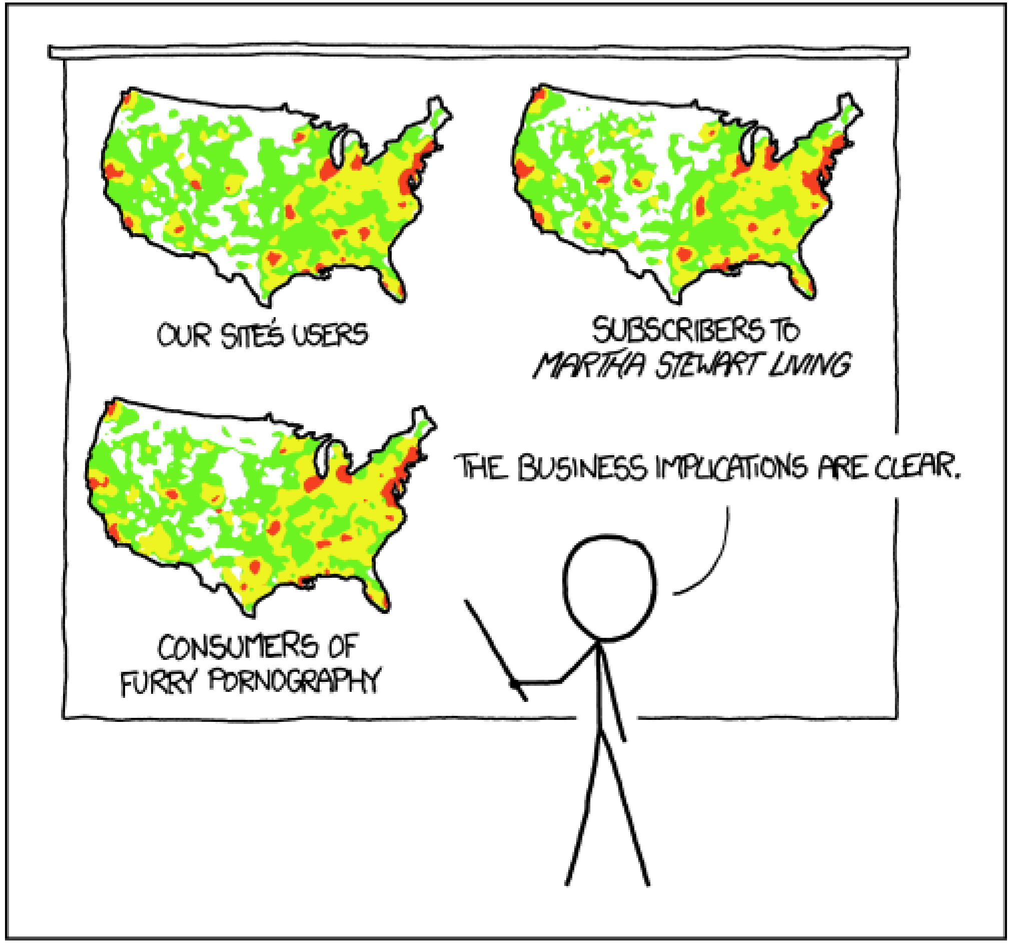


- The box map reveals that there are six upper outliers (i.e. planning subzone with extremely high numbers of aged population)
- Four of the upper outliers are located at the eastern region and they are closed to each others.
- There is no lower outlier.

But why some planning subzones were eaten by rats? Can you fix it?

Choropleth Map for Rates

In much of our readings we have now seen the importance to map rates rather than counts of things, and that is for the simple reason that population is not equally distributed in space. That means that if we do not account for how many people are somewhere, we end up mapping population size rather than our topic of interest.


OUR SITE'S USERS
SUBSCRIBERS TO MARTHA STEWART LIVING
CONSUMERS OF FURRY PORNOGRAPHY
THE BUSINESS IMPLICATIONS ARE CLEAR.

PET PEEVE #208:
GEOGRAPHIC PROFILE MAPS WHICH ARE BASICALLY JUST POPULATION MAPS

Raw rate map

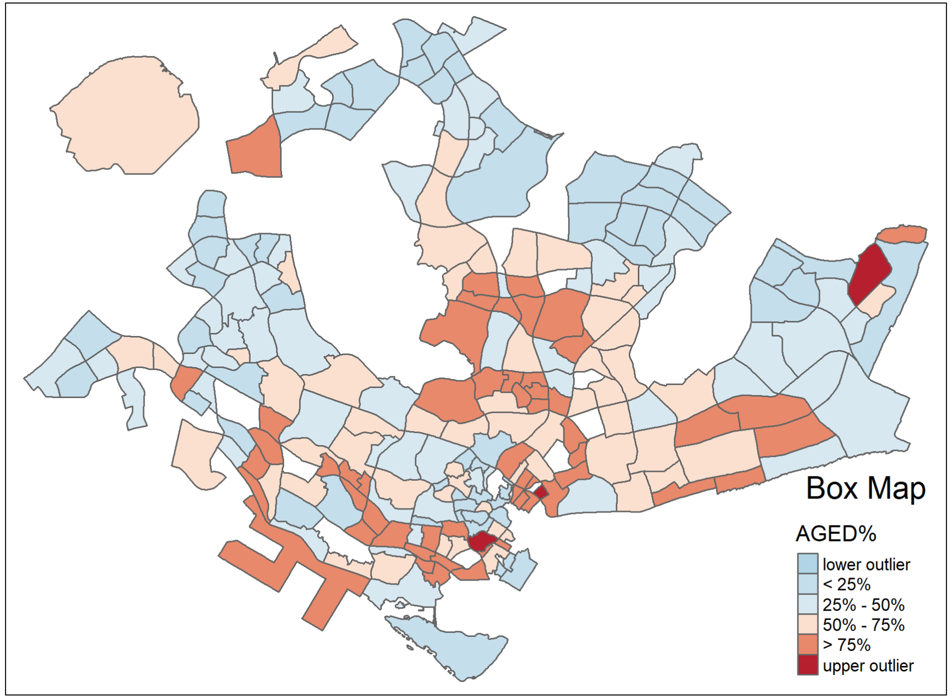
- First, compute the raw rate by using the code chunk below:

```
mpszpop2020a <- mpszpop2020 %>%
  mutate(`AGED%` = (`AGED`
/`TOTAL`)*100) %>%
  filter(`AGED%` >= 0)
```

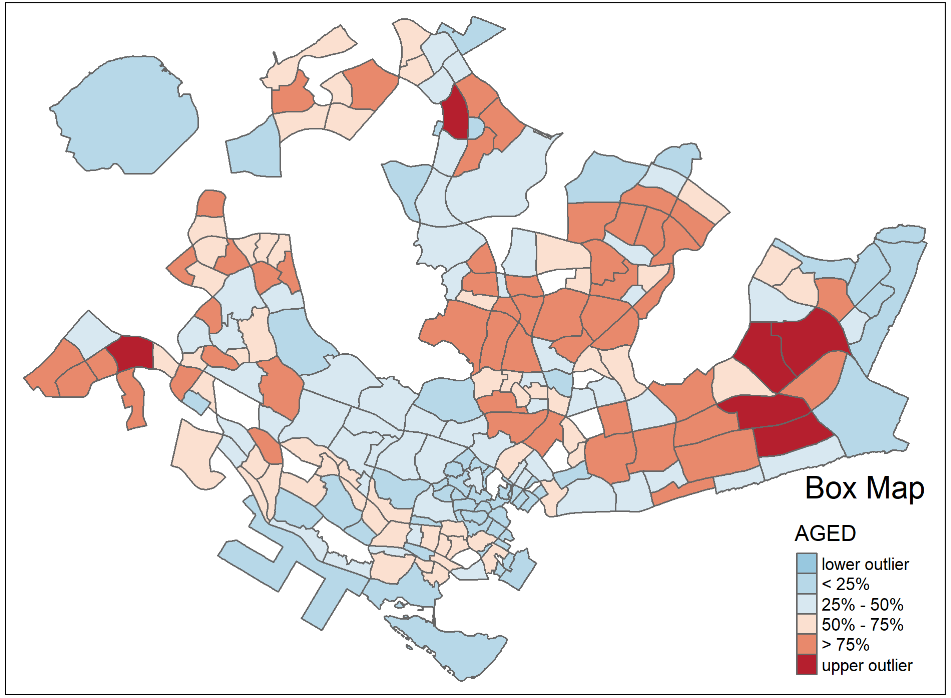
Next, the boxmap function will be used to plot the raw rate map as shown in the code chunk below.

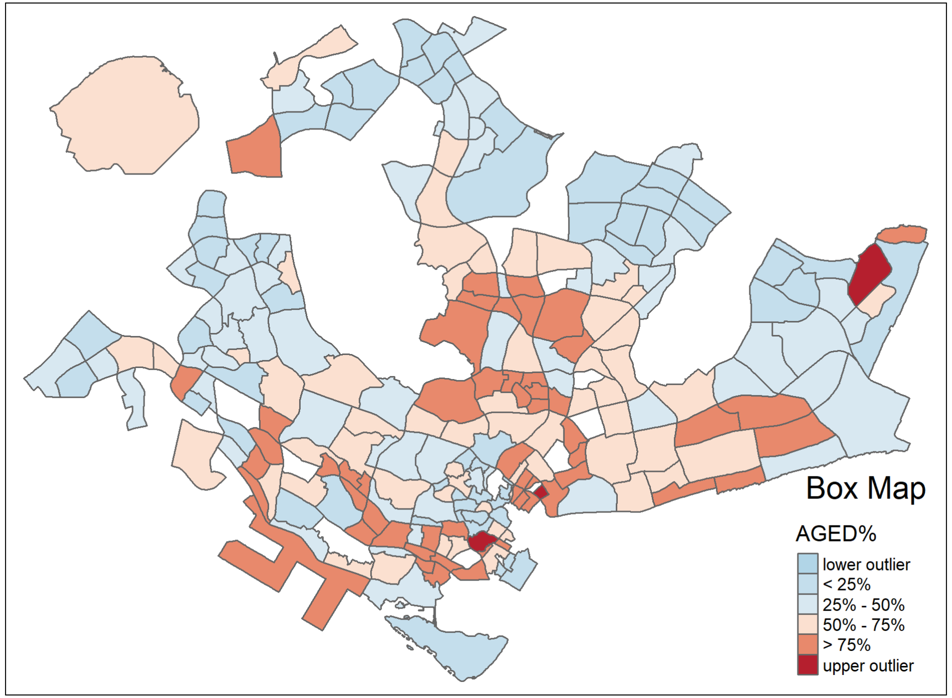
```
var <- get.var("AGED%", mpszpop2020a)
boxbreaks(var)
boxmap("AGED%",mpszpop2020a)
```

```
## [1] -2.17276  0.00000 11.28169 16.48199 20.25132 3
```



Comparing Absolute and Rate Choropleth Maps


Box Map
AGED
lower outlier
< 25%
25% - 50%
50% - 75%
> 75%
upper outlier


Box Map
AGED%
lower outlier
< 25%
25% - 50%
50% - 75%
> 75%
upper outlier