Hands-on Exercise 8: Geographical Segmentation withSpatially Constrained Clustering Techniques

In this hands-on exercise, you will learn how to perform geographical segmentation by using appropriate R packages. You will also expose to R packages for performing cluster analysis and visualising clustering results.

AUTHOR

Dr. Kam Tin Seong, Associate Professor of Information Systems (Practice)

AFFILIATION

School of Computing and Information Systems, Singapore Management University

PUBLISHED

Oct. 10, 2021

Contents

Introduction

In this hands-on exercise, you will gain hands-on experience on how to delineate homogeneous region by using geographically referenced multivariate data. There are two major analysis, namely:

- hierarchical cluster analysis; and
- spatially constrained cluster analysis.

Learning Outcome

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

- to convert GIS polygon data into R's simple feature data.frame by using appropriate functions of **sf** package of R;
- to convert simple feature data.frame into R's SpatialPolygonDataFrame object by using appropriate **sf** of package of R;
- to perform custer analysis by using *hclust()* of Base R;
- to perform spatially constrained cluster analysis using *skater()* of Base R; and

- to visualise the analysis output by using **ggplot2** and **tmap** package.

Getting Started

The analytical question

In geobusiness and spatial policy, it is a common practice to delineate the market or planning area into homogeneous regions by using multivariate data. In this hands-on exercise, we are interested to delineate Shan State, Myanmar into homogeneous regions by using multiple Information and Communication technology (ICT) measures, namely: Radio, Television, Land line phone, Mobile phone, Computer, and Internet at home.

The data

Two data sets will be used in this study. They are:

- Myanmar Township Boundary Data (i.e. *myanmar_township_boundaries*) : This is a GIS data in ESRI shapefile format. It consists of township boundary information of Myanmar. The spatial data are captured in polygon features.

- *Shan-ICT.csv*: This is an extract of **The 2014 Myanmar Population and Housing Census Myanmar** at the township level.

Both data sets are download from Myanmar Information Management Unit (MIMU)

Installing and loading R packages

Before we get started, it is important for us to install the necessary R packages into R and launch these R packages into R environment.

The R packages needed for this exercise are as follows:

- Spatial data handling
  - **sf**, **rgdal** and **spdep**

- Attribute data handling
  - **tidyverse**, especially **readr**, **ggplot2** and **dplyr**

- Choropleth mapping
  - **tmap**

- Multivariate data visualisation and analysis
  - **coorplot**, **ggpubr**, and **heatmaply**

- **coorplot**, **ggpubr**, and **heatmaply**

- Cluster analysis
  - **cluster**

The code chunks below installs and launches these R packages into R environment.

```
packages = c('rgdal', 'spdep', 'tmap', 'sf', 'ggpubr',
'cluster', 'factoextra', 'NbClust', 'heatmaply', 'corrplot', 'psych', 'tidyverse')
for (p in packages){
  if(!require(p, character.only = T)){
    install.packages(p)
    }
  library(p,character.only = T)
}
```

Note: With **tidyverse**, we do not have to install **readr**, **ggplot2** and **dplyr** packages separately. In fact, **tidyverse** also installs other very useful R packages such as **tidyr**.

Data Import and Prepatation

Importing geospatial data into R environment

In this section, you will import Myanmar Township Boundary GIS data and its associated attrbiute table into R environment.

The Myanmar Township Boundary GIS data is in ESRI shapefile format. It will be imported into R environment by using the *st_read()* function of **sf**.

The code chunks used are shown below:

```
shan_sf <- st_read(dsn = "data/geospatial", layer =
"myanmar_township_boundaries") %>%
  filter(ST %in% c("Shan (East)", "Shan (North)",
"Shan (South)"))
```

```
Reading layer `myanmar_township_boundaries' from data source
  `D:\tskam\IS415\Hands-on_Ex\Hands-on_Ex08\data\geospatial'
  using driver `ESRI Shapefile'
Simple feature collection with 330 features and 14 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 92.17275 ymin: 9.671252 xmax: 101.1699 ymax: 28.54554
Geodetic CRS:  WGS 84
```

The imported township boundary object is called *shan_sf*. It is saved in **simple feature data.frame** format.

We can view the content of the newly created *shan_sf* simple features data.frame by using the code chunk below.

```
shan_sf
```

```
Simple feature collection with 55 features and 14 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 96.15107 ymin: 19.29932 xmax: 101.1699 ymax: 24.15907
Geodetic CRS:  WGS 84
First 10 features:
   OBJECTID           ST ST_PCODE       DT   DT_PCODE        TS
1       163 Shan (North)   MMR015  Mongmit MMR015D008   Mongmit
2       203 Shan (South)   MMR014 Taunggyi MMR014D001   Pindaya
3       240 Shan (South)   MMR014 Taunggyi MMR014D001   Ywangan
4       106 Shan (South)   MMR014 Taunggyi MMR014D001  Pinlaung
5        72 Shan (North)   MMR015  Mongmit MMR015D008    Mabein
6        40 Shan (South)   MMR014 Taunggyi MMR014D001     Kalaw
7       194 Shan (South)   MMR014 Taunggyi MMR014D001     Pekon
8       159 Shan (South)   MMR014 Taunggyi MMR014D001  Lawksawk
9        61 Shan (North)   MMR015  Kyaukme MMR015D003 Nawnghkio
10      124 Shan (North)   MMR015  Kyaukme MMR015D003   Kyaukme
    TS_PCODE                 ST_2            LABEL2 SELF_ADMIN ST_RG
1  MMR015017 Shan State (North)    Mongmit\n61072       <NA> State
2  MMR014006 Shan State (South)    Pindaya\n77769       Danu State
3  MMR014007 Shan State (South)    Ywangan\n76933       Danu State
4  MMR014009 Shan State (South)  Pinlaung\n162537       Pa-O State
5  MMR015018 Shan State (North)     Mabein\n35718       <NA> State
6  MMR014005 Shan State (South)     Kalaw\n163138       <NA> State
7  MMR014010 Shan State (South)      Pekon\n94226       <NA> State

8  MMR014008 Shan State (South)          Lawksawk       <NA> State
9  MMR015013 Shan State (North) Nawnghkio\n128357       <NA> State
10 MMR015012 Shan State (North)   Kyaukme\n172874       <NA> State
   T_NAME_WIN
1    rdk;rdwf
2      yif;w,
3       &GmiH
4   yifavmif;
5      rbdrf;
6        uavm
7      z,fcHk
8    &yfapmuf
9   aemifcsdK
10   ausmufrJ
                                                             T_NAME_M3
1          <U+1019><U+102D><U+102F><U+1038><U+1019><U+102D><U+1010><U+103A>
2                          <U+1015><U+1004><U+103A><U+1038><U+1010><U+101A>
3                                  <U+101B><U+103D><U+102C><U+1004><U+1036>
4  <U+1015><U+1004><U+103A><U+101C><U+1031><U+102C><U+1004><U+103A><U+1038>
5                          <U+1019><U+1018><U+102D><U+1019><U+103A><U+1038>
```

```
6                                                 <U+1000><U+101C><U+1031><U+102C>
7                                 <U+1016><U+101A><U+103A><U+1001><U+102F><U+1036>
8         <U+101B><U+1015><U+103A><U+1005><U+1031><U+102C><U+1000><U+103A>
9   <U+1014><U+1031><U+102C><U+1004><U+103A><U+1001><U+103B><U+102D><U+102F>
10        <U+1000><U+103B><U+1031><U+102C><U+1000><U+103A><U+1019><U+1032>
       AREA                       geometry
1  2703.611 MULTIPOLYGON (((96.96001 23...
2   629.025 MULTIPOLYGON (((96.7731 21....
3  2984.377 MULTIPOLYGON (((96.78483 21...
4  3396.963 MULTIPOLYGON (((96.49518 20...
5  5034.413 MULTIPOLYGON (((96.66306 24...
6  1456.624 MULTIPOLYGON (((96.49518 20...
7  2073.513 MULTIPOLYGON (((97.14738 19...
8  5145.659 MULTIPOLYGON (((96.94981 22...
9  3271.537 MULTIPOLYGON (((96.75648 22...
10 3920.869 MULTIPOLYGON (((96.95498 22...
```

Notice that sf.data.frame is conformed to Hardy Wickham's tidy framework.

Since *shan_sf* is conformed to tidy framework, we can also *glimpse()* to reveal the data type of it's fields.

```
glimpse(shan_sf)
```

```
Rows: 55
Columns: 15
$ OBJECTID   <dbl> 163, 203, 240, 106, 72, 40, 194, 159, 61, 124, 71~
$ ST         <chr> "Shan (North)", "Shan (South)", "Shan (South)", "~
$ ST_PCODE   <chr> "MMR015", "MMR014", "MMR014", "MMR014", "MMR015",~
$ DT         <chr> "Mongmit", "Taunggyi", "Taunggyi", "Taunggyi", "M~
$ DT_PCODE   <chr> "MMR015D008", "MMR014D001", "MMR014D001", "MMR014~
$ TS         <chr> "Mongmit", "Pindaya", "Ywangan", "Pinlaung", "Mab~
$ TS_PCODE   <chr> "MMR015017", "MMR014006", "MMR014007", "MMR014009~
$ ST_2       <chr> "Shan State (North)", "Shan State (South)", "Shan~
$ LABEL2     <chr> "Mongmit\n61072", "Pindaya\n77769", "Ywangan\n769~
$ SELF_ADMIN <chr> NA, "Danu", "Danu", "Pa-O", NA, NA, NA, NA, NA, N~
$ ST_RG      <chr> "State", "State", "State", "State", "State", "Sta~
$ T_NAME_WIN <chr> "rdk;rdwf", "yif;w,", "&GmiH", "yifavmif;", "rbdr~
$ T_NAME_M3  <chr> "<U+1019><U+102D><U+102F><U+1038><U+1019><U+102D><U+1010><U+103A>", "<U+1015><U+10
$ AREA       <dbl> 2703.611, 629.025, 2984.377, 3396.963, 5034.413, ~
$ geometry   <MULTIPOLYGON [°]> MULTIPOLYGON (((96.96001 23..., MULT~
```

Importing aspatial data into R environment

The csv file will be import using *read_csv* function of **readr** package.

The code chunks used are shown below:

```
ict <- read_csv("data/aspatial/Shan-ICT.csv")
```

The imported InfoComm variables are extracted from **The 2014 Myanmar Population and Housing Census Myanmar**. The attribute data set is called *ict*. It is saved in R's * tibble data.frame* format.

The code chunk below reveal the summary statistics of *ict* data.frame.

```
summary(ict)
```

```
 District Pcode     District Name      Township Pcode    
 Length:55          Length:55          Length:55         
 Class :character   Class :character   Class :character  
 Mode  :character   Mode  :character   Mode  :character  



 Township Name      Total households     Radio          Television   
 Length:55          Min.   : 3318    Min.   :  115   Min.   :  728  
 Class :character   1st Qu.: 8711    1st Qu.: 1260   1st Qu.: 3744  
 Mode  :character   Median :13685    Median : 2497   Median : 6117  
                    Mean   :18369    Mean   : 4487   Mean   :10183  
                    3rd Qu.:23471    3rd Qu.: 6192   3rd Qu.:13906  
                    Max.   :82604    Max.   :30176   Max.   :62388  
 Land line phone  Mobile phone      Computer      Internet at home
 Min.   :  20.0   Min.   :  150   Min.   :  20.0   Min.   :   8.0  
 1st Qu.: 266.5   1st Qu.: 2037   1st Qu.: 121.0   1st Qu.:  88.0  
 Median : 695.0   Median : 3559   Median : 244.0   Median : 316.0  
 Mean   : 929.9   Mean   : 6470   Mean   : 575.5   Mean   : 760.2  
 3rd Qu.:1082.5   3rd Qu.: 7177   3rd Qu.: 507.0   3rd Qu.: 630.5  
 Max.   :6736.0   Max.   :48461   Max.   :6705.0   Max.   :9746.0  
```

There are a total of eleven fields and 55 observation in the tibble data.frame.

Derive new variables using **dplyr** package

The unit of measurement of the values are number of household. Using these values directly will be bias by the underlying total number of households. In general, the townships with relatively higher total number of households will also have higher number of households owning radio, TV, etc.

In order to overcome this problem, we will derive the penetration rate of each ICT variable by using the code chunk below.

```
ict_derived <- ict %>%
  mutate(`RADIO_PR` = `Radio`/`Total households`*1000) %>%
  mutate(`TV_PR` = `Television`/`Total households`*1000) %>%
  mutate(`LLPHONE_PR` = `Land line phone`/`Total households`*1000) %>%
  mutate(`MPHONE_PR` = `Mobile phone`/`Total households`*
```

```
1000      )         %>%
  mutate    (         `COMPUTER_PR` =         `Computer`/        `Total households`   *
1000      )         %>%
  mutate    (         `INTERNET_PR` =         `Internet at home`/        `Total households`
*         1000      )         %>%
  rename    (         `DT_PCODE` =         `District Pcode`  ,`DT`=         `District Name`   ,
         `TS_PCODE`=         `Township Pcode`  , `TS`=         `Township Name`   ,
         `TT_HOUSEHOLDS`=         `Total households`  ,
         `RADIO`=         `Radio`  , `TV`=         `Television`,
         `LLPHONE`=         `Land line phone`, `MPHONE`=         `Mobile phone`  ,
         `COMPUTER`=         `Computer`, `INTERNET`=         `Internet at home`)
```

Let us review the summary statistics of the newly derived penetration rates using the code chunk below.

```
summary  (         ict_derived)
```

```
   DT_PCODE              DT              TS_PCODE        
 Length:55          Length:55          Length:55         
 Class :character   Class :character   Class :character  
 Mode  :character   Mode  :character   Mode  :character  



      TS            TT_HOUSEHOLDS       RADIO             TV       
 Length:55          Min.   : 3318   Min.   :  115   Min.   :  728  
 Class :character   1st Qu.: 8711   1st Qu.: 1260   1st Qu.: 3744  
 Mode  :character   Median :13685   Median : 2497   Median : 6117  
                    Mean   :18369   Mean   : 4487   Mean   :10183  
                    3rd Qu.:23471   3rd Qu.: 6192   3rd Qu.:13906  
                    Max.   :82604   Max.   :30176   Max.   :62388  
    LLPHONE            MPHONE         COMPUTER          INTERNET     
 Min.   :  20.0   Min.   :  150   Min.   :  20.0   Min.   :   8.0  
 1st Qu.: 266.5   1st Qu.: 2037   1st Qu.: 121.0   1st Qu.:  88.0  
 Median : 695.0   Median : 3559   Median : 244.0   Median : 316.0  
 Mean   : 929.9   Mean   : 6470   Mean   : 575.5   Mean   : 760.2  
 3rd Qu.:1082.5   3rd Qu.: 7177   3rd Qu.: 507.0   3rd Qu.: 630.5  
 Max.   :6736.0   Max.   :48461   Max.   :6705.0   Max.   :9746.0  
    RADIO_PR           TV_PR         LLPHONE_PR       MPHONE_PR     
 Min.   : 21.05   Min.   :116.0   Min.   :  2.78   Min.   : 36.42  
 1st Qu.:138.95   1st Qu.:450.2   1st Qu.: 22.84   1st Qu.:190.14  
 Median :210.95   Median :517.2   Median : 37.59   Median :305.27  

 Mean   :215.68   Mean   :509.5   Mean   : 51.09   Mean   :314.05  
 3rd Qu.:268.07   3rd Qu.:606.4   3rd Qu.: 69.72   3rd Qu.:428.43  
 Max.   :484.52   Max.   :842.5   Max.   :181.49   Max.   :735.43  
  COMPUTER_PR      INTERNET_PR     
 Min.   : 3.278   Min.   :  1.041  
 1st Qu.:11.832   1st Qu.:  8.617  
 Median :18.970   Median : 22.829  
 Mean   :24.393   Mean   : 30.644  
 3rd Qu.:29.897   3rd Qu.: 41.281  
 Max.   :92.402   Max.   :117.985  
```

```
Max.   :92.462   Max.   :117.985
```

Notice that six new fields have been added into the data.frame. They are RADIO_PR, TV_PR, LLPHONE_PR, MPHONE_PR, COMPUTER_PR, and INTERNET_PR.

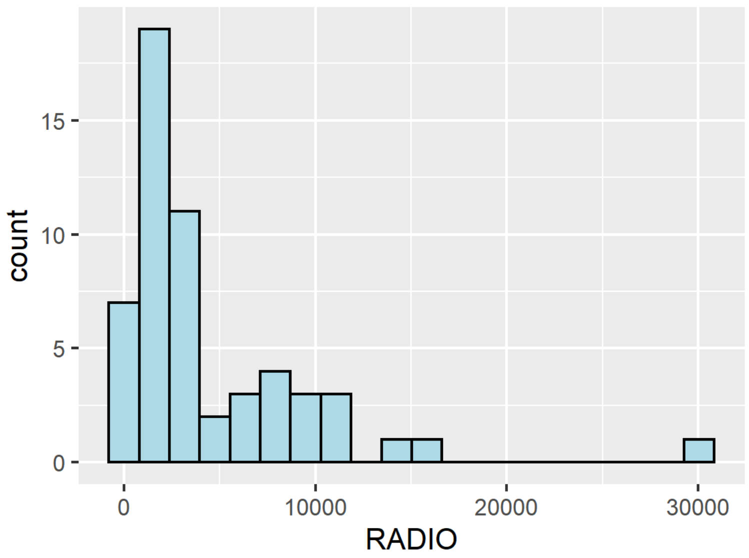
Exploratory Data Analysis (EDA)

EDA using statistical graphics

We can plot the distribution of the variables (i.e. Number of households with radio) by using appropriate Exploratory Data Analysis (EDA) as shown in the code chunk below.

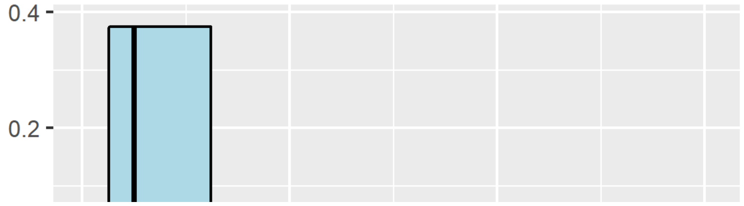
Histogram is useful to identify the overall distribution of the data values (i.e. left skew, right skew or normal distribution)

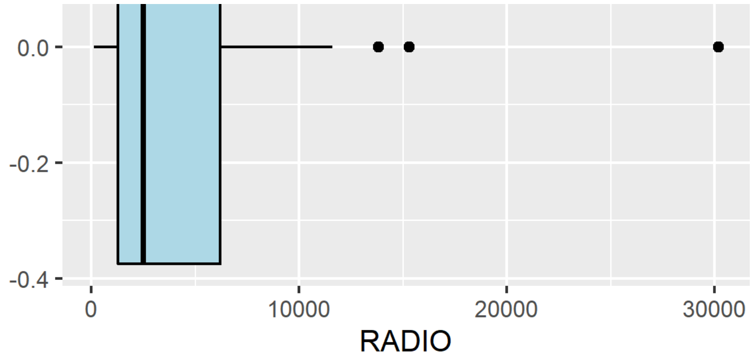
```
ggplot(data=ict_derived, aes(x=`RADIO`)) +
  geom_histogram(bins=20, color="black", fill="light blue")
```



Boxplot is useful to detect if there are outliers.

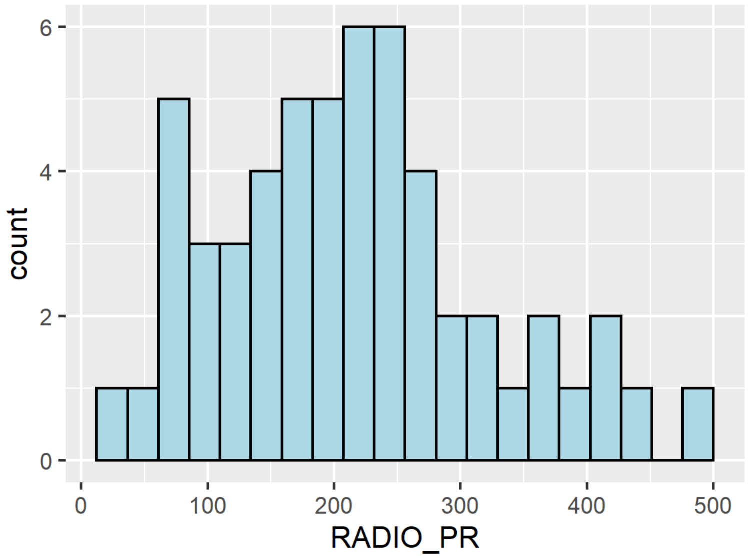
```
ggplot(data=ict_derived, aes(x=`RADIO`)) +
  geom_boxplot(color="black", fill="light blue")
```



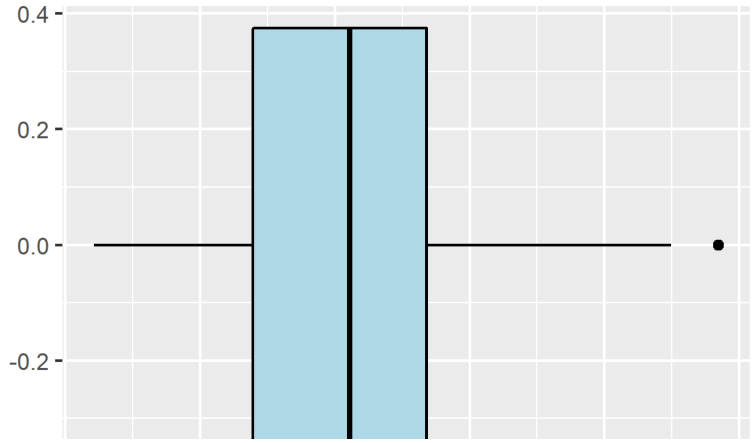


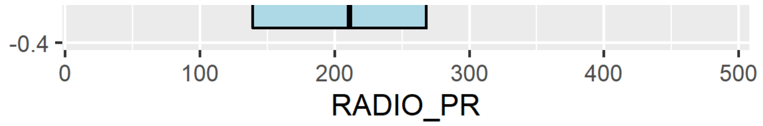
Next, we will also plotting the distribution of the newly derived variables (i.e. Radio penetration rate) by using the code chunk below.

```
ggplot(data=ict_derived, aes(x=`RADIO_PR`)) +
  geom_histogram(bins=20, color="black", fill="light blue")
```



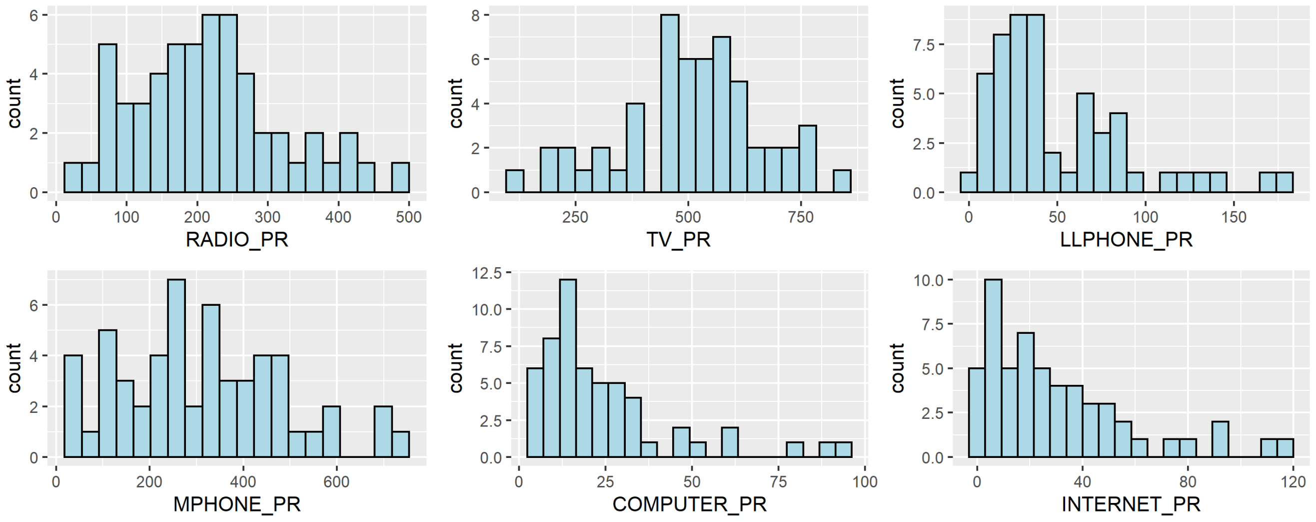

```
ggplot(data=ict_derived, aes(x=`RADIO_PR`)) +
  geom_boxplot(color="black", fill="light blue")
```





What can you observed from the distributions reveal in the histogram and boxplot.

In the figure below, multiple histograms are plotted to reveal the distribution of the selected variables in the *ict_derived* data.frame.



The code chunks below are used to create the data visualisation. They consist of two main parts. First, we will create the individual histograms using the code chunk below.

```
radio     <-        ggplot    (        data=        ict_derived,
               aes        (        x=         `RADIO_PR`)         )         +
  geom_histogram(        bins=        20        ,
                      color=        "black"   ,
                      fill=        "light blue"    )

tv        <-        ggplot    (        data=        ict_derived,
               aes        (        x=         `TV_PR`   )         )         +
  geom_histogram(        bins=        20        ,
                      color=        "black"   ,
                      fill=        "light blue"    )

llphone   <-        ggplot    (        data=        ict_derived,
               aes        (        x=         `LLPHONE_PR`)         )         +
  geom_histogram(        bins=        20        ,
                      color=        "black"   ,
                      fill=        "light blue"    )

mphone    <-        ggplot    (        data=        ict_derived,
               aes        (        x=         `MPHONE_PR`)         )         +
  geom_histogram(        bins=        20        ,
                      color=        "black"   ,
                      fill=        "light blue"    )
```

```
computer <- ggplot(data=ict_derived,

       aes(x=`COMPUTER_PR`)) +
  geom_histogram(bins=20,
                 color="black",
                 fill="light blue")

internet <- ggplot(data=ict_derived,
       aes(x=`INTERNET_PR`)) +
  geom_histogram(bins=20,
                 color="black",
                 fill="light blue")
```

Next, the *ggarange()* function of **ggpubr** package is used to group these histograms together.

```
ggarrange(radio, tv, llphone, mphone, computer, internet,
          ncol = 3,
          nrow = 2)
```

EDA using choropleth map

Joining geospatial data with aspatial data

Before we can prepare the choropleth map, we need to combine both the geospatial data object (i.e. *shan_sf*) and aspatial data.frame object (i.e. *ict_derived*) into one. This will be performed by using the *left_join* function of **dplyr** package. The *shan_sf* simple feature data.frame will be used as the base data object and the *ict_derived* data.frame will be used as the join table.

The code chunks below is used to perform the task. The unique identifier used to join both data objects is *TS_PCODE*.

```
shan_sf <- left_join(shan_sf, ict_derived, by=c(
"TS_PCODE"="TS_PCODE"))
```

The message above shows that *TS_CODE* field is the common field used to perform the left-join.

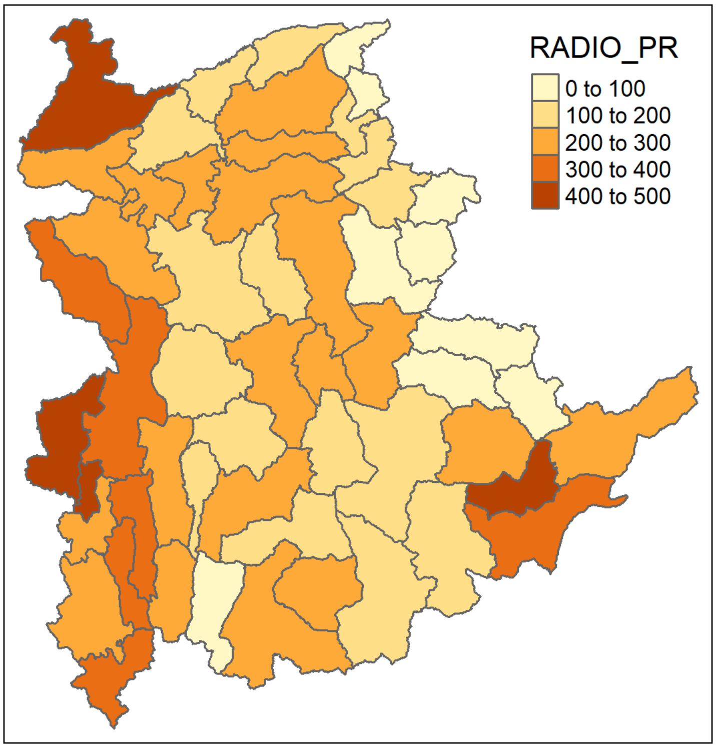
It is important to note that there is no new output data been created. Instead, the data fields from *ict_derived* data frame are now updated into the data frame of *shan_sf*.

Preparing a choropleth map

To have a quick look at the distribution of Radio penetration rate of Shan State at township level, a choropleth map will be prepared.

The code chunks below are used to prepare the choroplethby using the *qtm()* function of **tmap** package.

```r
qtm(shan_sf, "RADIO_PR")
```

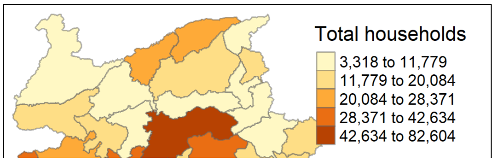


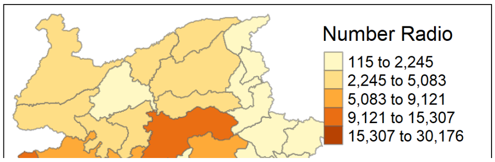
In order to reveal the distribution shown in the choropleth map above are bias to the underlying total number of households at the townships, we will create two choropleth maps, one for the total number of households (i.e. TT_HOUSEHOLDS.map) and one for the total number of household with Radio (RADIO.map) by using the code chunk below.

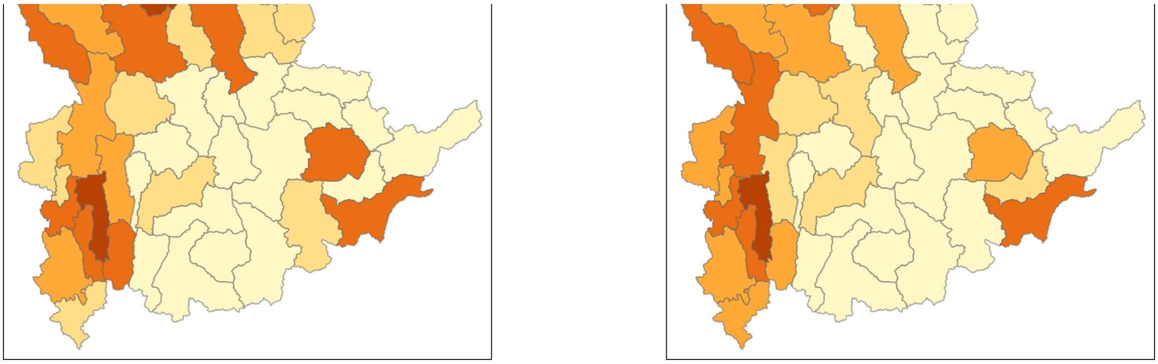
```r
TT_HOUSEHOLDS.map <- tm_shape(shan_sf) +
  tm_fill(col = "TT_HOUSEHOLDS",
          n = 5,
          style = "jenks",
          title = "Total households") +
  tm_borders(alpha = 0.5)

RADIO.map <- tm_shape(shan_sf) +
  tm_fill(col = "RADIO",
          n = 5,
          style = "jenks",
          title = "Number Radio ") +
  tm_borders(alpha = 0.5)

tmap_arrange(TT_HOUSEHOLDS.map, RADIO.map,
             asp=NA, ncol=2)
```



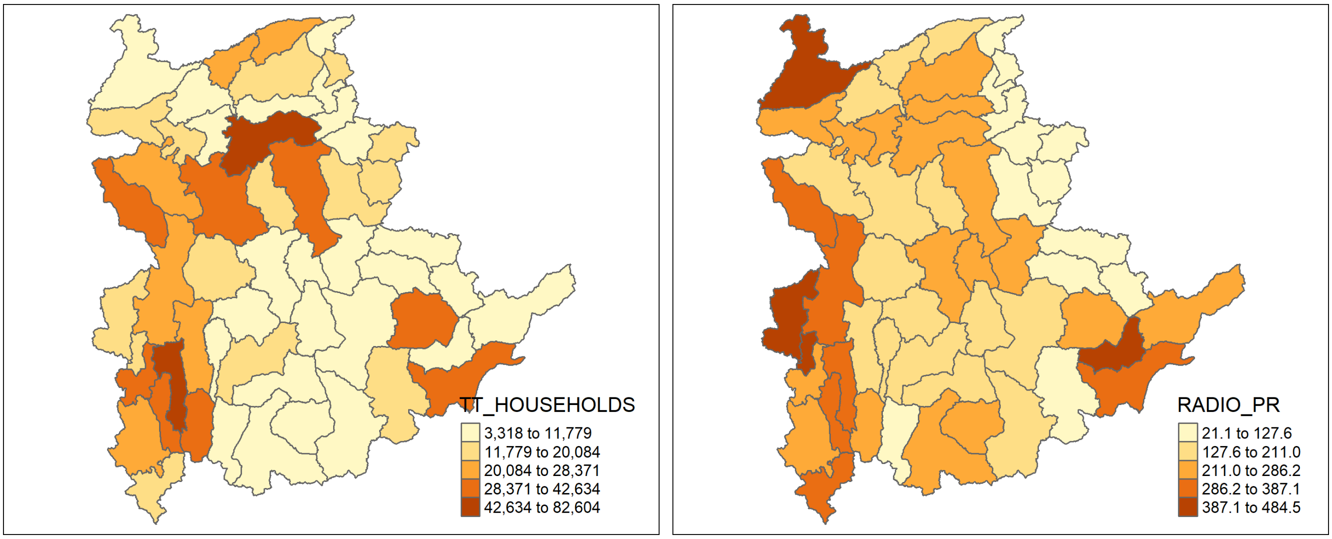




Notice that the choropleth maps above clearly show that townships with relatively larger number ot households are also showing relatively higher number of radio ownership.

Now let us plot the choropleth maps showing the dsitribution of total number of households and Radio penetration rate by using the code chunk below.

```
tm_shape(shan_sf) +
    tm_polygons(c("TT_HOUSEHOLDS", "RADIO_PR"),
                style="jenks") +
    tm_facets(sync = TRUE, ncol = 2) +
  tm_legend(legend.position = c("right", "bottom"))+
  tm_layout(outer.margins=0, asp=0)
```



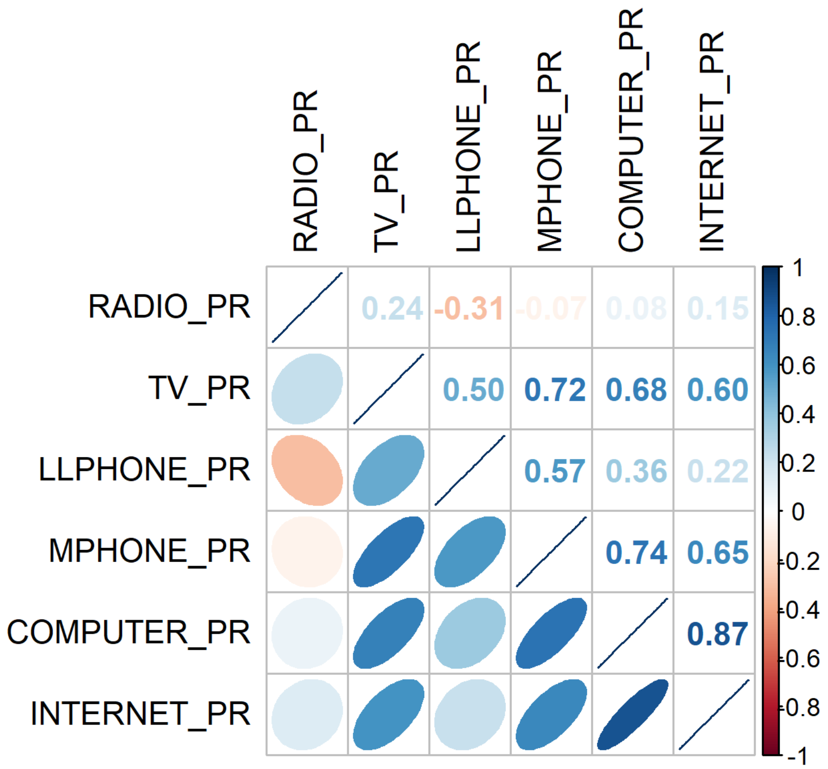
Can you identify the differences?

Correlation Analysis

Before we perform cluster analysis, it is important for us to ensure that the cluster variables are not highly correlated.

In this section, you will learn how to use *corrplot.mixed()* function of **corrplot** package to visualise and analyse the correlation of the input variables.

```
cluster_vars.cor = cor(ict_derived[,12:17])
corrplot.mixed(cluster_vars.cor,
         lower = "ellipse",
               upper = "number",
               tl.pos = "lt",
               diag = "l",
               tl.col = "black")
```



The correlation plot above shows that COMPUTER_PR and INTERNET_PR are highly correlated. This suggest that only one of them should be used in the cluster analysis instead of both.

Hierarchy Cluster Analysis

In this section, you will learn how to perform hierarchical cluster analysis. The analysis consists of four major steps:

Extrating clustering variables

The code chunk below will be used to extract the clustering variables from the *shan_sf* simple feature object into data.frame.

```
cluster_vars <- shan_sf %>%
  st_set_geometry(NULL) %>%
  select("TS.x", "RADIO_PR", "TV_PR", "LLPHONE_PR", "MPHONE_PR", "COMPUTER_PR"
```

```r
)
head        (        cluster_vars,10        )
```

```
       TS.x RADIO_PR    TV_PR LLPHONE_PR MPHONE_PR COMPUTER_PR
1    Mongmit 286.1852 554.1313   35.30618  260.6944    12.15939
2    Pindaya 417.4647 505.1300   19.83584  162.3917    12.88190
3    Ywangan 484.5215 260.5734   11.93591  120.2856     4.41465
4   Pinlaung 231.6499 541.7189   28.54454  249.4903    13.76255
5     Mabein 449.4903 708.6423   72.75255  392.6089    16.45042
6      Kalaw 280.7624 611.6204   42.06478  408.7951    29.63160
7      Pekon 318.6118 535.8494   39.83270  214.8476    18.97032
8   Lawksawk 387.1017 630.0035   31.51366  320.5686    21.76677
9  Nawnghkio 349.3359 547.9456   38.44960  323.0201    15.76465
10   Kyaukme 210.9548 601.1773   39.58267  372.4930    30.94709
```

Notice that the final clustering variables list does not include variable INTERNET_PR because it is highly correlated with variable COMPUTER_PR.

Next, we need to change the rows by township name instead of row number by using the code chunk below

```r
row.names(        cluster_vars)        <-        cluster_vars$        "TS.x"
head        (        cluster_vars,10        )
```

```
               TS.x RADIO_PR    TV_PR LLPHONE_PR MPHONE_PR
Mongmit     Mongmit 286.1852 554.1313   35.30618  260.6944
Pindaya     Pindaya 417.4647 505.1300   19.83584  162.3917
Ywangan     Ywangan 484.5215 260.5734   11.93591  120.2856
Pinlaung   Pinlaung 231.6499 541.7189   28.54454  249.4903
Mabein       Mabein 449.4903 708.6423   72.75255  392.6089
Kalaw         Kalaw 280.7624 611.6204   42.06478  408.7951
Pekon         Pekon 318.6118 535.8494   39.83270  214.8476
Lawksawk   Lawksawk 387.1017 630.0035   31.51366  320.5686
Nawnghkio Nawnghkio 349.3359 547.9456   38.44960  323.0201
Kyaukme     Kyaukme 210.9548 601.1773   39.58267  372.4930
          COMPUTER_PR
Mongmit      12.15939
Pindaya      12.88190
Ywangan       4.41465
Pinlaung     13.76255
Mabein       16.45042
Kalaw        29.63160
Pekon        18.97032
Lawksawk     21.76677
Nawnghkio    15.76465
Kyaukme      30.94709
```

Notice that the row number has been replaced into the township name.

Now, we will delete the TS.x field by using the code chunk below.

```
shan_ict <- select(cluster_vars, c(2:6))
head(shan_ict, 10)
```

```
          RADIO_PR    TV_PR LLPHONE_PR MPHONE_PR COMPUTER_PR
Mongmit   286.1852 554.1313   35.30618  260.6944    12.15939
Pindaya   417.4647 505.1300   19.83584  162.3917    12.88190
Ywangan   484.5215 260.5734   11.93591  120.2856     4.41465
Pinlaung  231.6499 541.7189   28.54454  249.4903    13.76255
Mabein    449.4903 708.6423   72.75255  392.6089    16.45042
Kalaw     280.7624 611.6204   42.06478  408.7951    29.63160
Pekon     318.6118 535.8494   39.83270  214.8476    18.97032
Lawksawk  387.1017 630.0035   31.51366  320.5686    21.76677
Nawnghkio 349.3359 547.9456   38.44960  323.0201    15.76465
Kyaukme   210.9548 601.1773   39.58267  372.4930    30.94709
```

Data Standardisation

In general, multiple variables will be used in cluster analysis. It is not unusual their values range are different. In order to avoid the cluster analysis result is baised to clustering variables with large values, it is useful to standardise the input variables before performing cluster analysis.

Min-Max standardisation

In the code chunk below, *normalize()* of *heatmaply* package is used to stadardisation the clustering variables by using Min-Max method. The *summary()* is then used to display the summary statistics of the standardised clustering variables.

```
shan_ict.std <- normalize(shan_ict)
summary(shan_ict.std)
```

```
    RADIO_PR          TV_PR          LLPHONE_PR       MPHONE_PR     
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000  
 1st Qu.:0.2544   1st Qu.:0.4600   1st Qu.:0.1123   1st Qu.:0.2199  
 Median :0.4097   Median :0.5523   Median :0.1948   Median :0.3846  
 Mean   :0.4199   Mean   :0.5416   Mean   :0.2703   Mean   :0.3972  
 3rd Qu.:0.5330   3rd Qu.:0.6750   3rd Qu.:0.3746   3rd Qu.:0.5608  
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  
  COMPUTER_PR     
 Min.   :0.00000  
 1st Qu.:0.09598  
 Median :0.17607  
 Mean   :0.23692  
 3rd Qu.:0.29868  
 Max.   :1.00000  
```

Notice that the values range of the Min-max standardised clustering variables are 0-1 now.

Z-score standardisation

Z-score standardisation can be performed easily by using *scale()* of Base R. The code chunk below will be used to stadardisation the clustering variables by using Z-score method.

```
shan_ict.z <- scale(shan_ict)
describe(shan_ict.z)
```

```
            vars  n mean sd median trimmed  mad   min  max range
RADIO_PR       1 55    0  1  -0.04   -0.06 0.94 -1.85 2.55  4.40
TV_PR          2 55    0  1   0.05    0.04 0.78 -2.47 2.09  4.56
LLPHONE_PR     3 55    0  1  -0.33   -0.15 0.68 -1.19 3.20  4.39
MPHONE_PR      4 55    0  1  -0.05   -0.06 1.01 -1.58 2.40  3.98
COMPUTER_PR    5 55    0  1  -0.26   -0.18 0.64 -1.03 3.31  4.34
             skew kurtosis   se
RADIO_PR     0.48    -0.27 0.13
TV_PR       -0.38    -0.23 0.13
LLPHONE_PR   1.37     1.49 0.13
MPHONE_PR    0.48    -0.34 0.13
COMPUTER_PR  1.80     2.96 0.13
```

Notice the mean and standard deviation of the Z-score standardised clustering variables are 0 and 1 respectively.

Note: *describe()* of **psych** package is used here instead of *summary()* of Base R because the earlier provides standard deviation.

Warning: Z-score standardisation method should only be used if we would assume all variables come from some normal distribution.

Visualising the standardised clustering variables

Beside reviewing the summary statistics of the standardised clustering variables, it is also a good practice to visualise their distribution graphical.

The code chunk below plot the scaled *Radio_PR* field.

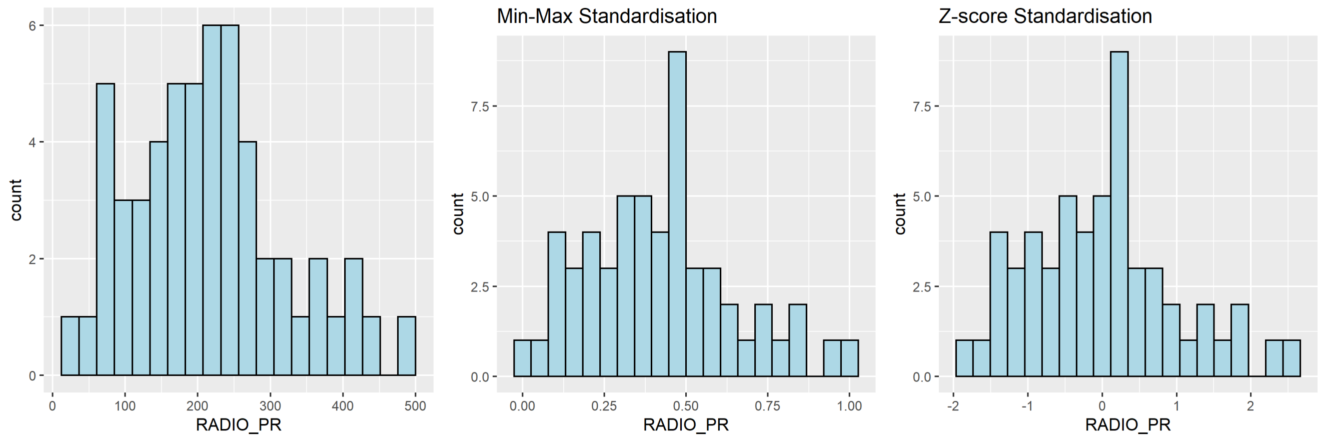
```
r <- ggplot(data=ict_derived,
            aes(x= `RADIO_PR`)) +
  geom_histogram(bins=20,
                 color="black",
                 fill="light blue")

shan_ict_s_df <- as.data.frame(shan_ict.std)
s <- ggplot(data=shan_ict_s_df,
       aes(x=`RADIO_PR`)) +
  geom_histogram(bins=20
```

```
  geom_histogram(bins=20,
                 color="black",
                 fill="light blue") +
  ggtitle("Min-Max Standardisation")

shan_ict_z_df <- as.data.frame(shan_ict.z)
z <- ggplot(data=shan_ict_z_df,
     aes(x= `RADIO_PR`)) +
  geom_histogram(bins=20,
                 color="black",
                 fill="light blue") +
  ggtitle("Z-score Standardisation")

ggarrange(r, s, z,
          ncol = 3,
          nrow = 1)
```



Notice that the overall distribution of the clustering variables will change after the data standardisation. Hence, it is advisible **NOT** to perform data standardisation if the values range of the clustering variables are not very large.

Computing proximity matrix

In R, many packages provide functions to calculate distance matrix. We will compute the proximity matrix by using *dist()* of R.

dist() supports six distance proximity calculations, they are: **euclidean, maximum, manhattan, canberra, binary and minkowski**. The default is *euclidean* proximity matrix.

The code chunk below is used to compute the proximity matrix using *euclidean* method.

```
proxmat <- dist(shan_ict, method = 'euclidean')
```

The code chunk below can then be used to list the content of *proxmat* for visual inspection.

```
proxmat
```

Computing hierarchical clustering

In R, there are several packages provide hierarchical clustering function. In this hands-on exercise, *hclust()* of R stats will be used.

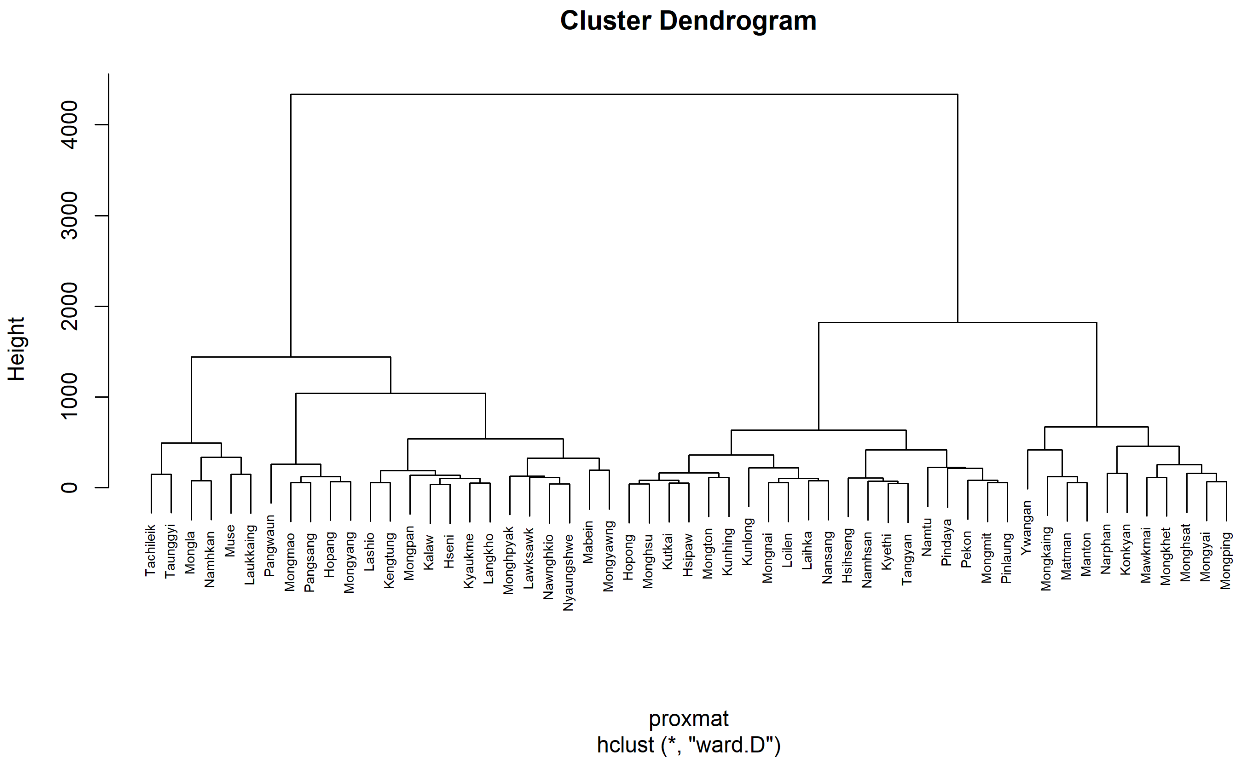
hclust() employed agglomeration method to compute the cluster. Eight clustering algorithms are supported, they are: ward.D, ward.D2, single, complete, average(UPGMA), mcquitty(WPGMA), median(WPGMC) and centroid(UPGMC).

The code chunk below performs hierarchical cluster analysis using ward.D method. The hierarchical clustering output is stored in an object of class **hclust** which describes the tree produced by the clustering process.

```
hclust_ward <- hclust(proxmat, method = 'ward.D')
```

We can then plot the tree by using *plot()* of R Graphics as shown in the code chunk below.

```
plot(hclust_ward, cex = 0.6)
```



Selecting the optimal clustering algorithm

One of the challenge in performing hierarchical clustering is to identify stronger clustering structures. The issue can be solved by using use *agnes()* function of **cluster** package. It functions like *hclus()*, however, with the *agnes()* function you can also get the agglomerative coefficient, which measures the amount of clustering structure found (values closer to 1 suggest strong clustering structure).

The code chunk below will be used to compute the agglomerative coefficients of all hierarchical clustering algorithms.

```
m <- c( "average", "single", "complete", "ward")
names(m) <- c( "average", "single", "complete", "ward")

ac <- function(x) {
  agnes(shan_ict, method = x)$ac
}

map_dbl(m, ac)
```

```
  average    single  complete      ward 
0.8131144 0.6628705 0.8950702 0.9427730 
```

With reference to the output above, we can see that Ward's method provides the strongest clustering structure among the four methods assessed. Hence, in the subsequent analysis, only Ward's method will be used.

Determining Optimal Clusters

Another technical challenge face by data analyst in performing clustering analysis is to determine the optimal clusters to retain.

There are three commonly used methods to determine the optimal clusters, they are:

- Elbow Method
- Average Silhouette Method
- Gap Statistic Method

Gap Statistic Method

The **gap statistic** compares the total within intra-cluster variation for different values of k with their expected values under null reference distribution of the data. The estimate of the optimal clusters will be value that maximize the gap statistic (i.e., that yields the largest gap statistic). This means that the clustering structure is far away from the random uniform distribution of points.

To compute the gap statistic, *clusGap()* of **cluster** package will be used.

```
set.seed(12345)
gap_stat <- clusGap(shan_ict, FUN = hcut, nstart =
```

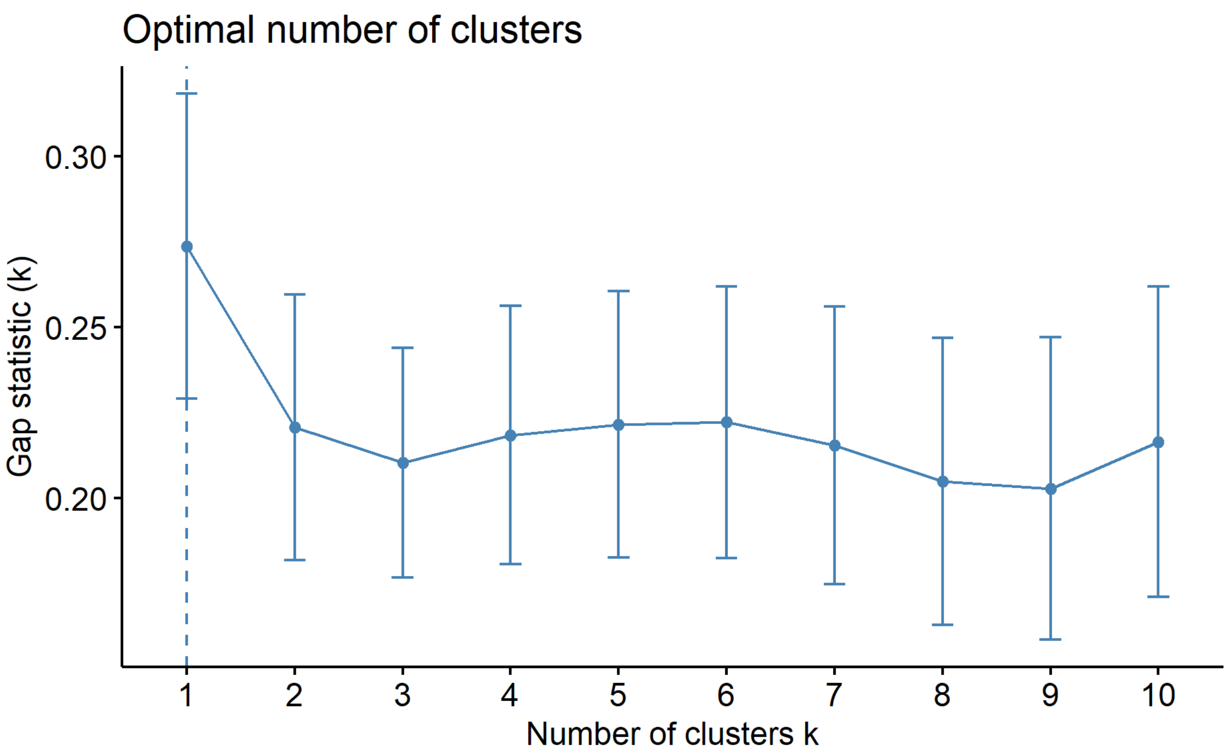
```
gap_stat <- clusGap(shan_ict, FUN = hcut, nstart =
25, K.max = 10, B = 50)
# Print the result
print(gap_stat, method = "firstmax")
```

```
Clustering Gap statistic ["clusGap"] from call:
clusGap(x = shan_ict, FUNcluster = hcut, K.max = 10, B = 50,     nstart = 25)
B=50 simulated reference sets, k = 1..10; spaceH0="scaledPCA"
 --> Number of clusters (method 'firstmax'): 1
          logW   E.logW       gap     SE.sim
 [1,] 8.407129 8.680794 0.2736651 0.04460994
 [2,] 8.130029 8.350712 0.2206824 0.03880130
 [3,] 7.992265 8.202550 0.2102844 0.03362652
 [4,] 7.862224 8.080655 0.2184311 0.03784781
 [5,] 7.756461 7.978022 0.2215615 0.03897071
 [6,] 7.665594 7.887777 0.2221833 0.03973087
 [7,] 7.590919 7.806333 0.2154145 0.04054939
 [8,] 7.526680 7.731619 0.2049390 0.04198644
 [9,] 7.458024 7.660795 0.2027705 0.04421874
[10,] 7.377412 7.593858 0.2164465 0.04540947
```

Also note that the *hcut* function used is from **factoextra** package.

Next, we can visualise the plot by using *fviz_gap_stat()* of **factoextra** package.

```
fviz_gap_stat(gap_stat)
```



With reference to the gap statistic graph above, the recommended number of cluster to retain is 1. However, it is not logical to retain only one cluster. By examine the gap statistic graph, the 6-cluster gives the largest gap statistic and should be the next best cluster to pick.

Note: In addition to these commonly used approaches, the NbClust package, published by Charrad et al., 2014, provides 30 indices for determining the relevant number of clusters and proposes to users the best clustering scheme from the different results obtained by varying all combinations of number of clusters, distance measures, and clustering methods.

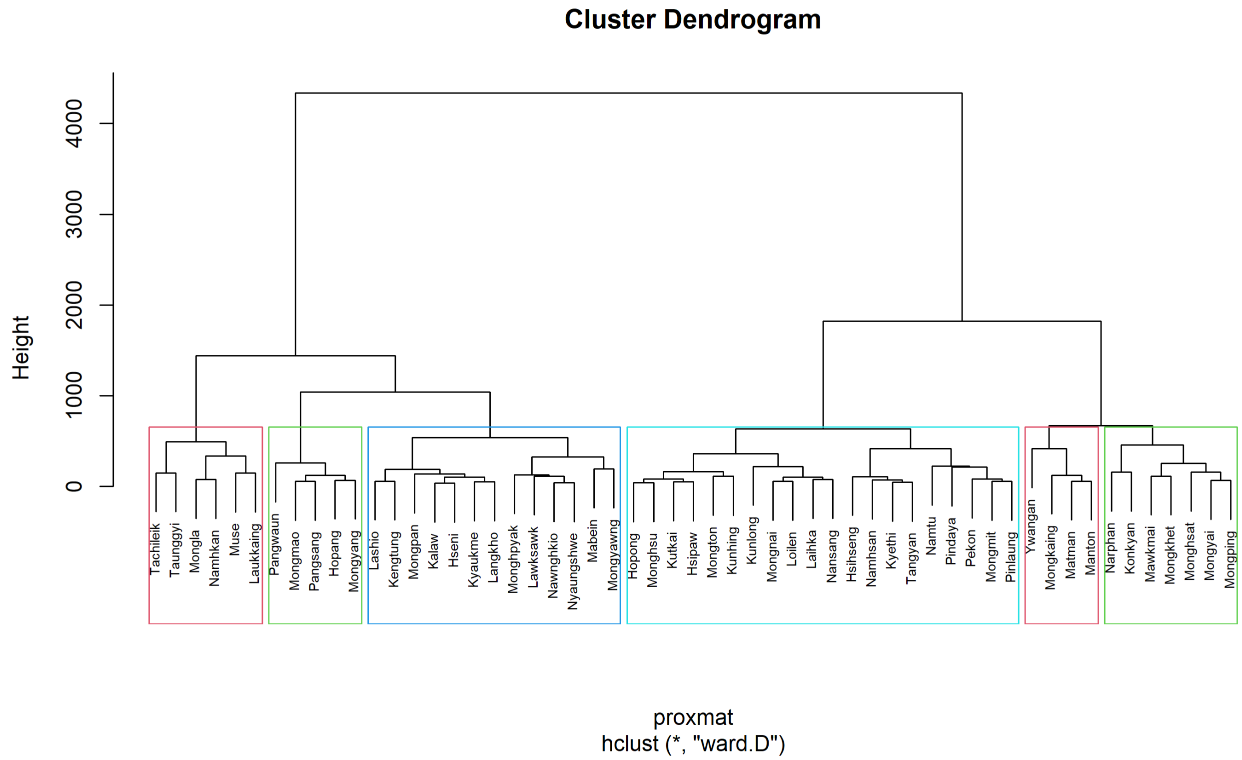
Interpreting the dendrograms

In the dendrogram displayed above, each leaf corresponds to one observation. As we move up the tree, observations that are similar to each other are combined into branches, which are themselves fused at a higher height.

The height of the fusion, provided on the vertical axis, indicates the (dis)similarity between two observations. The higher the height of the fusion, the less similar the observations are. Note that, conclusions about the proximity of two observations can be drawn only based on the height where branches containing those two observations first are fused. We cannot use the proximity of two observations along the horizontal axis as a criteria of their similarity.

It's also possible to draw the dendrogram with a border around the selected clusters by using *rect.hclust()* of R stats. The argument *border* is used to specify the border colors for the rectangles.

```
plot(hclust_ward, cex = 0.6)
rect.hclust(hclust_ward, k = 6, border = 2:5)
```



Visually-driven hierarchical clustering analysis

Visually-driven hierarchical clustering analysis

In this section, we will learn how to perform visually-driven hiearchical clustering analysis by using *heatmaply* package.

With **heatmaply**, we are able to build both highly interactive cluster heatmap or static cluster heatmap.

Transforming the data frame into a matrix

The data was loaded into a data frame, but it has to be a data matrix to make your heatmap.

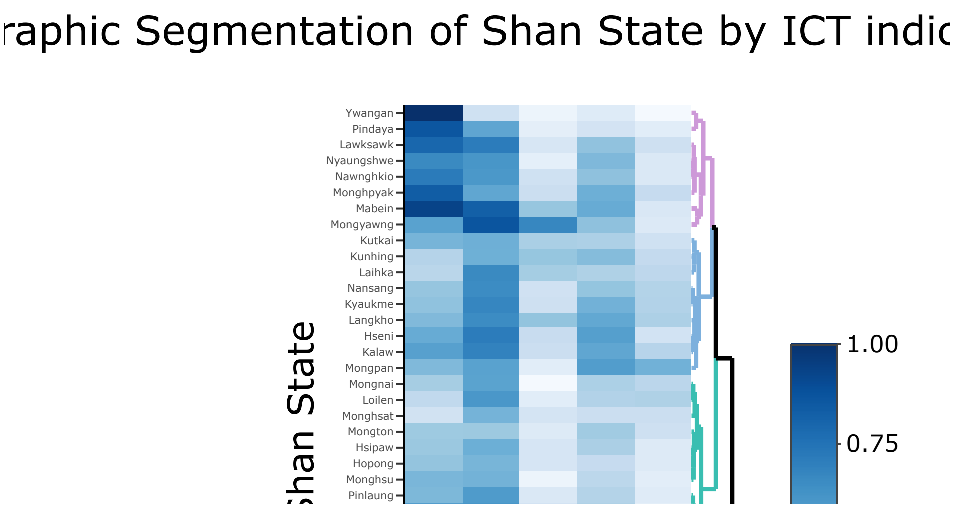
The code chunk below will be used to transform *shan_ict* data frame into a data matrix.

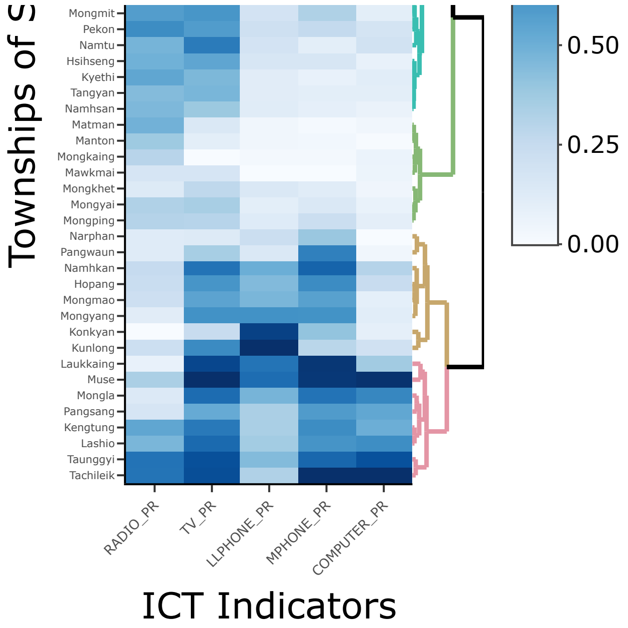
```
shan_ict_mat <- data.matrix(shan_ict)
```

Plotting interactive cluster heatmap using ***heatmaply()***

In the code chunk below, the *heatmaply()* is used to build an interactive cluster heatmap.

```
heatmaply(normalize(shan_ict_mat),
          Colv=NA,
          dist_method = "euclidean",
          hclust_method = "ward.D",
          seriate = "OLO",
          colors = Blues,
          k_row = 6,
          margins = c(NA,200,60,NA),
          fontsize_row = 4,
          fontsize_col = 5,
          main="Geographic Segmentation of Shan State by ICT indicators",
          xlab = "ICT Indicators",
          ylab = "Townships of Shan State"
          )
```





Mapping the clusters formed

With closed examination of the dendragram above, we have decided to retain five clusters.

cutree() of R Base will be used in the code chunk below to derive a 5-cluster model.

```
groups <- as.factor(cutree(hclust_ward, k=6))
```

The output is called *groups*. It is a *list* object.

In order to visualise the clusters, the *groups* object need to be appended onto *shan_sf* simple feature object.

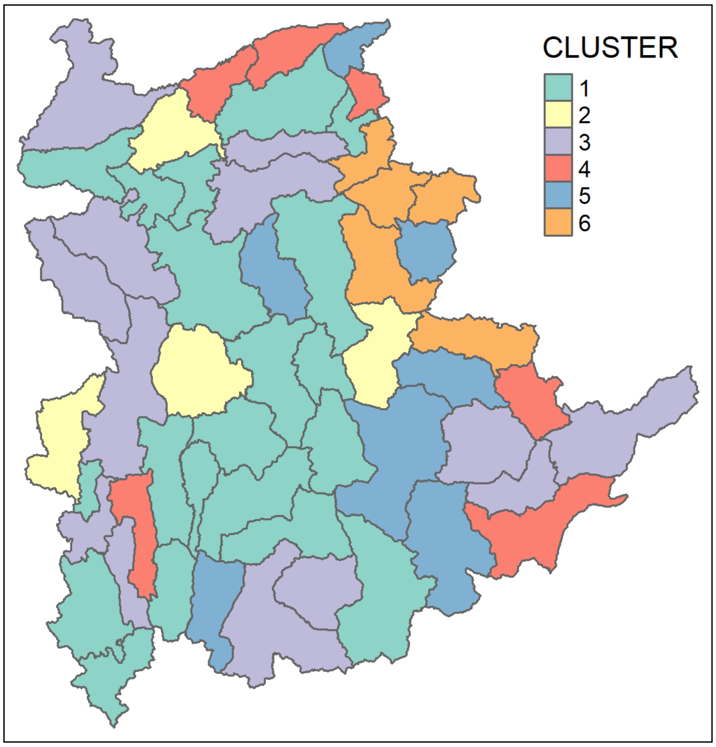
The code chunk below form the join in three steps:

- the *groups* list object will be converted into a matrix;
- *cbind()* is used to append *groups* matrix onto shan_sf to produce an output simple feature object called shan_sf_cluster; and
- *rename* of **dplyr** package is used to rename *as.matrix.groups* field as *CLUSTER*.

```
shan_sf_cluster <- cbind(shan_sf, as.matrix(groups)) %>%
  rename(`CLUSTER`=`as.matrix.groups.`)
```

Next, *qtm()* of **tmap** package is used to plot the choropleth map showing the cluster formed.

```
qtm(shan_sf_cluster, "CLUSTER")
```



The choropleth map above reveals the clusters are very fragmented. The is one of the major limitation when non-spatial clustering algorithm such as hierarchical cluster analysis method is used.

Spatially Constrained Clustering - SKATER approach

In this section, you will learn how to derive spatially constrained cluster by using **SKATER** method.

Converting into SpatialPolygonsDataFrame

First, we need to convert *shan_sf* into SpatialPolygonDataFrame. This is because SKATER function only support **sp** objects such as SpatialPolygonDataFrame.

The code chunk below uses *as_Spatial()* of **sf** package to convert *shan_sf* into a SpatialPolygonDataFrame called *shan_sp*.

```
shan_sp <- as_Spatial(shan_sf)
```

Computing Neighbour List

Next, poly2nd() of **spdep** package will be used to compute the neighbours list from polygon list.

```
shan.nb <- poly2nb(shan_sp)
```

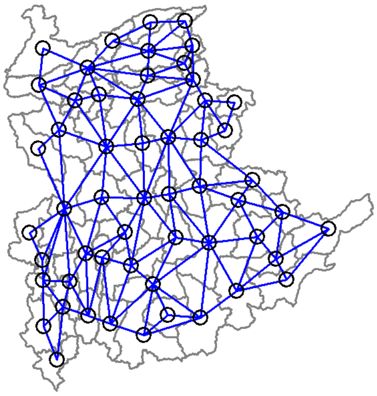
```
summary(shan.nb)
```

```
Neighbour list object:
Number of regions: 55
Number of nonzero links: 264
Percentage nonzero weights: 8.727273
Average number of links: 4.8
Link number distribution:

 2  3  4  5  6  7  8  9
 5  9  7 21  4  3  5  1
5 least connected regions:
3 5 7 9 47 with 2 links
1 most connected region:
8 with 9 links
```

We can plot the neighbours list on shan_sp by using the code chunk below. Since we now can plot the community area boundaries as well, we plot this graph on top of the map. The first plot command gives the boundaries. This is followed by the plot of the neighbor list object, with coordinates applied to the original SpatialPolygonDataFrame (Shan state township boundaries) to extract the centroids of the polygons. These are used as the nodes for the graph representation. We also set the color to blue and specify add=TRUE to plot the network on top of the boundaries.

```
plot(shan_sp, border=grey(.5))
plot(shan.nb, coordinates(shan_sp), col="blue", add=TRUE)
```



Note that if you plot the network first and then the boundaries, some of the areas will be clipped. This is because the plotting area is determined by the characteristics of the first plot. In this example, because the boundary map extends further than the graph, we plot it first.

Computing minimum spanning tree

Calculating edge costs

Next, *nbcosts()* of **spdep** package is used to compute the cost of each edge. It is the distance between it nodes. This function compute this distance using a data.frame with observations vector in each node.

The code chunk below is used to compute the cost of each edge.

```
lcosts <- nbcosts(shan.nb, shan_ict)
```

For each observation, this gives the pairwise dissimilarity between its values on the five variables and the values for the neighbouring observation (from the neighbour list). Basically, this is the notion of a generalised weight for a spatial weights matrix.

Next, We will incorporate these costs into a weights object in the same way as we did in the calculation of inverse of distance weights. In other words, we convert the neighbour list to a list weights object by specifying the just computed ***lcosts*** as the weights.

In order to achieve this, *nb2listw()* of **spdep** package is used as shown in the code chunk below.

Note that we specify the *style* as **B** to make sure the cost values are not row-standardised.

```
shan.w <- nb2listw(shan.nb, lcosts, style="B")
summary(shan.w)
```

```
Characteristics of weights list object:
Neighbour list object:
Number of regions: 55
Number of nonzero links: 264
Percentage nonzero weights: 8.727273
Average number of links: 4.8
Link number distribution:

 2  3  4  5  6  7  8  9
 5  9  7 21  4  3  5  1
5 least connected regions:
3 5 7 9 47 with 2 links
1 most connected region:
8 with 9 links

Weights style: B
Weights constants summary:
   n   nn       S0       S1        S2
B 55 3025 76267.65 58260785 522016004
```

Computing minimum spanning tree

The minimum spanning tree is computed by mean of the *mstree()* of **spdep** package as shown in the code chunk below.

```
shan.mst <- mstree(shan.w)
```

After computing the MST, we can check its class and dimension by using the code chunk below.

```
class(shan.mst)
```

```
[1] "mst"    "matrix"
```

```
dim(shan.mst)
```

```
[1] 54  3
```

Note that the dimension is 54 and not 55. This is because the minimum spanning tree consists on n-1 edges (links) in order to traverse all the nodes.

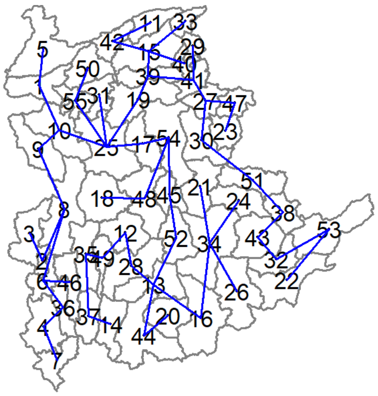
We can display the content of *shan.mst* by using *head()* as shown in the code chunk below.

```
head(shan.mst)
```

```
     [,1] [,2]      [,3]
[1,]   31   25 229.44658
[2,]   25   10 163.95741
[3,]   10    1 144.02475
[4,]   10    9 157.04230
[5,]    9    8  90.82891
[6,]    8    6 140.01101
```

The plot method for the MST include a way to show the observation numbers of the nodes in addition to the edge. As before, we plot this together with the township boundaries. We can see how the initial neighbour list is simplified to just one edge connecting each of the nodes, while passing through all the nodes.

```
plot(shan_sp, border=gray(.5))
plot.mst(shan.mst, coordinates(shan_sp),
     col="blue", cex.lab=0.7, cex.circles=0.005, add
= TRUE)
```



Computing spatially constrained clusters using SKATER method

The code chunk below compute the spatially constrained cluster using *skater()* of **spdep** package.

```
clust6 <- skater(shan.mst[,1:2], shan_ict, method = "euclidean", 5)
```

The *skater()* takes three mandatory arguments: - the first two columns of the MST matrix (i.e. not the cost), - the data matrix (to update the costs as units are being grouped), and - the number of cuts. Note: It is set to **one less than the number of clusters**. So, the value specified is **not** the number of clusters, but the number of cuts in the graph, one less than the number of clusters.

The result of the *skater()* is an object of class **skater**. We can examine its contents by using the code chunk below.

```
str(clust6)
```

```
List of 8
 $ groups      : num [1:55] 3 3 6 3 3 3 3 3 3 3 ...
 $ edges.groups:List of 6
  ..$ :List of 3
  .. ..$ node: num [1:22] 13 48 54 55 45 37 34 16 25 31 ...
  .. ..$ edge: num [1:21, 1:3] 48 55 54 37 34 16 45 31 13 13 ...
  .. ..$ ssw : num 3423
  ..$ :List of 3
  .. ..$ node: num [1:18] 47 27 53 38 42 15 41 51 43 32 ...
  .. ..$ edge: num [1:17, 1:3] 53 15 42 38 41 51 15 27 15 43 ...
  .. ..$ ssw : num 3759
  ..$ :List of 3
  .. ..$ node: num [1:11] 2 6 8 1 36 4 10 9 46 5 ...
```

```
 .. ..$ edge: num [1:10, 1:3] 6 1 8 36 4 6 8 10 10 9 ...
 .. ..$ ssw : num 1458
 ..$ :List of 3
 .. ..$ node: num [1:2] 44 20
 .. ..$ edge: num [1, 1:3] 44 20 95
 .. ..$ ssw : num 95
 ..$ :List of 3
 .. ..$ node: num 23
 .. ..$ edge: num[0 , 1:3]
 .. ..$ ssw : num 0
 ..$ :List of 3
 .. ..$ node: num 3

 .. ..$ edge: num[0 , 1:3]
 .. ..$ ssw : num 0
$ not.prune   : NULL
$ candidates  : int [1:6] 1 2 3 4 5 6
$ ssto        : num 12613
$ ssw         : num [1:6] 12613 10977 9962 9540 9123 ...
$ crit        : num [1:2] 1 Inf
$ vec.crit    : num [1:55] 1 1 1 1 1 1 1 1 1 1 ...
- attr(*, "class")= chr "skater"
```

The most interesting component of this list structure is the groups vector containing the labels of the cluster to which each observation belongs (as before, the label itself is arbitary). This is followed by a detailed summary for each of the clusters in the edges.groups list. Sum of squares measures are given as ssto for the total and ssw to show the effect of each of the cuts on the overall criterion.

We can check the cluster assignment by using the conde chunk below.

```
ccs6 <- clust6$groups
ccs6
```

```
 [1] 3 3 6 3 3 3 3 3 3 3 2 1 1 1 2 1 1 1 2 4 1 2 5 1 1 1 2 1 2 2 1 2 2
[34] 1 1 3 1 2 2 2 2 2 2 4 1 3 2 1 1 1 2 1 2 1 1
```

We can find out how many observations are in each cluster by means of the table command. Parenthetially, we can also find this as the dimension of each vector in the lists contained in edges.groups. For example, the first list has node with dimension 12, which is also the number of observations in the first cluster.

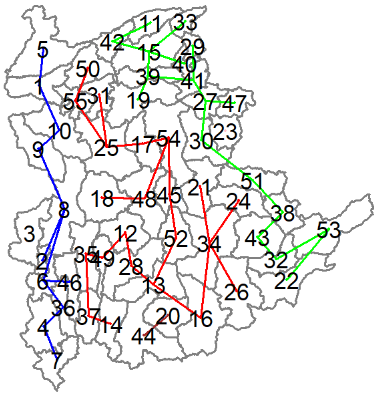
```
table(ccs6)
```

```
ccs6
 1  2  3  4  5  6 
22 18 11  2  1  1 
```

Lastly, we can also plot the pruned tree that shows the five clusters on top of the townshop area.

```
plot(shan_sp, border=gray(.5))
```

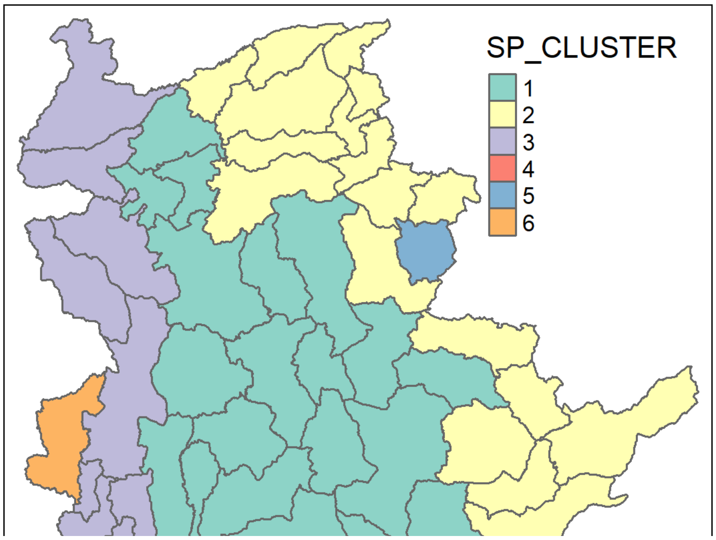
```
plot(clust6, coordinates(shan_sp), cex.lab=.7,
     groups.colors=c("red","green","blue", "brown", "pink"), cex.circles=0.005, add=TRUE)
```

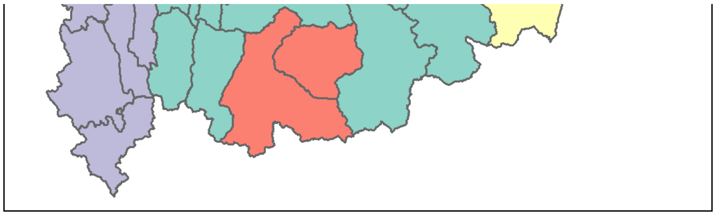


Visualising the clusters in choropleth map

The code chunk below is used to plot the newly derived clusters by using SKATER method.

```
groups_mat <- as.matrix(clust6$groups)
shan_sf_spatialcluster <- cbind(shan_sf_cluster, as.factor(groups_mat)) %>%
  rename(`SP_CLUSTER`=`as.factor.groups_mat.`)
qtm(shan_sf_spatialcluster, "SP_CLUSTER")
```





For easy comparison, it will be better to place both the hierarchical clustering and spatially constrained hierarchical clustering maps next to each other.

```
hclust.map <- qtm(shan_sf_cluster,
                  "CLUSTER") +
  tm_borders(alpha = 0.5)

shclust.map <- qtm(shan_sf_spatialcluster,
                   "SP_CLUSTER") +
  tm_borders(alpha = 0.5)

tmap_arrange(hclust.map, shclust.map,
             asp=NA, ncol=2)
```