Introduction to geographic chart in R

Zihao Liu

UNI: zl2986

1.1 Basic conceptions

Our data in the real world often contains geographic location information, so it is inevitable to draw maps involving geographic coordinate systems. The geographic coordinate system uses a three-dimensional sphere to define the position of the earth's surface, so as to realize a coordinate system that references points on the earth's surface through latitude and longitude.

Spatial data refers to data defined in a three-dimensional space with geographic location information. Map projection is a particularly important key technology. The most basic step of map information visualization is map projection, which converts the geographic coordinate information on the non-expandable surface to a two-dimensional plane, which is equivalent to surface parameterization.

Projection can be divided into many types. In practical applications, we should choose the projection method that best meets the target according to different needs. Among them, the most commonly used projection methods are：

- Azimuthal or Zenithal projection
- Spherical or equirectangular projection
- Orthographic projection
- Lambert's equal-area meridional map projection
- Albers projection

1.2 Choropleth map

A **choropleth map** is a type of thematic map in which a set of pre-defined areas is colored or patterned in proportion to a statistical variable that represents an aggregate summary of a geographic characteristic within each area, such as population density or per-capita income. Choropleth maps provide an easy way to visualize how a variable varies across a geographic area or show the level of variability within a region.

The final element of a choropleth map is the set of colors used to represent the different values of the variable.The most common types of color progressions used in choropleth (and other thematic) maps include:

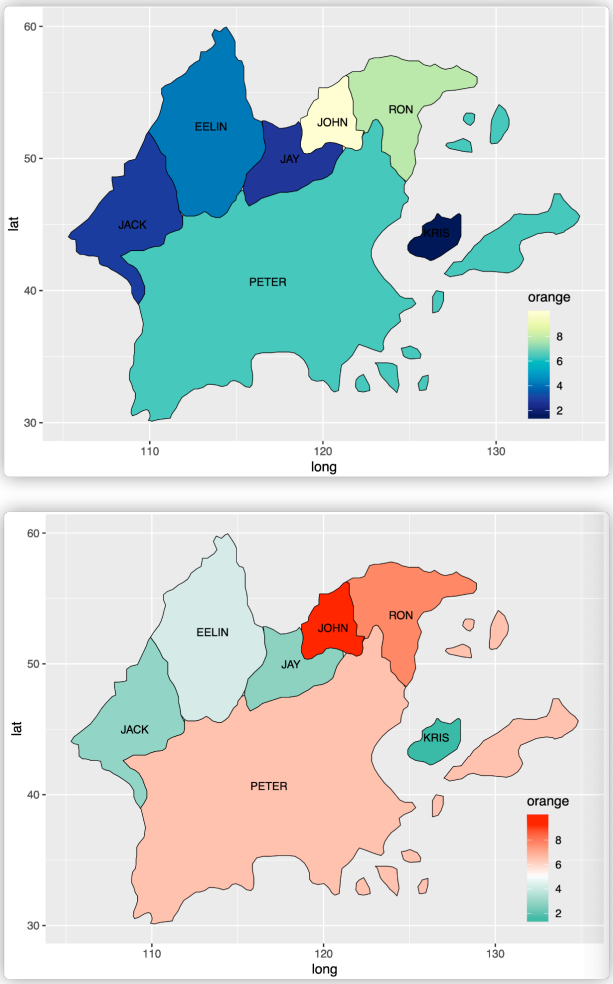
- Single hue progression
- Bi-polar color progression
- Full-spectral color progression

The biggest problem with choropleth maps is the asymmetry of data distribution and geographic area size. Usually a large amount of data is concentrated in densely populated areas, while sparsely populated areas occupies most of the screen space. The practice of using a large amount of screen space to represent a small part of data is very uneconomical in the use of space. This asymmetry often results It causes users to misunderstand the data and cannot help users accurately distinguish and compare the data values of each area on the map. Therefore, sometimes other geospatial charts can be used to represent regional data more reasonably, such as hexagonal maps.

Making choropleth maps

The key to drawing a Choropleth map is to map the values of different regions to the colors of different regions or polygons. Therefore, the map database and the data frame containing the countries and their values must be joined. This can be done using the left-Join() function of the dplyr package. Based on their shared columns 'country', use ggplot's geom_polygon() to draw areas of different colors.

```
library(rgdal)   #for readOGR()
library(ggplot2)
library(dplyr)
library(RColorBrewer)
-------------------------
dataProjected <- readOGR("Virtual_Map1.shp")
dataProjected@data$id <- rownames(dataProjected@data)
watershedPoints <- fortify(dataProjected)
watershedDF <- full_join(watershedPoints, dataProjected@data,by='id')
dataProjected@data$id <- rownames(dataProjected@data)
watershedPoints <- fortify(dataProjected)
df_Map <- full_join(watershedPoints, dataProjected@data, by = "id")
df_city<-read.csv("Virtual_City.csv")
df <- left_join(df_Map, df_city[c('country','orange')], by = "country")
#(a) Single hue progression
ggplot()+
  geom_polygon(data=df, aes(x=long, y=lat,
group=group,fill=orange),colour="black",size=0.25)+
  geom_text(data=df_city,aes(x=long, y=lat,
label=country),colour="black",size=3)+
  scale_fill_gradientn(colours = rev(brewer.pal(9,'YlGnBu')))+
  theme(legend.position = c(0.9,0.2),
        legend.background = element_blank())
#(b) Bi-polar color progression
ggplot()+
  geom_polygon(data=df, aes(x=long, y=lat,
group=group,fill=orange),colour="black",size=0.25)+
  geom_text(data=df_city,aes(x=long, y=lat,
label=country),colour="black",size=3)+
  scale_fill_gradient2(low="#00A08A",mid="white",high="#FF0000",
                       midpoint = mean(df_city$orange))+
  theme(legend.position = c(0.9,0.2),
        legend.background = element_blank())
```



(a)Single hue progression (b) Bi-polar color progression

1.3 Dot map

Dot map, also called dot distribution map or dot density map, is a method using dots with same size to illustrate the data distribution on a geographic map. The items on the dot map contains longitude and latitude rather than size information. Dot map is an ideal way to observe the geographic distribution. However, when it comes to the specific items and its data, dot map is not proper.

There are two common types:

- **One-to-one**: one dot only represents one item because the location of the dot only has one data source to make sure the dot is located at the correct place.
- **One-to-many**: one dot represents a special unit, not the specific location because the dot here is collection data which put on the map randomly.

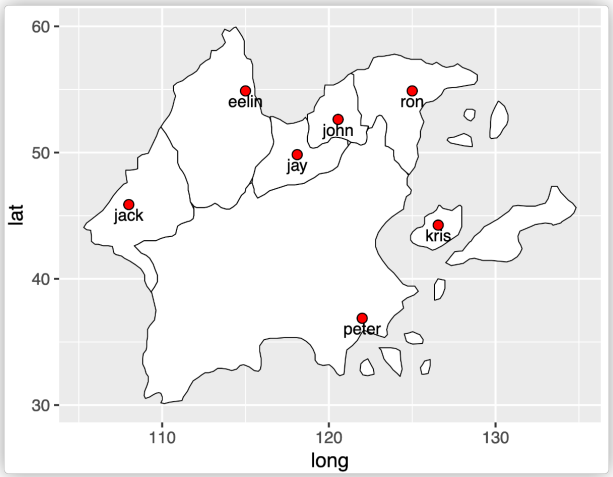
Making dot maps

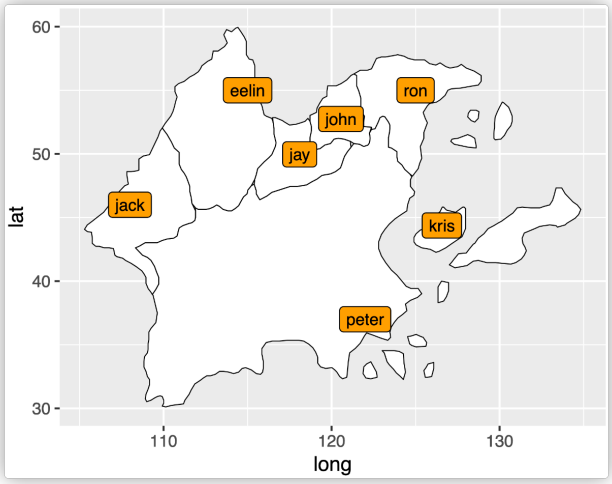
Dot map is a combination of dots and a map. The kay is to translate the location data (x, y) to longitude and latitude data (long, lat). **Figure a** illustrates a basic dot map with dots and texts. Firstly, make map layer by using **geom_polygon()** function, then use **geom_point()** function to make dots, and then use **geom_text** function to add texts. **Figure b** employs text boxes on a dot map by using **df_map** and **df_city**.

Bubble map is a combination of dot map and bubbles. It requires data (lat, long, value) to make bubbles on a map. In this method, longitude and latitude can show the geographic location and value stands for the size of bubbles. Sometimes we can distinguish the data with different colors by using another dimension-catergory. However, we need to pay more attention on the bubbles when the bubbles are too many or too large, otherwise the bubbles will overlap.

Similar to dot map, bubble map only adds new variables to illustrate size and color. **Figure c** is a common bubble map, while **Figure d** uses different color to distinguish data and therefore get a more straightforward plot.

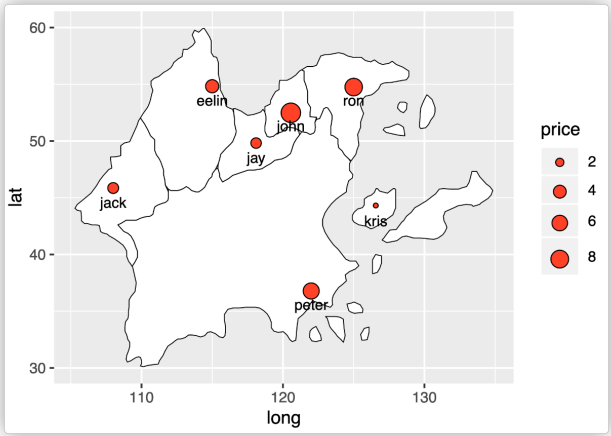
```
library(rgdal)   #readOGR()
library(ggplot2)
library(dplyr)
library(RColorBrewer)
dataProjected <- readOGR("Virtual_Map1.shp")
dataProjected@data$id <- rownames(dataProjected@data)
watershedPoints <- fortify(dataProjected)
df_Map <- full_join(watershedPoints, dataProjected@data, by = "id")
df_city<-read.csv("Virtual_City.csv")
#(a)  (point) + (text)
ggplot()+
  geom_polygon(data=df_Map, aes(x=long, y=lat,
group=group),fill='white',colour="black",size=0.25)+
  geom_point(data=df_city,aes(x=long,
y=lat),shape=21,fill='red',colour="black",size=4)+
  geom_text(data=df_city,aes(x=long, y=lat,
label=city),vjust=1.5,colour="black",size=3)
#(b)  (label)
ggplot()+
  geom_polygon(data=df_Map, aes(x=long, y=lat,
group=group),fill='white',colour="black",size=0.25)+
  #geom_point(data=df_city,aes(x=long,
y=lat),shape=21,fill='red',colour="black",size=4)+
  geom_label(data=df_city,aes(x=long, y=lat,
label=city),fill='orange',colour="black",size=3)
```

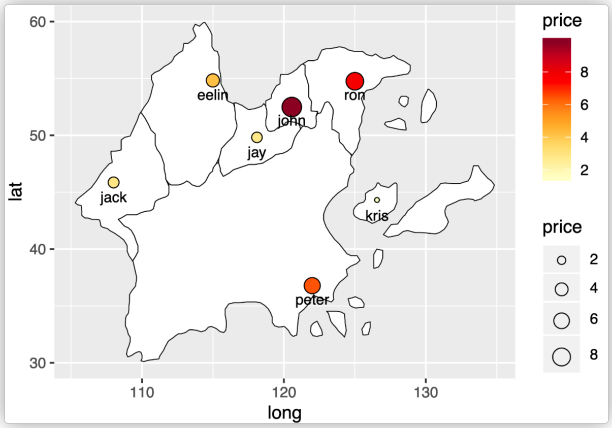




(a) point + text (b) label

```
dataProjected <- readOGR("Virtual_Map1.shp")
dataProjected@data$id <- rownames(dataProjected@data)
watershedPoints <- fortify(dataProjected)
watershedDF <- full_join(watershedPoints, dataProjected@data,by='id')
#(c) represent by size
ggplot()+
  geom_polygon(data=df_Map, aes(x=long, y=lat,
group=group),fill='white',colour="black",size=0.25)+
  geom_point(data=df_city,aes(x=long,
y=lat,size=orange),shape=21,fill='#EF5439',colour="black")+
  geom_text(data=df_city,aes(x=long, y=lat,
label=city),vjust=2,colour="black",size=3)+
  scale_size(range=c(2,9),name='price')
#(d) represent by size and color of bubble
ggplot()+
  geom_polygon(data=df_Map, aes(x=long, y=lat,
group=group),fill='white',colour="black",size=0.25)+
  geom_point(data=df_city,aes(x=long,
y=lat,size=orange,fill=orange),shape=21,colour="black")+
  geom_text(data=df_city,aes(x=long, y=lat,
label=city),vjust=2,colour="black",size=3)+
  scale_size(range=c(2,9),name='price')+
  scale_fill_gradientn(colours = brewer.pal(9,'YlOrRd'),name='price')
```





(c) bubble size (d) bubble size and color

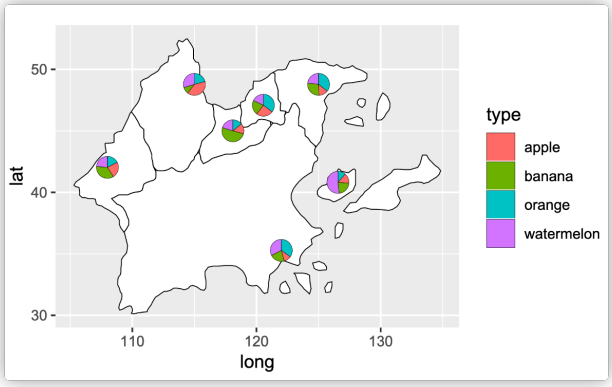
1.4 Pie map

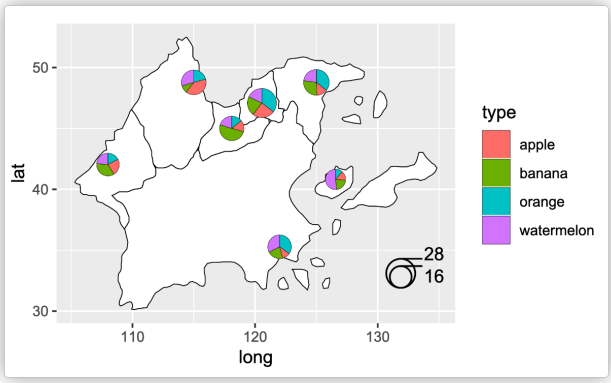
Pie map is a combination of pie chart and map. By using pie map, we can easily tell the proportion of different values in a given area. And different colors can also help distinguish different categories.

Both **geom_scatterpi**e function from scatterpie package and **geom_arc_bar()** function from ggforce package can be used to draw pie map, which requires the coordinate system as coord_fixed() type to make sure that x-axis and y-axis have the same minimized unit. **Figure a** and **Figure b** are typical pie maps while Figure b employs bubble map to show the size of each value at the same time.

```
library(rgdal)   #readOGR()
library(ggplot2)
library(dplyr)
library(RColorBrewer)
library(reshape2)
library(scatterpie)
#library(wesanderson)
dataProjected <- readOGR("Virtual_Map1.shp")
dataProjected@data$id <- rownames(dataProjected@data)
watershedPoints <- fortify(dataProjected)
watershedDF <- full_join(watershedPoints, dataProjected@data,by='id')
dataProjected@data$id <- rownames(dataProjected@data)
watershedPoints <- fortify(dataProjected)
df_Map <- full_join(watershedPoints, dataProjected@data, by = "id")
df_city<-read.csv("Virtual_City.csv")
```

```
Map_Scale<-0.75
min_lat<-min(df_Map$lat) #30.11628
max_lat<-max(df_Map$lat) #59.94186
min_long<-min(df_Map$long) #105.2945
max_long<-max(df_Map$long) #134.8317
df_Map$x<-(df_Map$long-min_long)/(max_long-min_long)
df_Map$y<-(df_Map$lat-min_lat)/(max_lat-min_lat)*Map_Scale
df_city$x<-(df_city$long-min_long)/(max_long-min_long)
df_city$y<-(df_city$lat-min_lat)/(max_lat-min_lat)*Map_Scale
labels_x<-seq(110,135,10)
breaks_x<-(labels_x-min_long)/(max_long-min_long)
labels_y<-seq(30,60,10)
breaks_y<-(labels_y-min_lat)/(max_lat-min_lat)
df_city$Sumindex<-rowSums(df_city[,c("orange","apple","banana","watermelon")])
Bubble_Scale<-0.04
radius<-sqrt(df_city$Sumindex/pi)
Max_radius<-max(radius)
df_city$radius<-radius/Max_radius*Bubble_Scale
#(a) pie map with scatter
ggplot()+
  geom_polygon(data=df_Map, aes(x=x, y=y,
group=group),fill='white',colour="black",size=0.25)+
  geom_scatterpie(data=df_city,aes(x=x, y=y, group=city,r=0.03),# 0.03
                  cols=c("orange","apple","banana","watermelon"),
color="black", alpha=1,size=0.1)+
  scale_x_continuous(breaks=breaks_x,labels=labels_x)+
  scale_y_continuous(breaks=breaks_y,labels=labels_y)+
  xlab("long") +
  ylab("lat") +
  coord_fixed()
#(b) pie map with complex bubble
ggplot()+
  geom_polygon(data=df_Map, aes(x=x, y=y,
group=group),fill='white',colour="black",size=0.25)+
  geom_scatterpie(data=df_city,aes(x=x, y=y, group=city,r=radius),# 0.03
                   cols=c("orange","apple","banana","watermelon"),
color="black", alpha=1,size=0.1)+
  geom_scatterpie_legend(df_city$radius, x=0.9, y=0.1,
                         n=5,
                        labeller=function(x) round((x*
Max_radius/Bubble_Scale)^2*pi))+
  scale_x_continuous(breaks=breaks_x,labels=labels_x)+
  scale_y_continuous(breaks=breaks_y,labels=labels_y)+
  xlab("long") +
  ylab("lat") +
  coord_fixed()
```





(a)pie map with scatter (b) pie map with bubbles

1.5 bar map

Bar map is the overlap of two layers: map and bar chart. The height of bars represents values. And it can also be divided by colors (**Figure a**). **Figure b** employs coxcomb in its bars.

To draw rectangle, we can use **geom_rect()** function from ggplot2 package. So we only need to set the value of coordinates of rectangles: (xmin, ymin) and (xmax, ymax), then use geom_rect() function to realize the bars.

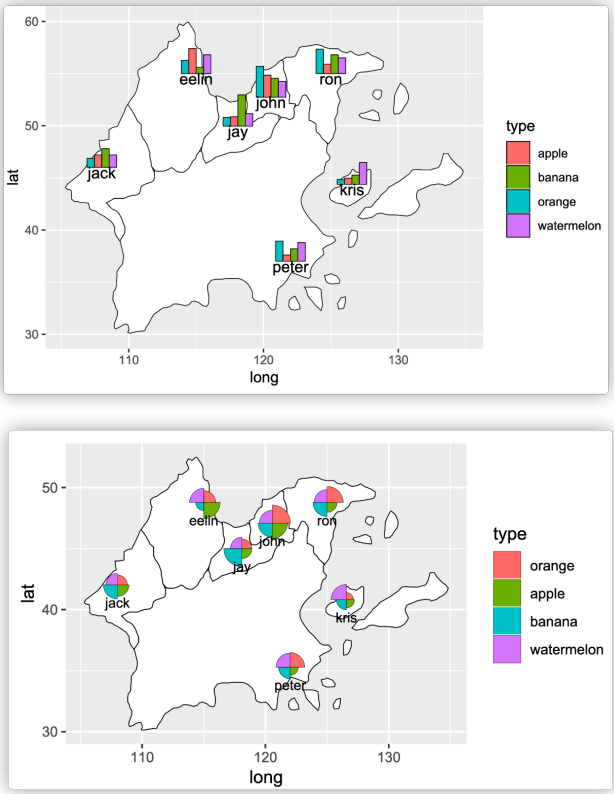
```
ibrary(rgdal)   #readOGR()
library(ggplot2)
library(dplyr)
library(RColorBrewer)
dataProjected <- readOGR("Virtual_Map1.shp")
dataProjected@data$id <- rownames(dataProjected@data)
watershedPoints <- fortify(dataProjected)
watershedDF <- full_join(watershedPoints, dataProjected@data,by='id')
dataProjected@data$id <- rownames(dataProjected@data)
watershedPoints <- fortify(dataProjected)
df_Map <- full_join(watershedPoints, dataProjected@data, by = "id")
df_city<-read.csv("Virtual_City.csv")
selectCol<-c("orange","apple","banana","watermelon")
MaxH<-max(df_city[,selectCol])
Scale<-3
width<-1.1
df_city[,selectCol]<-df_city[,selectCol]/MaxH*Scale
```

```
df_city<-melt(df_city[c('lat','long','group','city',selectCol)],
              id.vars=c('lat','long','group','city'))
df_city<-transform(df_city,hjust1=ifelse(variable=='orange',-width,
                                         ifelse(variable=='apple',-width/2,

ifelse(variable=='banana',0,width/2))),
                           hjust2=ifelse(variable=='orange',-width/2,
                                  ifelse(variable=='apple',0,

 ifelse(variable=='banana',width/2,width))))
Lengend_data<-data.frame(X=rep(132,5),Y=rep(32,5),index=seq(0,MaxH,MaxH/4))
#(a) bar map
ggplot() +
  geom_polygon(data=df_Map, aes(x = long, y = lat,group=group),
               fill="white",colour="black",size=0.25)+
  geom_rect(data = df_city, aes(xmin = long +hjust1, xmax = long+hjust2,
                                ymin = lat, ymax = lat + value ,
                                fill= variable),
            size =0.25, colour ="black", alpha = 1)+
  geom_text(data=df_city[!duplicated(df_city$city),],aes(x=long,y=lat-
0.5,label=city),size=4)+
  labs(fill='type')+
  geom_rect(data = Lengend_data,aes(xmin = X , xmax = X+0.5 ,
                                    ymin = Y, ymax = Y+index / MaxH * Scale),
            size =0.25, colour ="black",fill = NA,alpha = 1)+
  geom_text(data = Lengend_data,aes(x=X+1.5,y=  Y+index / MaxH * Scale,
                                    label=index),size=3)
#(b) coxcomb map
Map_Scale<-0.75
min_lat<-min(df_Map$lat) #30.11628
max_lat<-max(df_Map$lat) #59.94186
min_long<-min(df_Map$long) #105.2945
max_long<-max(df_Map$long) #134.8317
df_Map$x<-(df_Map$long-min_long)/(max_long-min_long)
df_Map$y<-(df_Map$lat-min_lat)/(max_lat-min_lat)*Map_Scale
df_city$x<-(df_city$long-min_long)/(max_long-min_long)
df_city$y<-(df_city$lat-min_lat)/(max_lat-min_lat)*Map_Scale
labels_x<-seq(110,135,10)
breaks_x<-(labels_x-min_long)/(max_long-min_long)
labels_y<-seq(30,60,10)
breaks_y<-(labels_y-min_lat)/(max_lat-min_lat)
r <- 0.05
scale <- r/max(sqrt(df_city$value))
df_city$start<-rep(c(0,90,180,270)/180*pi,each=nrow(df_city)/4)
df_city$end<-df_city$start+pi/2
ggplot() +
  geom_polygon(data=df_Map, aes(x = x, y = y,group=group),
               fill="white",colour="black",size=0.25)+
  geom_arc_bar(data=df_city,aes(x0=x,y0=y,r0=0,r=sqrt(value)*scale,
```

```
                    start=start,
                    end=end,
                    fill=variable),size=0.1)+
  geom_text(data=df_city[!duplicated(df_city$city),],aes(x=x,y=y,label=city),vju
st=2.3,size=3)+
    labs(x='long',y='lat',fill='type')+
    scale_x_continuous(breaks=breaks_x,labels=labels_x)+
    scale_y_continuous(breaks=breaks_y,labels=labels_y)+
    coord_fixed()
```



(a) bar map (b)coxcomb map

1.6 Voronoi map

Voronoi diagram (also known as Dirichlet tessellation, Dirichlet tessellation) is a space division algorithm established by Russian mathematician Georgi Voronoi. The inspiration comes from Descartes' idea of dividing space with convex domains. It is a continuous polygon composed of a set of vertical bisectors linking two adjacent points. N points that are distinct on the plane, divide the plane according to the nearest neighbor principle; each point is associated with its nearest area.

Simply put, when we see a series of given points in space, such as x, y1, y2, y3, we want to delineate an area surrounding this point for each point, such as x, such as area Cx. This cx containing x can be called a voronoi cell. For any point located in cx, such as px, we always want it to be the distance from point x to all other given points. For example, the distance of y1, y2, and y3.

In practice, we can link each point with some neighboring points, and connect them with one line segment after another. For this line segment, we can make its vertical bisector (if it is three-dimensional, it is the vertical bisector), These vertical bisectors will enclose an area, such an area is a cell.

Vornoi map can be seen as the map that has more limitation of area on Vornoi diagram.

Choropleth maps drawing

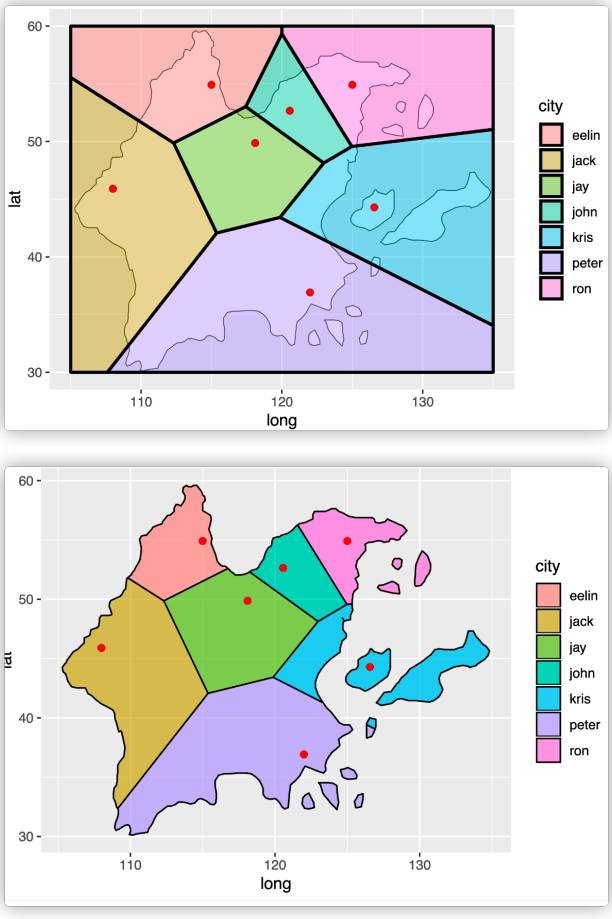
The realization of the map requires the spatial geographic location data of the scattered points from the data format of SpatialPointsDataFrame, and then use SPointDF_to_voronoi_SPolyDF() to generate a Voronoi diagram in SpatialPolygonsDataFrame format.

After that, use the intersect() function in the raster package to obtain the overlapping areas of the Voronoi diagram and different countries and regions one by one, so as to obtain the Voronoi map

```
#Reference:
#https://gis.stackexchange.com/questions/140504/extracting-intersection-areas-in-r
#https://www.codesd.com/item/fill-the-voronoi-polygons-with-ggplot.html
library(ggplot2)
library(sp)
library(deldir) #deldir()
library(raster) #intersect()
dataProjected <- readOGR("Virtual_Map0.shp")
dataProjected@data$id <- rownames(dataProjected@data)
watershedPoints <- fortify(dataProjected)
df_map <- full_join(watershedPoints, dataProjected@data, by = "id")
df_city<-read.csv("Virtual_City.csv")
dati<-
data.frame(x=df_city$long,y=df_city$lat,z=df_city$orange,city=df_city$city)
vor_pts <- SpatialPointsDataFrame(cbind(dati$x,dati$y),dati, match.ID=TRUE)
SPointsDF_to_voronoi_SPolysDF <- function(sp) {
  # tile.list extracts the polygon data from the deldir computation
  vor_desc <- tile.list(deldir(sp@coords[,1],
sp@coords[,2],rw=c(105,135,30,60)))
  lapply(1:(length(vor_desc)), function(i) {
    # tile.list gets us the points for the polygons but we
    # still have to close them, hence the need for the rbind
    tmp <- cbind(vor_desc[[i]]$x, vor_desc[[i]]$y)
    tmp <- rbind(tmp, tmp[1,])
    # now we can make the Polygon(s)
    Polygons(list(Polygon(tmp)), ID=i)
  }) -> vor_polygons
  # hopefully the caller passed in good metadata!
  sp_dat <- sp@data
  # this way the IDs _should_ match up w/the data & voronoi polys
  rownames(sp_dat) <-
sapply(slot(SpatialPolygons(vor_polygons),'polygons'),slot, 'ID')
  SpatialPolygonsDataFrame(SpatialPolygons(vor_polygons),data=sp_dat)}
vor <- SPointsDF_to_voronoi_SPolysDF(vor_pts)
#(a) voronoi diagram
vor@data$id <- rownames(vor@data)
df_vor <- fortify(vor)
df_vor <- full_join(df_vor, vor@data,by='id')
ggplot() +
  geom_polygon(data=df_map, aes(x = long, y = lat,group=group),
```

```
                fill="white",colour="black",size=0.25) +
  geom_polygon(data=df_vor, aes(x = long, y = lat,group=group,fill=city),
           color="black", size=1,alpha=0.4)+
  geom_point(data = dati, aes(x,y), size=4, shape=21, color="black",
fill="red", stroke=0.1)
#(b) voronoi map
group<-1:length(vor)
mypolys<-lapply(group,function(x) {
                  tmp = intersect(dataProjected, vor[x,]);
                  df_pi= fortify(tmp)[c('long','lat','group')]
                  df_pi$city=as.character(vor[x,]@data$city[1])
                  df_pi$group=as.numeric(df_pi$group)+runif(1,0,100)
                  df_pi})

df_vor<-
data.frame(long=numeric(0),lat=numeric(0),group=numeric(0),city=character(0))
for (i in group ){
  df_vor<-rbind(df_vor,mypolys[[i]])
}
ggplot() +
  geom_polygon(data=df_vor,aes(x = long, y = lat,group=group,fill=city),
           color="black", size=0.5,alpha=0.6)+
  geom_point(data = dati, aes(x,
y),size=4,shape=21,color="black",fill="red",stroke=0.1)
```



(a). Voronoi diagram (b). Voronoi map

1.7 Connection map

Connection map is a method using straight lines or curves to connect locations on a map. It shows different routes (**Figure a** and **Figure b**). In addition, through analyzing the connection or distribution on a given map, we can have a better understanding of its spatial pattern.

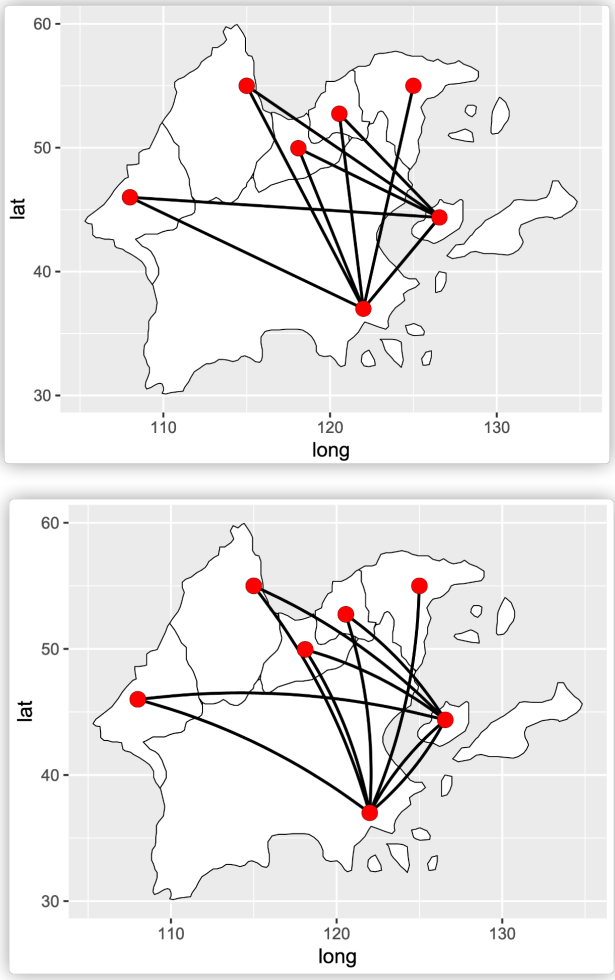
The most common connection map is the flight route map with origins and destinations of each flight, so there will be a lot of overlaps. To avoid this situation, we can set transparency of lines.

Flow map can show the movements information. Each flow has its origin and destination, and its values as well. The values can be shown by the width of lines (**Figure c**). Sometimes, arrows can be added to show the directions clearly (**Figure d**).

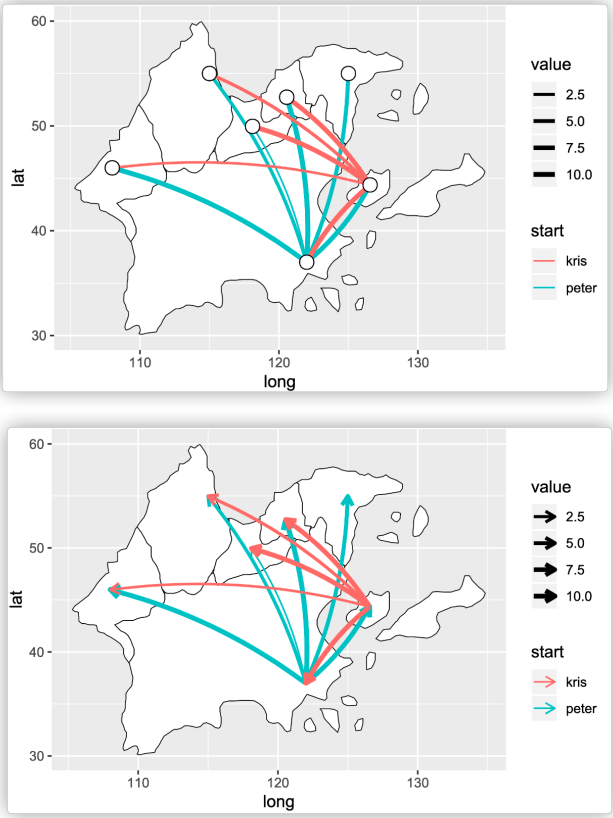
The geom_curve() function from ggplot2 package defines origin (x, y) and destination (xend, yend) and therefore connects two dots. The curvature can control the bending degree, and the arrow can control the size of arrows.

```
library(rgdal)   #readOGR()
library(ggplot2)
library(dplyr)
library(RColorBrewer)
library(tidyr)
dataProjected <- readOGR("Virtual_Map1.shp")
dataProjected@data$id <- rownames(dataProjected@data)
watershedPoints <- fortify(dataProjected)
df_map <- full_join(watershedPoints, dataProjected@data, by = "id")
df_city<-read.csv("Virtual_City.csv") [c('long','lat','city')]
df_connect<-read.csv("Virtual_Connect.csv")
df_connect<-df_connect %>%
  left_join(df_city,by=c('start'='city'))%>%
  left_join(df_city,by=c('end'='city'))
#(a) connection map with straight lines
ggplot() +
  geom_polygon(data=df_map, aes(x = long, y = lat,group=group),
               fill="white",colour="black",size=0.25)+
  geom_curve(data=df_connect,aes(x=long.x,y=lat.x,xend=long.y,yend=lat.y),
             size=0.75,colour="black",curvature = 0)+
  geom_point(data =df_connect,aes(x=long.y,y=lat.y),
             size=4,shape=21,fill="#F00000",colour="black",stroke=0.1)
#(b)connection map with curves
ggplot() +
  geom_polygon(data=df_map, aes(x = long, y = lat,group=group),
               fill="white",colour="black",size=0.25)+
  geom_curve(data=df_connect,aes(x=long.x,y=lat.x,xend=long.y,yend=lat.y),
             size=0.75,colour="black",curvature = 0.1)+
  geom_point(data =df_connect,aes(x=long.y,y=lat.y),
             size=4,shape=21,fill="#F00000",colour="black",stroke=0.1)
#(c) flow map without arrow
ggplot() +
  geom_polygon(data=df_map, aes(x = long, y = lat,group=group),
```

```
                fill="white",colour="black",size=0.25)+
  geom_curve(data=df_connect,aes(x=long.x,y=lat.x,xend=long.y,yend=lat.y,
                                 colour=start,size=value),
             curvature = 0.1)+
  scale_size(range=c(0.5,1.5))+
  geom_point(data =df_connect,aes(x=long.y,y=lat.y),
             size=4,shape=21,fill="white",colour="black",stroke=0.5)
#(d) flow map with arrow
ggplot() +
  geom_polygon(data=df_map, aes(x = long, y = lat,group=group),
               fill="white",colour="black",size=0.25)+
  geom_curve(data=df_connect,aes(x=long.x,y=lat.x,xend=long.y,yend=lat.y,
                                 size=value,colour=start),
             arrow = arrow(length = unit(0.25, "cm")),curvature = 0.1)+
  scale_size(range=c(0.5,1.5))
```



(a) connection map with straight lines (b) connection map with curves



(c) flow map without arrow (d) flow map with arrow

1.8 Isopleth map

Isopleth maps simplify information about a region by showing areas with continuous distribution. Isopleth maps may use lines to show areas where elevation, temperature, rainfall, or some other quality is the same; values between lines can be interpolated. Isopleths may also use color to show regions where some quality is the same; for example, a map that use shades from red to blue to indicate temperature ranges. An isopleth differs from a choropleth because the lines or regions are determined by the data rather than conforming to a predefined area, such as a political subdivision.

This type of map is ideal for showing gradual change over space and avoids the abrupt changes which boundary lines produce on choropleth maps. Temperature, for example, is a phenomenon that should be mapped using isoplething, since temperature exists at every point (is continuous), yet does not change abruptly at any point (like population density may do as you cross into another census zone). Relief maps should always be in isopleth form for this reason.

There are two main types of isopleth map in R. In terms of displayed area, it can be divided into local isopleth maps and global isopleth maps. The first method can be implemented by geom_contour() function in ggplot2 in R, directly draw and overlay layers of contour lines based on data. The second method is to use geom_tile() or geom_raster() function and do interpolation processing to get data of whole map. Then we draw the thermodynamic plot.

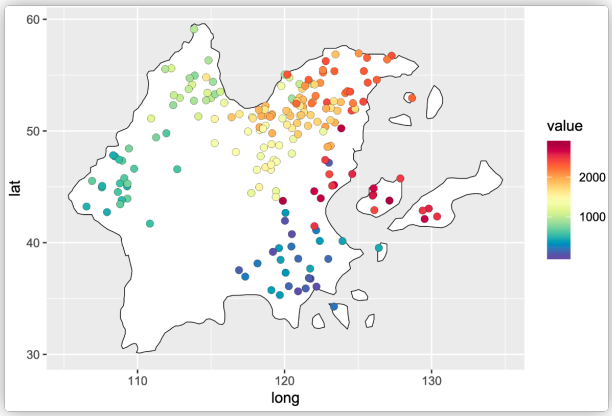
Choropleth maps drawing

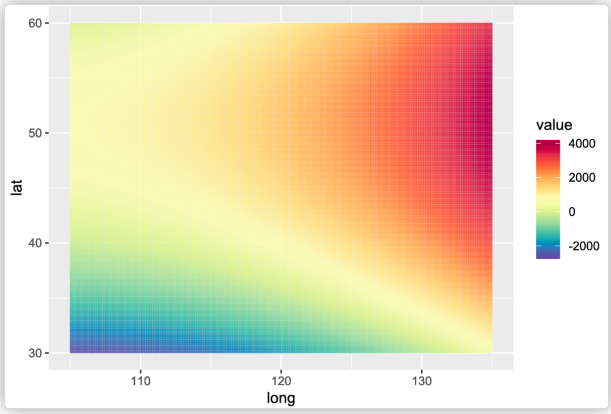
To draw a two-dimensional kernel density equipotential map, first use the sm.density function in the sm package to calculate the two-dimensional kernel density distribution map based on the scattered point distribution data, and then convert the two-dimensional kernel density distribution data into SpatialPixelsDataFrame format data, use The !is.na() function calculates the overlap area of the two-dimensional kernel density distribution map and the map in SpatialPolygonsDataFrame format. Finally,

the geom_raster() and geom_contour() functions are used to draw the heat map and contour lines.

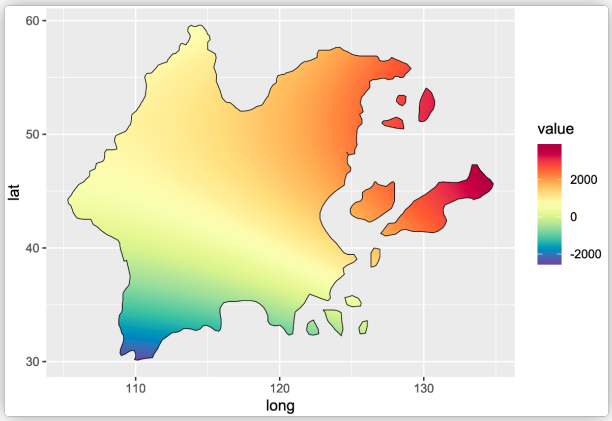
```
library(rgdal)   #readOGR()
library(ggplot2)
library(dplyr)
library(RColorBrewer)
library(sm) #sm.density
colormap<-colorRampPalette(rev(brewer.pal(11,'Spectral')))(32)
dataProjected <- readOGR("Virtual_Map0.shp")
dataProjected@data$id <- rownames(dataProjected@data)
watershedPoints <- fortify(dataProjected)
df_map <- full_join(watershedPoints, dataProjected@data, by = "id")
df_huouse<-read.csv("Virtual_huouse.csv")
ggplot()+
  geom_polygon(data=df_map, aes(x=long, y=lat,
group=group),fill='white',colour="black",size=0.25)+
  geom_point(data=df_huouse,aes(x=long,
y=lat),shape=19,colour="black",size=1,alpha=1)
cycle_dens<- sm.density(data.frame(df_huouse$long, df_huouse$lat),
                        display= "image", ngrid=500,
                        ylim=c(30,60),xlim=c(105,135))
Density_map<-SpatialPoints(expand.grid(x=cycle_dens$eval.points[,1],
                                y=cycle_dens$eval.points[,2]))
Density_map<-SpatialPixelsDataFrame(Density_map, data.frame(kde =
array(cycle_dens$estimate,
     length(cycle_dens$estimate))))
#(b)
df_density<-data.frame(Density_map)
ggplot(df_density,aes(x=x,y=y,z=kde))+
  geom_tile(aes(fill=kde))+
  scale_fill_gradientn(colours=colormap)+
  #stat_contour(aes(colour= ..level..),breaks=breaks_lines,color="black")+
  labs(fill='kde',
       x='long',
       y='lat')
#(c)
group<-1:length(dataProjected)
mypolys<-lapply(group,
                function(x) {
                  tmp = !is.na(over(Density_map, dataProjected[x,]));
                  clipped_grid= Density_map[tmp[,1],];
                  clipped_grid})
df_density<-data.frame(x=numeric(0),y=numeric(0),kde=numeric(0))
for (i in group){
  df_density<-rbind(df_density,cbind(mypolys[[i]]@coords,mypolys[[i]]@data))
}
min_z<-min(df_density$kde)
max_z<-max(df_density$kde)
breaks_lines<-seq(min_z,max_z,by=(max_z-min_z)/20)
```

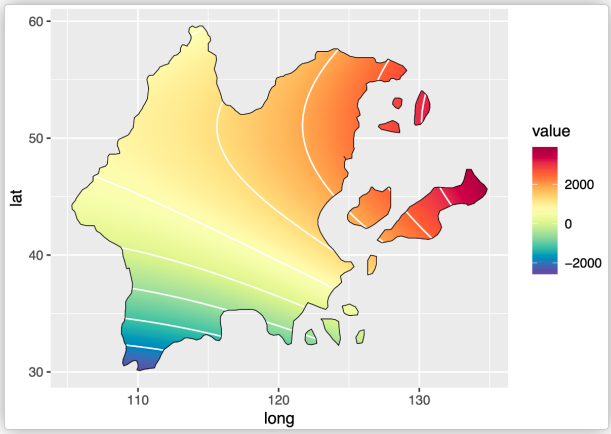
```
ggplot() +
  geom_raster(data=df_density,aes(x=x,y=y,fill=kde))+
  geom_contour(data=df_density,aes(x=x,y=y,z=kde),color="white",breaks=breaks_l
ines)+
  scale_fill_gradientn(colours=colormap)+
  geom_path(data=df_map, aes(x=long, y=lat,
group=group),colour="black",size=0.25)+
  coord_cartesian()
```





(a) dot map (b)2-d interpolation heat diagram





(c) 2-d interplation estimation heat map (d) 2-d interpolation isopleth maps