Hands-on Exercise 7: Handling and Visualising Geospatial Data with R

Dr. Kam Tin Seong
Assoc. Professor of Information Systems

School of Computing and Information Systems,
Singapore Management University

2020-2-15 (updated: 2022-05-21)

Content

- Mapping Geospatial Point Data with R
- Choropleth Mapping with R

Ex 1:Mapping Geospatial Point Data with R

In this hands-on exercise, you will learn how to create proportional symbol map 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 convert it into simple point feature by using **sf** package, and
  - to create interactive proportional symbol maps by using **tmap** package.

Getting Started

Write a code chunk to check, install and launch **readr**, **sf** and **tmap** packages of R

The solution:

Importing the data

- Next, you are required to write a code chunk to import *SGPools_svy21.csv* into R by using `read_csv()` of **readr** package. You can called the tibble data frame object *sgpools*.

The solution:

- After importing the data file into R, it is important for us to review the data object.

Creating a sf data frame

The code chunk on the right converts *sgpools* data frame into a simple feature data frame by using *st_as_sf()* of **sf** packages

Things to learn from the arguments:

- The *coords* argument requires you to provide the column name of the x-coordinates first then followed by the column name of the y-coordinates.
- The *crs* argument required you to provide the coordinates system in epsg format. EPSG: 3414 is Singapore SVY21 Projected Coordinate System. You can search for other country's epsg code by refering to epsg.io.

```
sgpools_sf <- st_as_sf(
  sgpools,
  coords = c("XCOORD",
             "YCOORD"),
             crs= 3414)
```

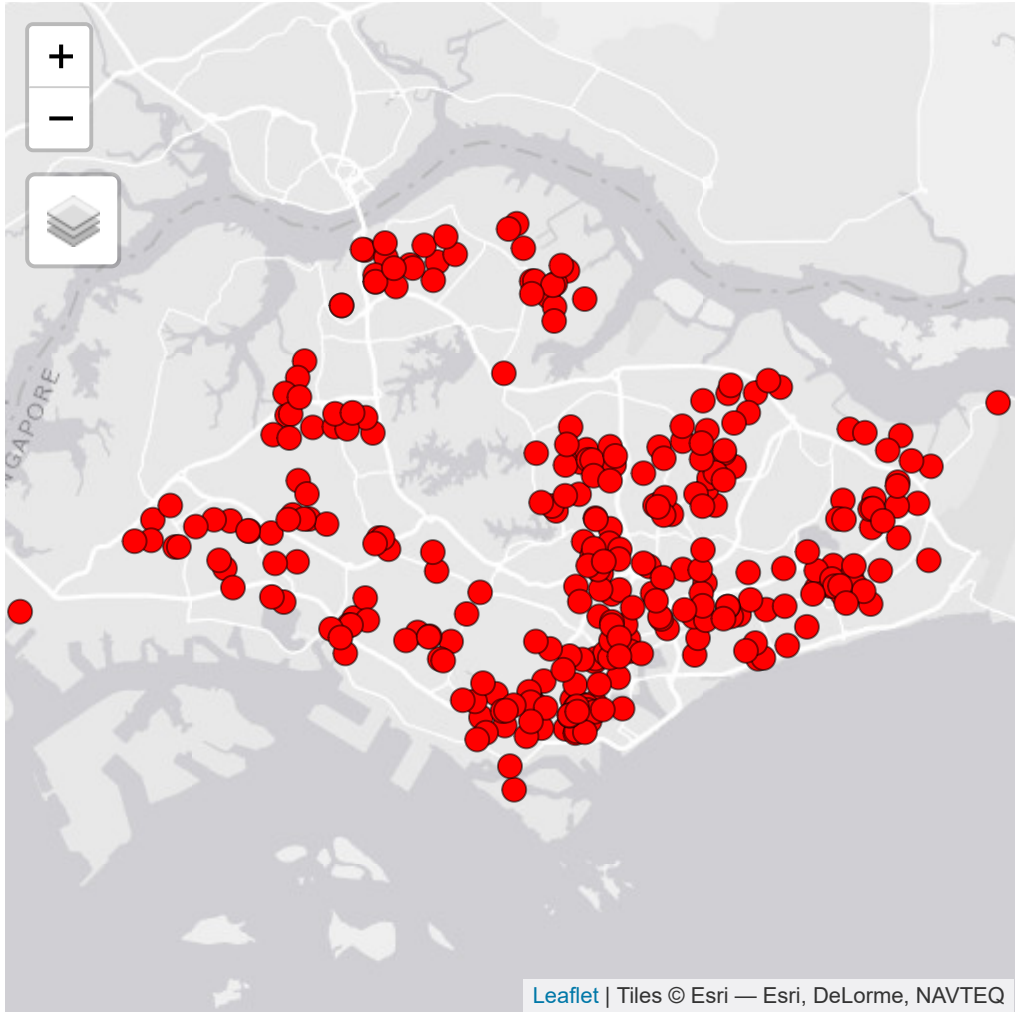
Ploting a point symbol map

The code chunk below is used to create an interactive point symbol map.

```
tmap_mode("view")
tm_shape(sgpools_sf)+
tm_bubbles(col = "red",
           size = 1,
           border.col = "black",
           border.lwd = 1)
```

Things to learn from the code chunk:

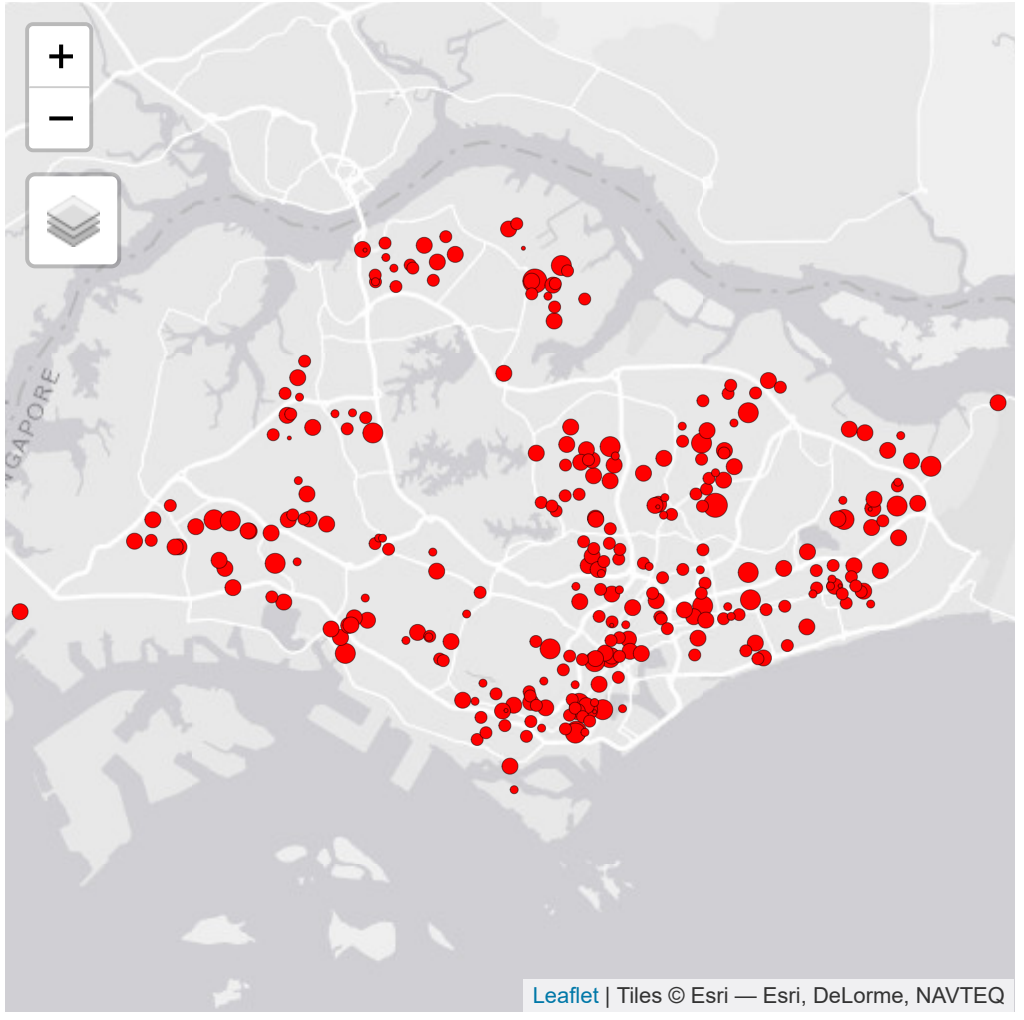
- `tmap_mode()` is used to switch the display from static mode (i.e. "plot") to interactive mode (i.e. "view").
- `tm_shape()` is used to create a **tmap-element** that specifies a spatial data object (i.e. point).
- `tm_bubble()` is used to create a **tmap-element** that draws bubbles or small dots. Both colors and sizes of the bubbles can be mapped to data variables.



Lets make it proportional

To draw a proportional symbol map, we need to assign a numerical variable to the size visual attribute. The code chunks below show that the variable *Gp1Gp2Winnings* is assigned to size visual attribute.

```
tm_shape(sgpools_sf)+
tm_bubbles(col = "red",
           size = "Gp1Gp2 Winnings",
           border.col = "black",
           border.lwd = 0.5)
```

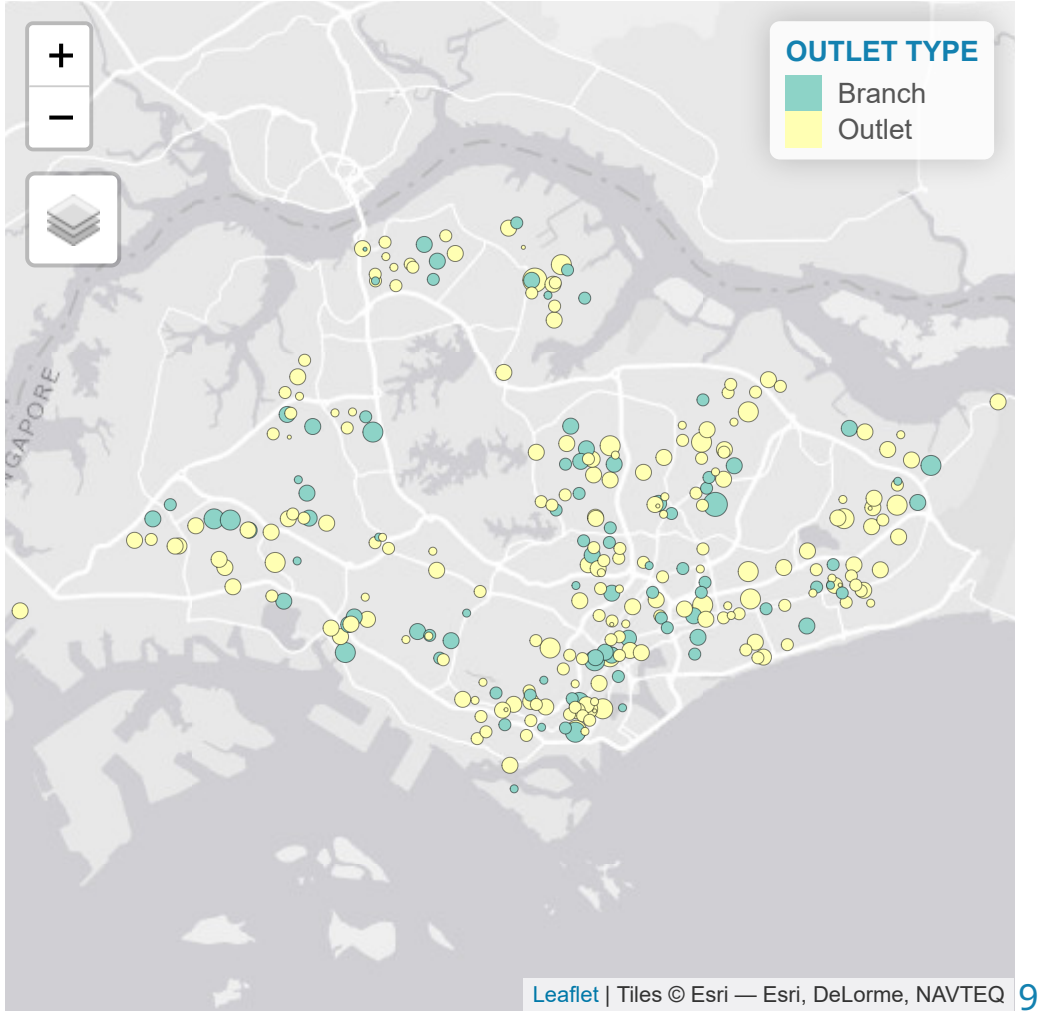


Lets give it a different colour

The proportional symbol map can be further improved by using the colour visual attribute. In the code chunks below, *OUTLET_TYPE* variable is used as the colour attribute variable.

```
tm_shape(sgpools_sf)+
tm_bubbles(col = "OUTLET TYPE",
           size = "Gp1Gp2 Winnings",
           border.col = "black",
           border.lwd = 0.5)
```

The solution:



I have a twin brothers :)

An impressive and little-know feature of **tmap**'s view mode is that it also works with faceted plots. The argument *sync* in *tm_facets()* can be used in this case to produce multiple maps with synchronised zoom and pan settings.

```
tm_shape(sgpools_sf) +
  tm_bubbles(col = "OUTLET TYPE",
          size = "Gp1Gp2 Winnings",
          border.col = "black",
          border.lwd = 1) +
  tm_facets(by= "OUTLET TYPE",
            nrow = 1,
            sync = TRUE)
```

Ex 2: Choropleth Mapping with R

- 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.

The Data

Two data set will be used to create the choropleth map, they are:

- URA Master Plan subzone boundary in shapefile format (i.e. *MP14_SUBZONE_WEB_PL*). This is a geospatial data. It consists of the geographical boundary of Singapore at the planning subzone level. The data is based on URA Master Plan 2014.

- Singapore Residents by Planning Area/Subzone, Age Group and Sex, June 2000 - 2018 in csv format (i.e. *respopagsex2000to2018.csv*). This is an aspatial data fie. Although it does not contain any coordinates values, but it's PA and SZ fields can be used as unique identifiers to georeference to *MP14_SUBZONE_WEB_PL* shapefile.

Importing geospatial data into R

The code chunk below uses the `st_read()` function of *sf* package to import *MP14_SUBZONE_WEB_PL* shapfile into R as a simple feature data frame called *mpsz*.

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

```
## Reading layer `MP14_SUBZONE_WEB_PL' from data source
##   `D:\tskam\ISSS608\Hands-on_Ex\Hands-on_Ex07\data\geospatial'
##   using driver `ESRI Shapefile'
## Simple feature collection with 323 features and 15 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 2667.538 ymin: 15748.72 xmax: 56396.44 ymax: 50256.33
## Projected CRS: SVY21
```

Importing attribute data into R

Next, we will import *respopagsex2000to2018.csv* file into RStudio and save the file into an R dataframe called *popagsex*.

The task will be performed by using `read_csv()` function of **readr** package as shown in the code chunk below.

```
popagsex <- read_csv("data/aspatial/respopagsex2000to2018.csv")
```

Data Preparation

Before a thematic map can be prepared, you need to preform the following data preparation.

- Extracting 2018 records only.
- Extracting Males records only.
- Deriving three new variables, namely: Young, Economic Active and Aged.

The following data wrangling and transformation functions will be used:

- `spread()` of **tidyr** package, and
- `mutate()`, `filter()`, and `select()` of **dplyr** package

The code chunk:

```
popagsex2018_male <- popagsex %>%
  filter(Sex == "Males") %>%
  filter(Time == 2018) %>%
  spread(AG, Pop) %>%
  mutate(YOUNG = `0_to_4`+`5_to_9`+`10_to_14`+
`15_to_19`+`20_to_24`) %>%
mutate(`ECONOMY ACTIVE` = rowSums(.[9:13])+
rowSums(.[15:17]))%>%
mutate(`AGED`=rowSums(.[18:22])) %>%
mutate(`TOTAL`=rowSums(.[5:22])) %>%
mutate(`DEPENDENCY` = (`YOUNG` + `AGED`)
/`ECONOMY ACTIVE`) %>%
mutate_at(.vars = vars(PA, SZ),
          .funs = funs(toupper)) %>%
  select(`PA`, `SZ`, `YOUNG`,
         `ECONOMY ACTIVE`, `AGED`,
         `TOTAL`, `DEPENDENCY`) %>%
  filter(`ECONOMY ACTIVE` > 0)
```

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.

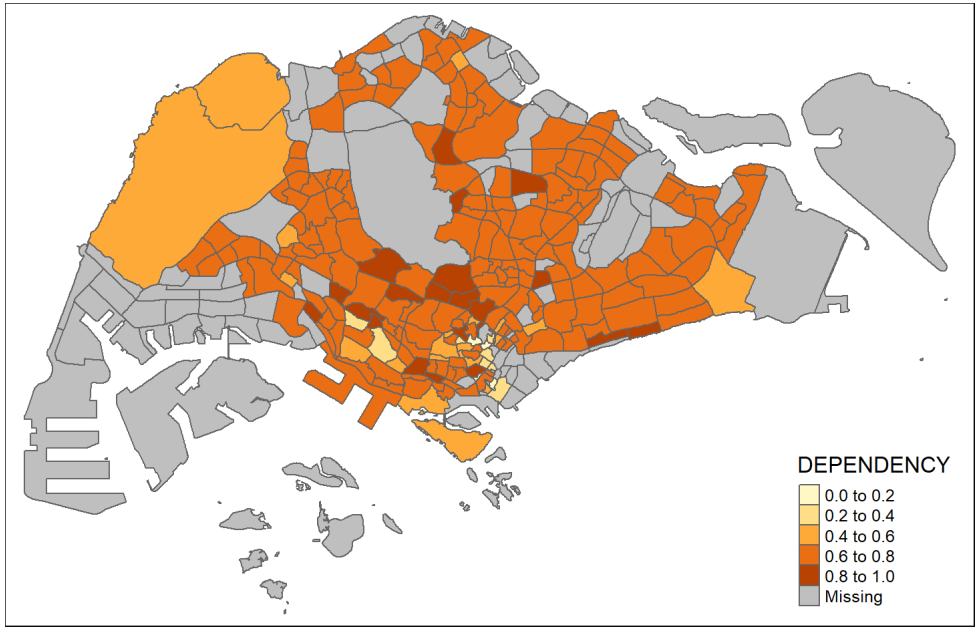
```
mpsz_agemale2018 <- left_join(mpsz,
                              popagsex2018_male,
                              by = c("SUBZONE_N" = "SZ"))
```

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_agemale2018,
    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_shape()` is used to define the input data (i.e *mpsz_agmale2018*) and `tm_polygons()` is used draw the planning subzone polygons

```
tm_shape(mpsz_agemale2018) +
  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.

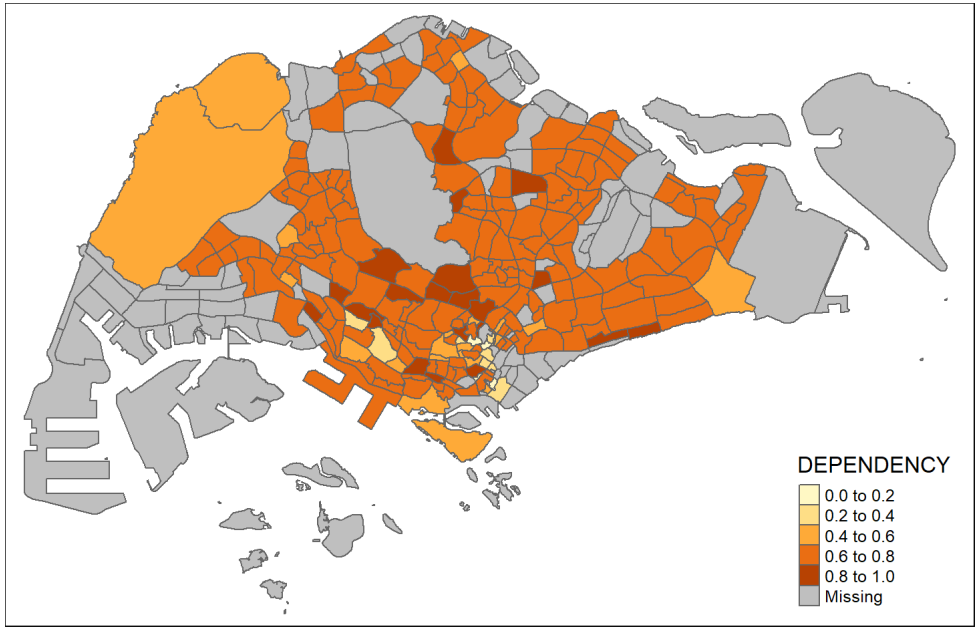
Drawing a choropleth map using `tm_polygons()`

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_agemale2018)+
  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.



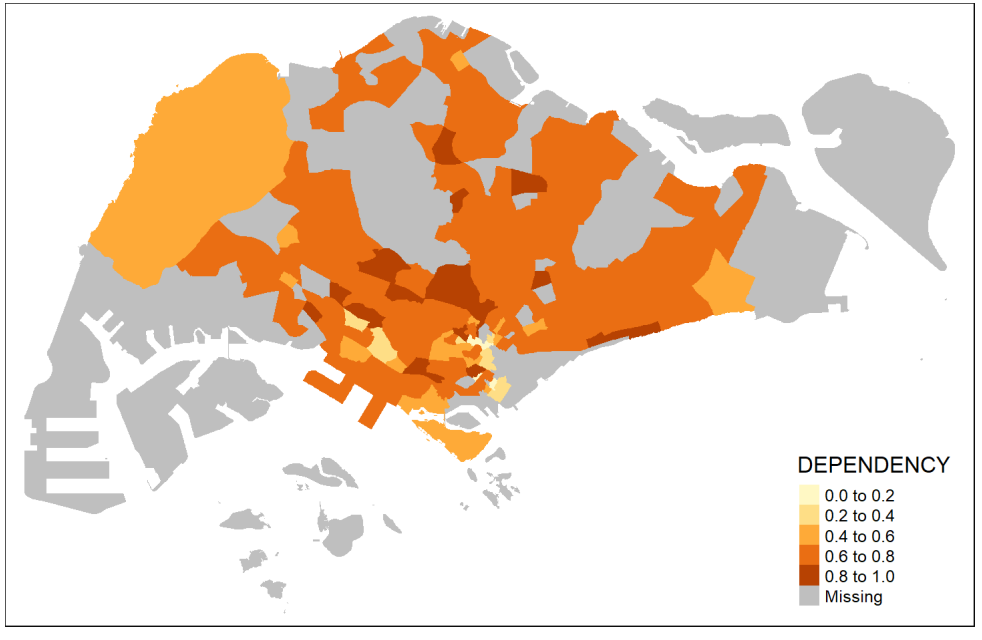
Drawing a choropleth map using `tm_fill()` and `tm_border()`

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_agemale2018)+
  tm_fill("DEPENDENCY")
```

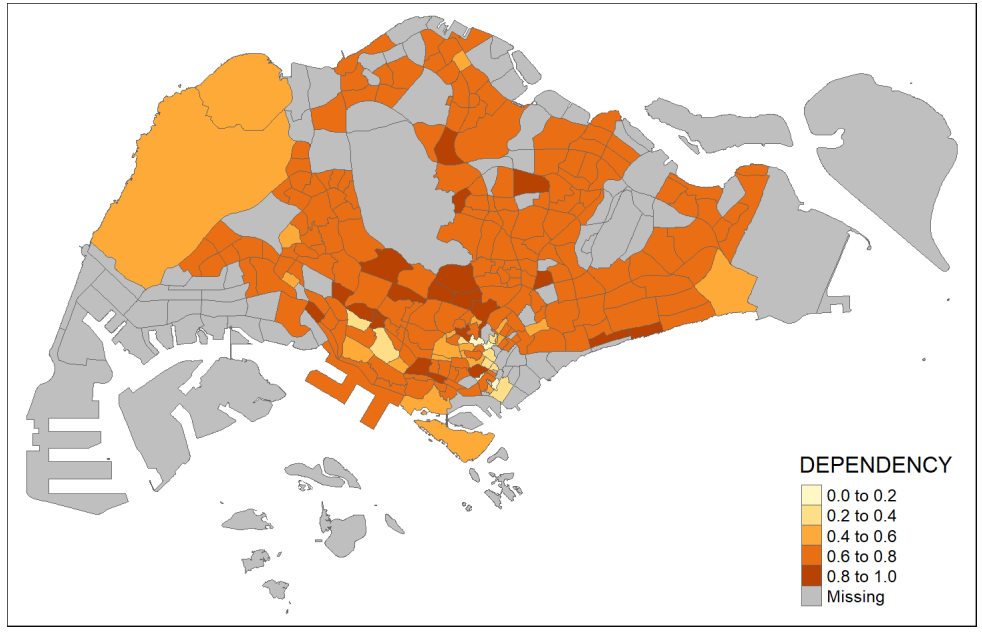
Notice that the planning subzones are shared according to the respective dependecy values



Drawing a choropleth map using `tm_border()`

To add the boundary of the planning subzones, `tm_border()` will be used as shown in the code chunk below.

```
tm_shape(mpsz_agemale2018)+
  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

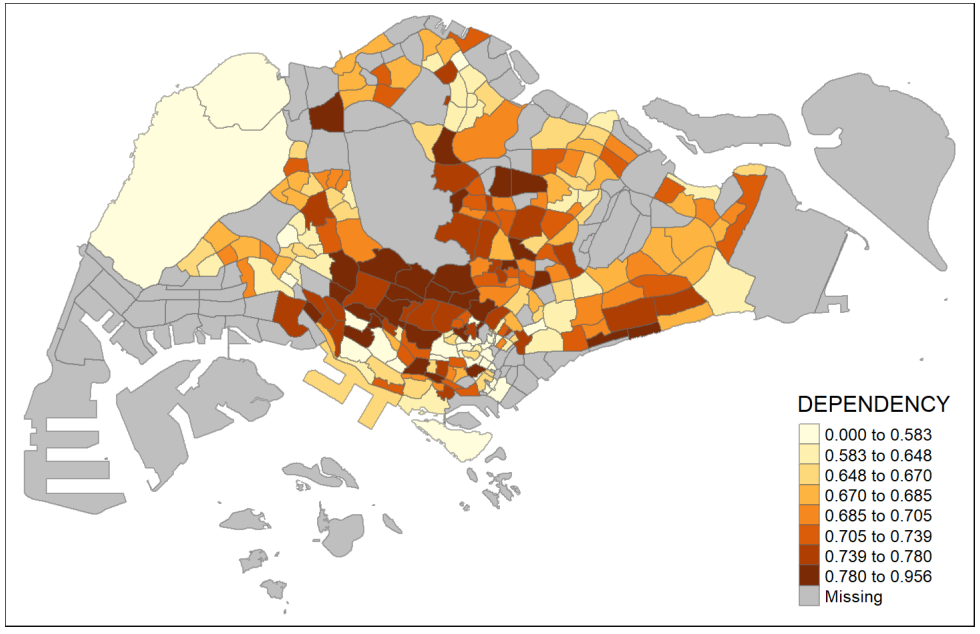
Most choropleth maps employ some method of data classification. The point of classification is to take a large number of observations and group them into data ranges or classes.

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_agemale2018)+
  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.

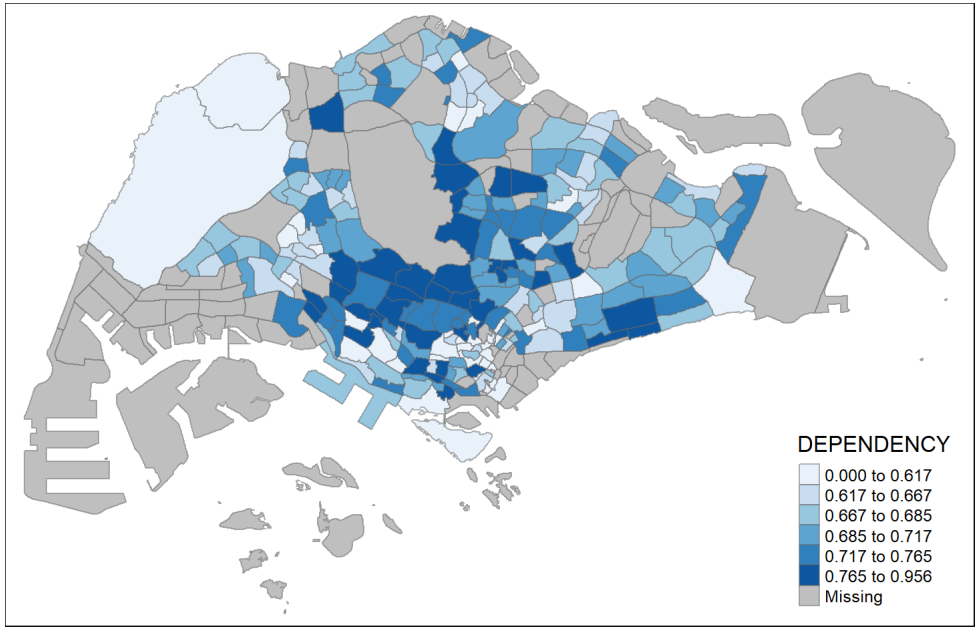
Colour Scheme

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 preferred colour to `palette` argument of `tm_fill()` as shown in the code chunk below.

```
tm_shape(mpsz_agemale2018)+
  tm_fill("DEPENDENCY",
          n = 6,
          style = "quantile",
          palette = "Blues") +
  tm_borders(alpha = 0.5)
```

Notice that the choropleth map is shaded in blue.

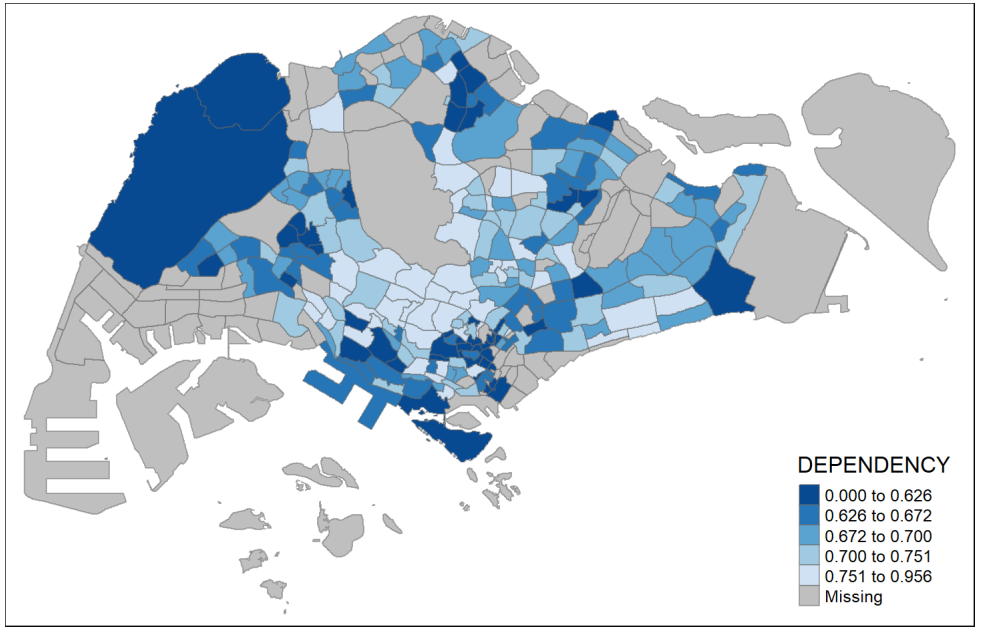


More about colour

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

```
tm_shape(mpsz_agemale2018)+
  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.

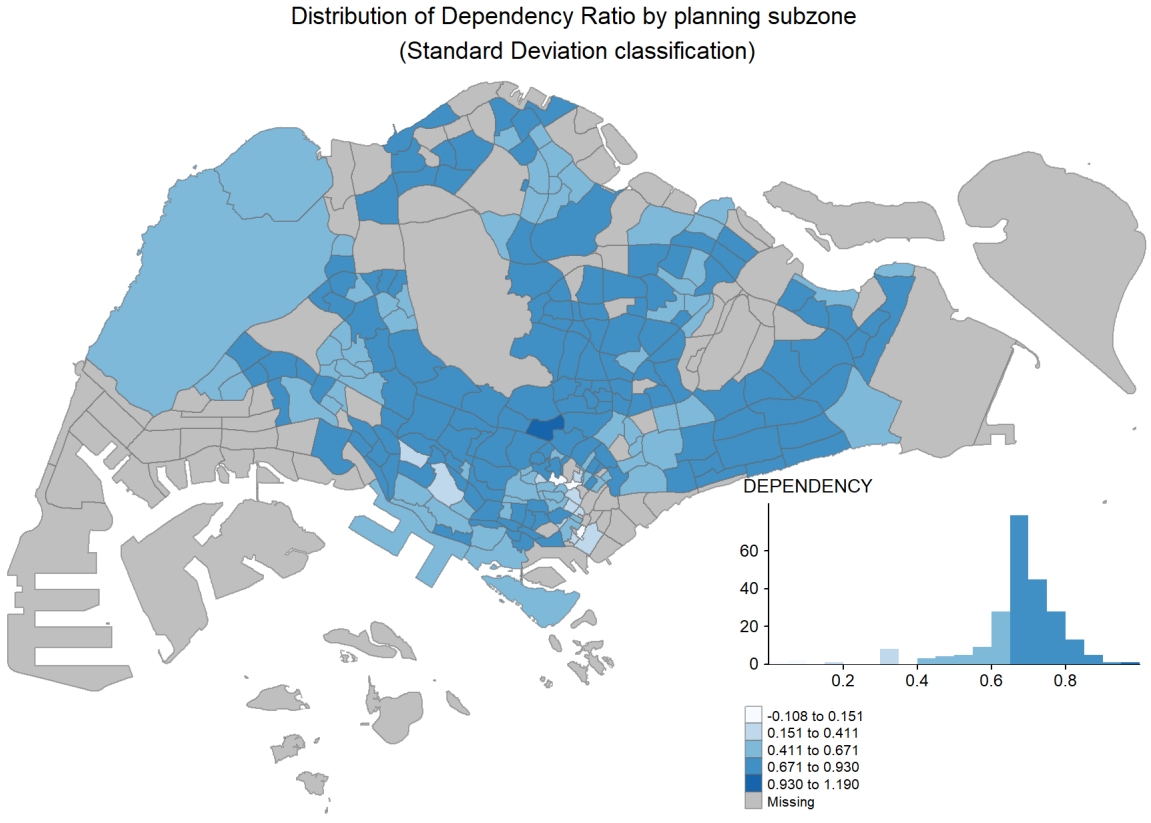
Map Legend

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

```
tm_shape(mpsz_agemale2018)+
  tm_fill("DEPENDENCY",
          style = "quantile",
          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(Quantile classifi
            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

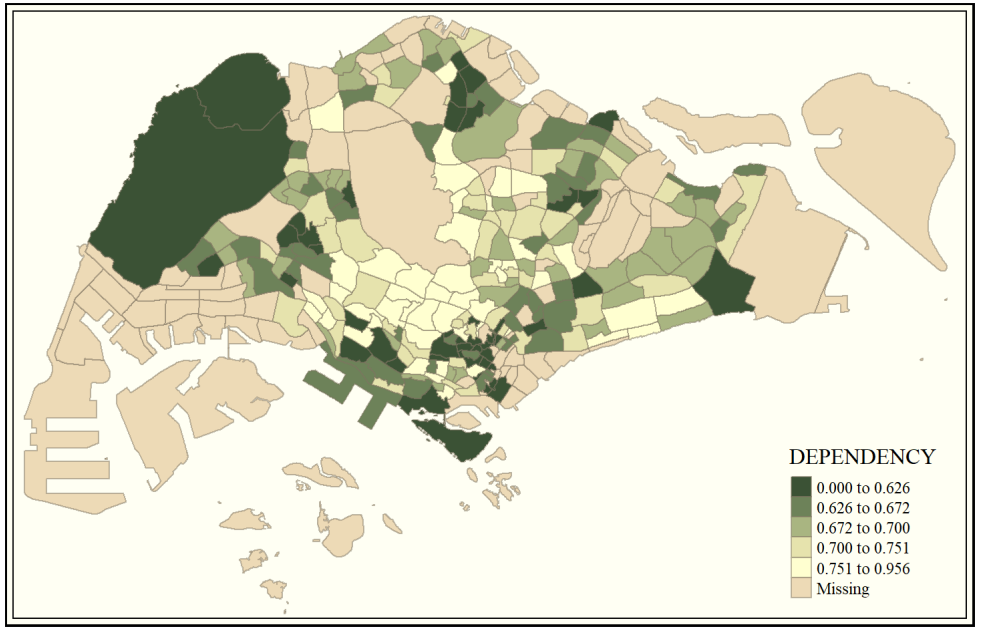


Map style

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_agemale2018)+
  tm_fill("DEPENDENCY",
          style = "quantile",
          palette = "-Greens") +
  tm_borders(alpha = 0.5) +
  tmap_style("classic")
```



Cartographic Furniture

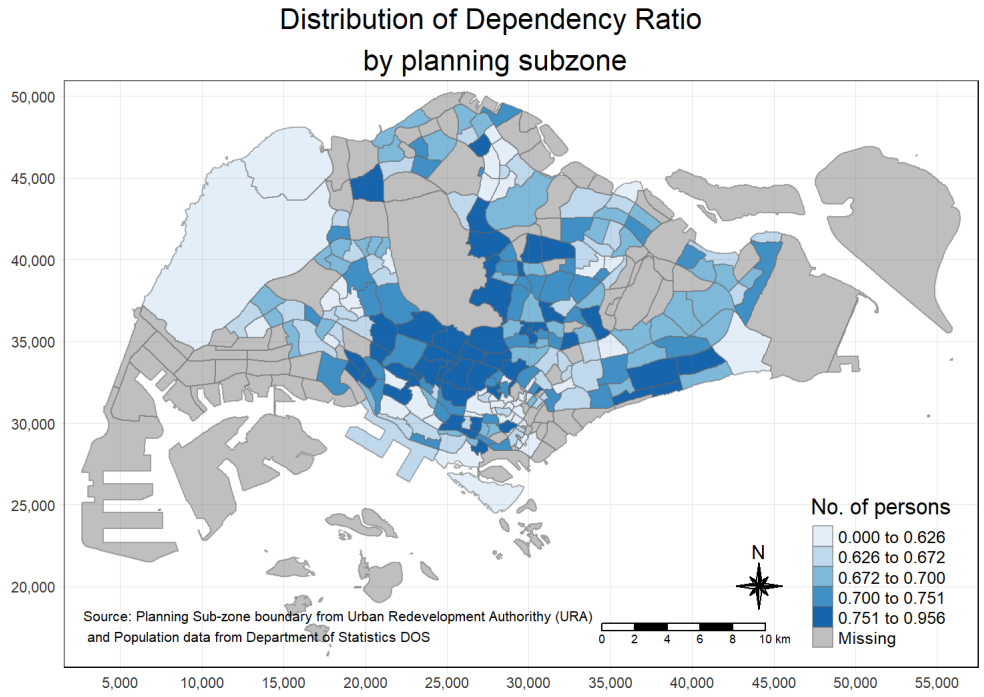
Beside map style, **tmap** also also provides arguments to draw other map furniture such as compass, scale bar and grid lines.

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_agemale2018)+
  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 back to the default style, the code chunk below should be used..

```
tmap_style("white")
```

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.

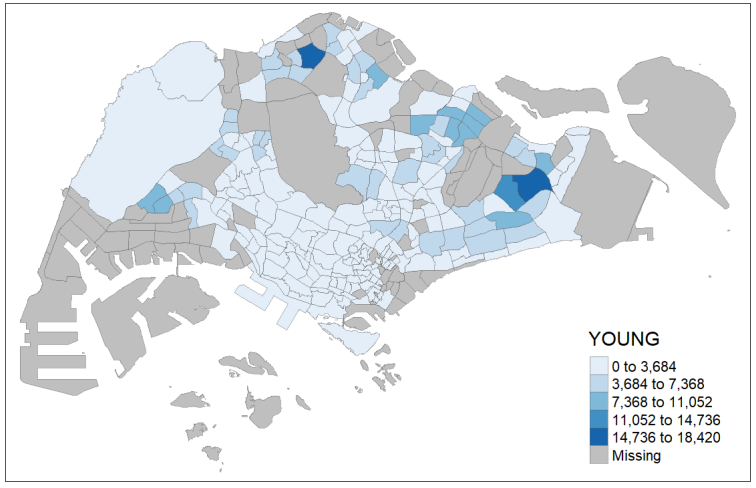
In **tmap**, small multiple maps can be plotted in three ways:

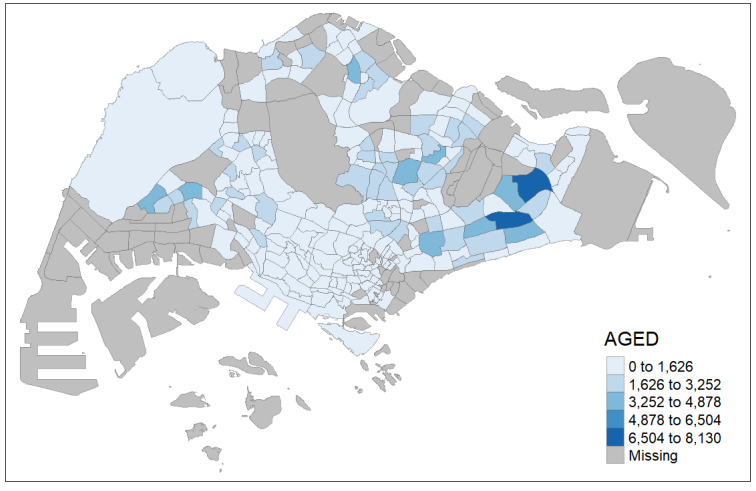
- by assigning multiple values to at least one of the asthetic arguments,
- by defining a group-by variable in `tm_facets()`, and
- by creating multiple stand-alone maps with `tmap_arrange()`.

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_agemale2018)+
  tm_fill(c("YOUNG", "AGED"),
          style = "equal",
          palette = "Blues") +
  tm_layout(legend.position = c("right",
                                "bottom")) +
  tm_borders(alpha = 0.5) +
  tmap_style("white")
```

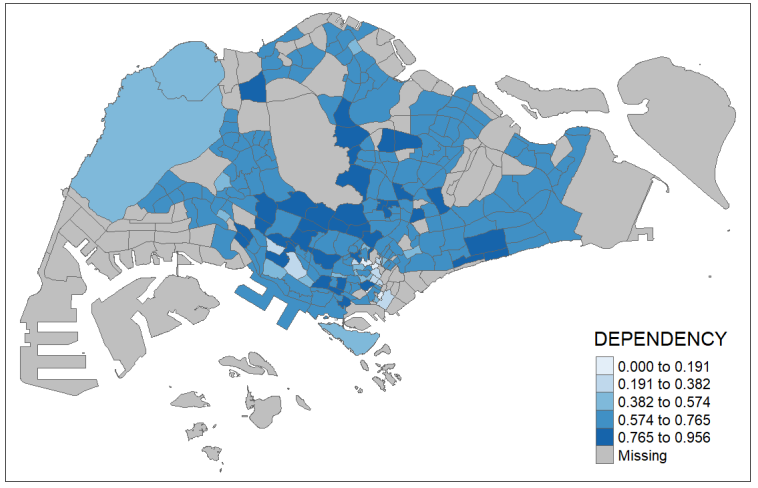


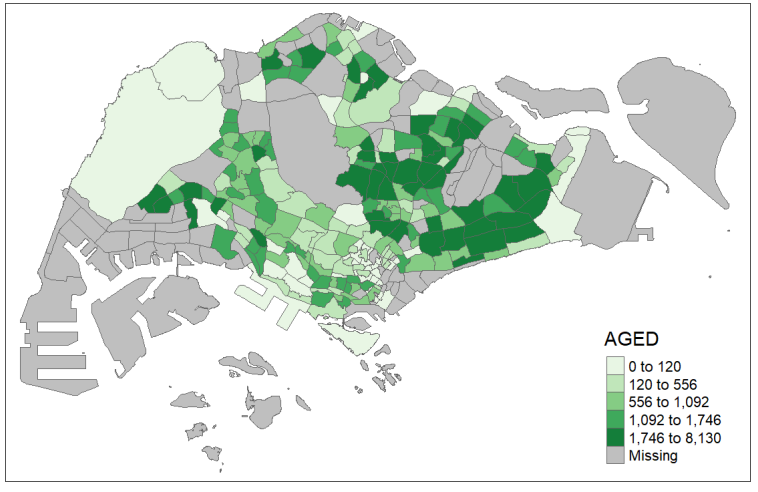


By assigning multiple values to at least one of the aesthetic arguments

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

```
tm_shape(mpsz_agemale2018)+
  tm_polygons(c("DEPENDENCY","AGED"),
          style = c("equal", "quantile"),
          palette = list("Blues","Greens")) +
  tm_layout(legend.position = c("right",
                                "bottom"))
```

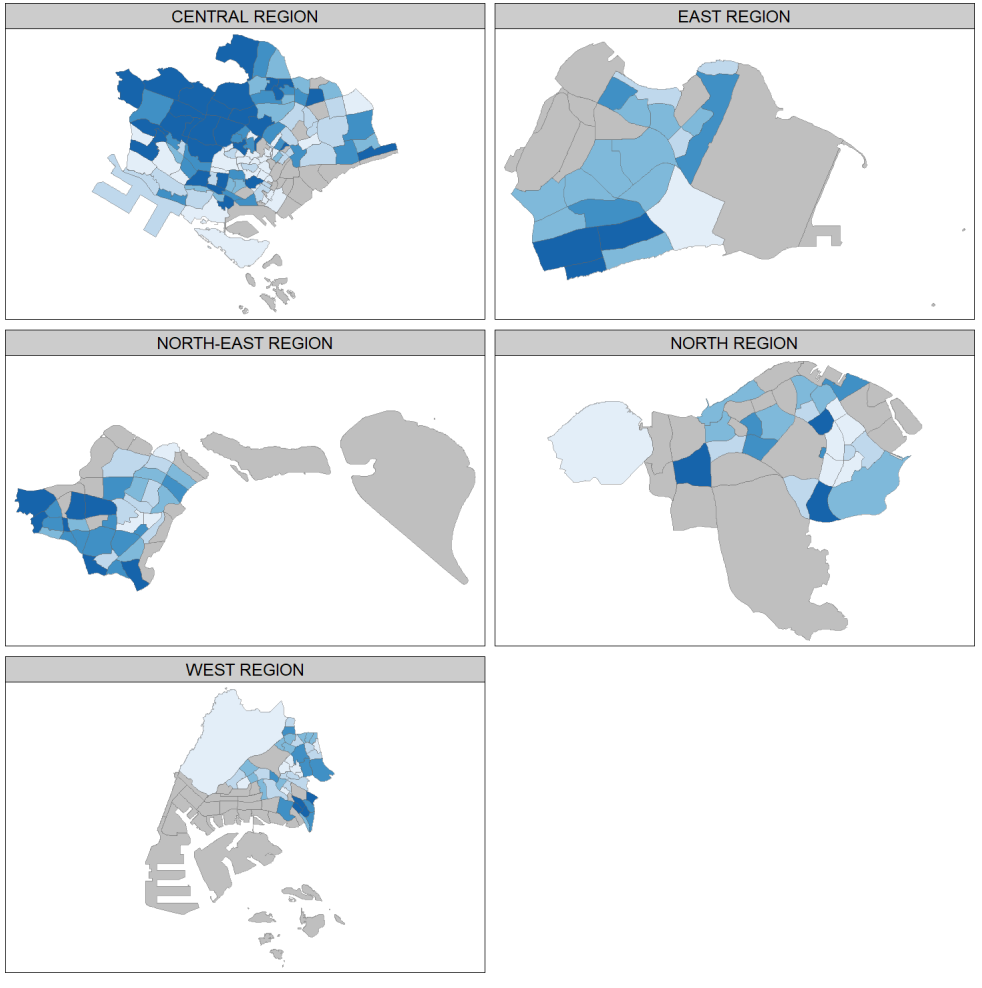




By defining a group-by variable in `tm_facets()`

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

```
tm_shape(mpsz_agemale2018) +
  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)
```



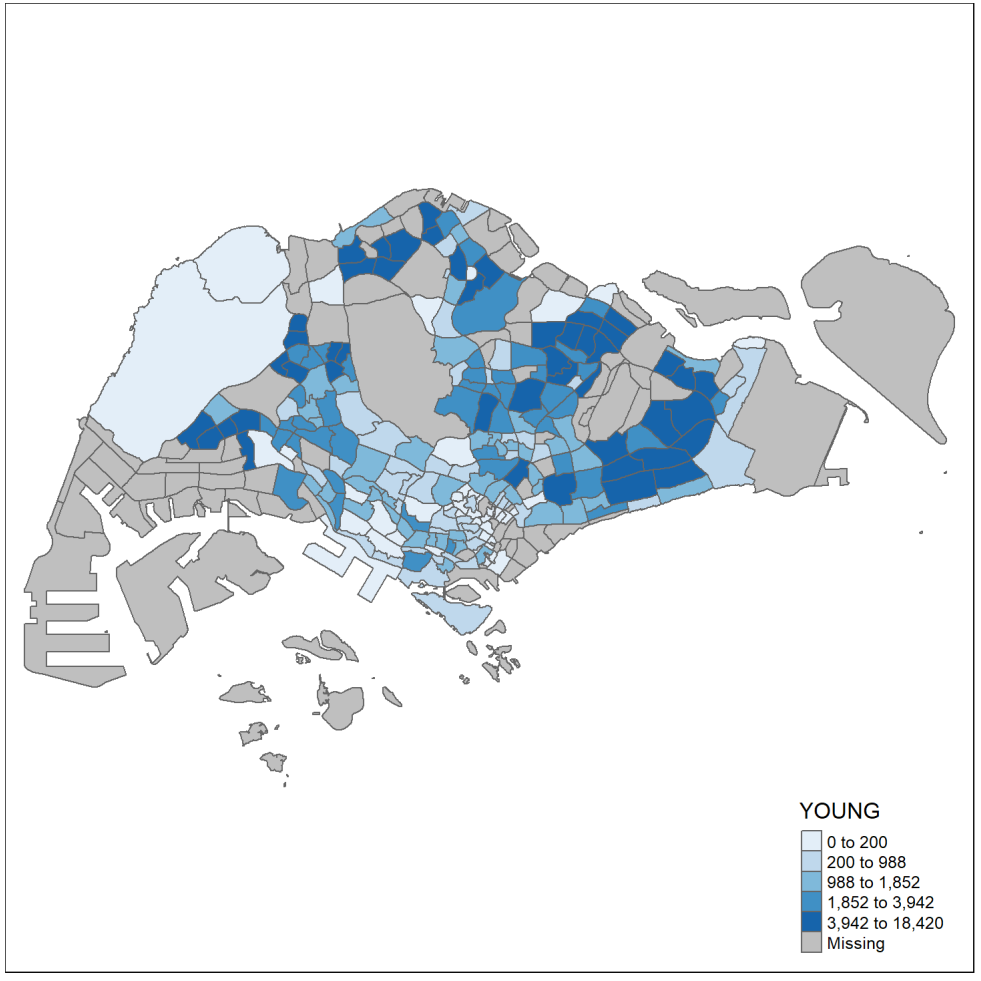
By creating multiple stand-alone maps with `tmap_arrange()`

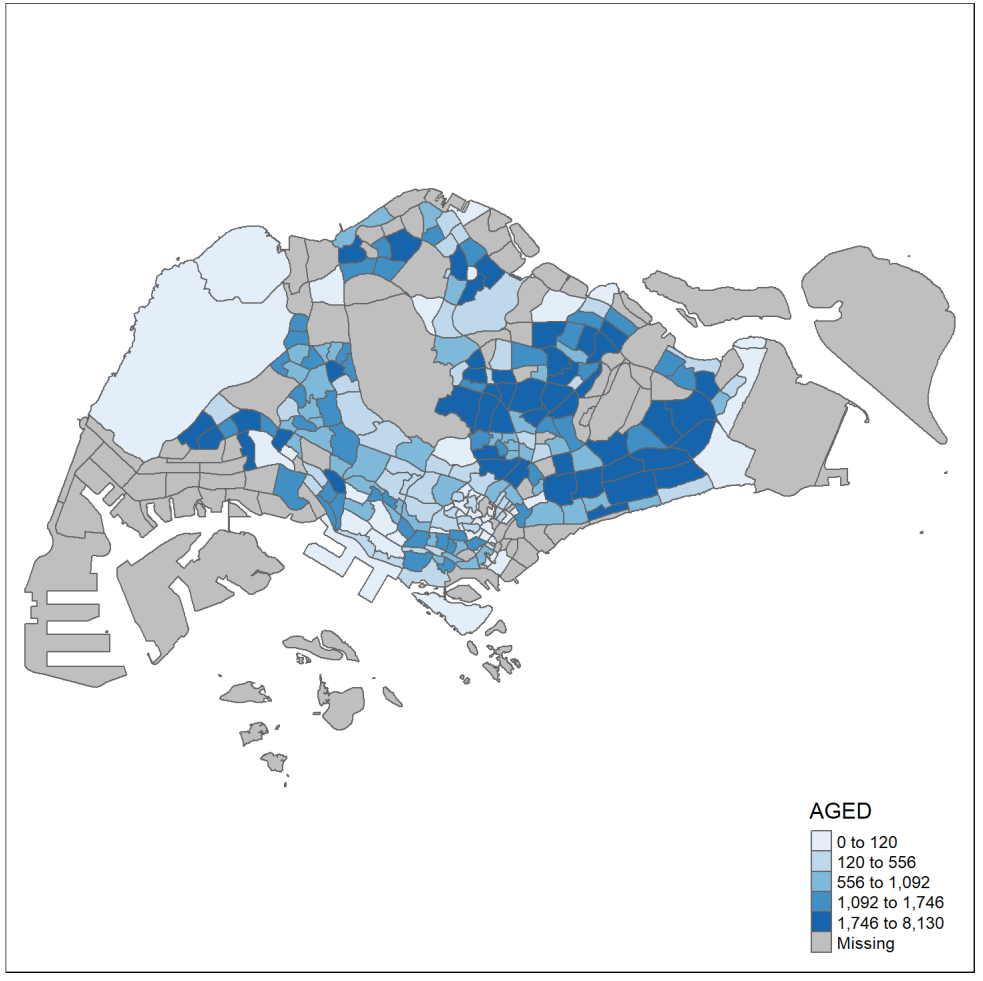
In this example, multiple small choropleth maps are created by creating multiple stand-alone maps with `tmap_arrange()`.

```
youngmap <- tm_shape(mpsz_agemale2018)+
  tm_polygons("YOUNG",
              style = "quantile",
              palette = "Blues")

agedmap <- tm_shape(mpsz_agemale2018)+
  tm_polygons("AGED",
              style = "quantile",
              palette = "Blues")

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

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

The output choropleth maps





Mappping Spatial Object Meeting a Selection Criterion

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

```
tm_shape(mpsz_agemale2018[mpsz_agemale2018$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.