

Map, GIS, SHP and R

장동익

한국교통연구원 국가교통DB센터

May 15, 2015

1 들어가며

교통은 도로와 철도, 바다, 하늘에서 이루어지는 자동차, 기차, 선박, 항공기들의 다양한 경로와 이들 자체와 이들의 이용자인 사람들의 이동과 관련된 정보를 분석하고 연구하는 영역이다. 따라서 교통에서는 이들의 Geographic Information System(GIS) 자료를 지도 위에 표출하고 분석하는 일들이 일상적으로 이루어진다. 교통의 흐름을 분석하기 위하여 도로의 차선 수, 최고·최저 속도 정보, 도로등급 정보, 철도망, 버스 노선망, 항공기 노선망과 같은 기본적인 시설물 정보에서부터 실시간으로 수집되는 자동차 내비게이션에 설치된 GPS 정보, 스마트폰 내비게이션 App 사용자 GPS 정보, 교통카드의 승하차 정보, 버스의 BIS/BMS 정보, 항공기의 위치 정보, 스마트폰 사용자의 접속 기지국 정보, WIFI 기기 정보 등, 다양한 GIS 정보들이 활용이 되고 있다.

이러한 GIS 자료를 분석하기 위해 다양한 설계/관리 플랫폼이 이용이 되고 있으며 대표적인 프로그램인 ArcGISTM 와 QGIS가 널리 사용이 되고 있다. ArcGISTM는 다양한 분석툴과 높은 시장 점유율 바탕으로 다양한 분야의 GIS 기본 엔진으로 활용되고 있는 대표적인 프로그램이다. ArcGISTM가 Windows 전용 상업용 소프트웨어라면 QGIS는 Open Source이며 Windows 뿐만 아니라 Mac OS X과 Linux를 지원하고 있어서 많은 개발자들이 애용하고 있는 어플리케이션이다.

GIS 소프트웨어들은 raster 이미지 형태의 배경 지도를 활용하거나 다양한 벡터 형태의 layer 데이터를 중첩하여 자료를 표출함으로써 지도를 이용하여 보다 의미 있는 분석을 위한 기반을 제공해주기 때문에 지도를 이용한 매쉬업(mashup)에 널리 이용되고 있다. 또한 이들은 간단한 공간통계적 분석 뿐만 아니라 복잡한 네트워크 분석들도 지원을 하고 있으며 Python과 같은 프로그램 언어들과 결합하여 기본적인 분석에 다양한 분석의 툴을 추가하여 분석의 범위를 확장할 수 있다.

통계학에서도 공간통계 뿐만 아니라 기상/기후, 보건의료 등의 영역에서 공간구조를 가진 다양한 자료를 R을 이용하여 분석하여 이를 표출하여 패턴을 이해하고 새로운 통찰은 얻는 과정을 오랫동안 수행해 왔다. 최근에는 R이 더욱 다양한 영역에서 활용이 되면서 통계분석 뿐만 아니라 GIS 분석이나 웹 데이터 parsing, interactive chart 등 다양한 패키지들의 개발이 되고 있다. 그리고 교통의 영역에서도 다양한 세부 분야에 R이 활용되고 있으며 GIS 프로그램에 익숙한 연구원들이 R을 지도기반 자료와 결합하여 분석하려는 시도가 증가하고 있다.

이 글에서는 R의 기초적인 `plot` 함수를 활용한 매쉬업에서 출발하여 최근에 개발된 다양한 지도 자료들을 활용할 수 있는 패키지들을 소개하고 이들의 활용법을 정리하여 R을 통하여 다양한 GIS 자료와 지도들을 어떻게 활용할 수 있는지에 대해서

간단히 소개하고자 한다.

본격적으로 시작하기 전 필 요한 패키지와 자료를 다운로드하기 위한 source는 다음과 같다.

```
dir.create("KSSLetter")
setwd("KSSLetter")

gitadd <- "https://github.com/dongikjang/"
gitadd2 <- "https://raw.githubusercontent.com/dongikjang/"

# install required packages
reqpkgs <- c("mapdata", "RColorBrewer", "RNetCDF", "colorRamps", "rgl", "ggmap", "jpeg", "png",
            "plyr", "fields", "geosphere", "XML", "bitops", "jpeg", "ggplot2", "scales",
            "polyclip", "maptools", "rgdal", "data.table", "RgoogleMaps", "sp", "maps", "RCurl")
inspkgind <- !reqpkgs %in% installed.packages() [,1]
if(any(inspkgind)){
  for(inspkg in reqpkgs[inspkgind]) install.packages(inspkg)
}

update.packages()

download.file.Bin <- function(url, destfile, encoding="UTF-8"){
  require(RCurl)
  binfile <- getBinaryURL(url, encoding=encoding,
                           cainfo = system.file("CurlSSL", "cacert.pem", package = "RCurl"))
  con <- file(destfile, open = "wb")
  writeBin(binfile, con)
  close(con)
}
source_https <- function(url, ...) {
  # load package
  require(RCurl)
  require(plyr)
  # parse and evaluate each .R script
  l_ply(c(url, ...), function(u) {
    eval(parse(text = getURL(u, followlocation = TRUE,
                           cainfo = system.file("CurlSSL", "cacert.pem", package = "RCurl"))),
    envir = .GlobalEnv)
  })
}
if(.Platform$OS.type == "windows"){
  curlstate <- system("curl --help", invisible=TRUE) == 0L
} else{
  curlstate <- system("curl --help > /dev/null") == 0L
}
```

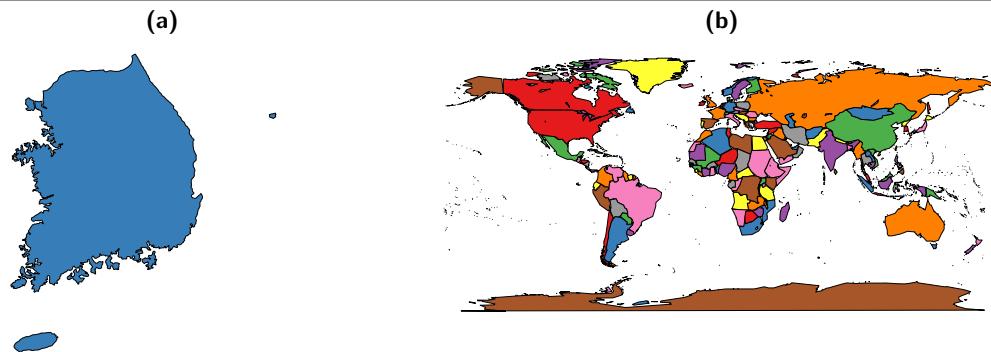


Figure 1: 패키지 `mapdata`에 들어있는 전 세계 해안선과 국가별 영역 자료. (a) 대한민국, (b) 전 세계 국가별 영역.

2 패키지 `mapdata` 와 NetCDF

2.1 `mapdata`

R에서 데이터를 지도 위에 표출하기 위하여 가장 널리 사용되는 패키지는 `mapdata` (Becker et al., 2014)이다. 이 패키지에는 `list` 형태로 전 세계 해안선과 국가별 영역 자료가 내장되어 있다. 하지만 이 영역 자료는 매우 오래된 CIA 데이터베이스를 기반으로 만들어졌기 때문에 러시아 연방이 여전히 USSR이라는 이름으로 지정되어 있는 등 현재 시점과 맞지 않는 요소들이 많이 들어있다. 패키지 `mapdata`에 있는 지도 자료를 R에서 불러오기 위해서는 `map` 함수를 사용한다. 아무런 옵션을 지정하지 않으면 CIA World Data Bank II 데이터 (<http://www.ev1.uic.edu/pape/data/WDB/>)를 축소하여 30,000여개의 점으로 전 세계 해안선과 국가별 경계를 표현하는 지도를 사용하도록 설정되어 있으며 `database=worldHires`로 옵션을 변경하면 2백만여개의 점들을 사용하는 고해상도 지도를 사용할 수 있다. 대서양을 중심으로 하는 세계 지도가 기본이며 태평양을 중심으로 하는 세계 지도를 그리고자 하는 경우 `database`를 `world2`나 `world2Hires`로 바꾸면 된다. 전 세계 지도가 아니라 특정 국가들만을 사용하고자 하면 `regions`에 국가 이름들을 넣어주면 된다. 한반도의 영역만을 추출하고자 하는 경우에는 `regions=c('South Korea', 'North Korea')`로 선언하면 된다. 패키지에서 지원하는 영역 이름들은 다음과 같이 `map` 함수에서 `namesonly=TRUE`를 지정하여 확인할 수 있다.

```
library(maps)
str(map("world", namesonly=TRUE, plot=FALSE))

# chr [1:2284] "Canada" "South Africa" "Denmark" ...
```

패키지 `mapdata`를 사용하여 `South Korea`와 전 세계 지도를 사용하는 간단한 예는 다음과 같다.

```
# Figure 1
library(mapdata)
library(RColorBrewer)
cols <- brewer.pal(9, "Set1")
# (a)
southkor <- map('worldHires', 'South Korea', fill=TRUE, plot=FALSE)
str(southkor)

# List of 4
# $ x      : num [1:6627] 127 127 127 127 127 ...
```

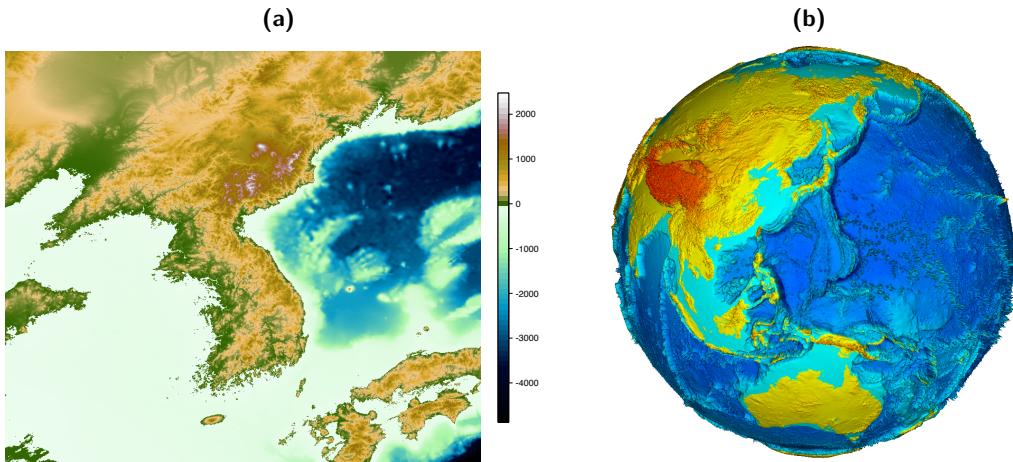


Figure 2: NOAA NGDC에서 제공하는 Grid of Earth's surface 자료를 패키지 `RNetCDF`를 통해 불러온 지구지표면 그림 (a) 한반도 주변의 지표면 고도 (b) 패키지 `rgl`을 이용한 3D 지표면.

```
# $ y      : num [1:6627] 37.8 37.8 37.8 37.8 37.8 ...
# $ range: num [1:4] 124.6 130.9 33.2 38.6
# $ names: chr [1:8] "South Korea" "South Korea:Ullung Do" "South Korea:?" "South Korea:Namhae Do" ...
# - attr(*, "class")= chr "map"

plot(southkor, type="n", asp=1, axes=FALSE, xlab="", ylab="", main="")
polygon(southkor, col = cols[2], border=1)
# (b)
#map('worldHires', fill=TRUE, plot=TRUE, col=cols)
```

그림 1은 위의 R 스크립트의 실행결과를 보여 주고 있다. 각 영역의 시작점과 끝점이 일치하는 `polygon` 형태의 자료를 불러오기 위하여 `fill=TRUE` 옵션을 지정하였으며 이를 `polygon` 함수를 사용하여 각 지역별로 다른 색을 지정하여 영역을 채우도록 설정 하였다.

2.2 NetCDF

Network Common Data Form(NetCDF)는 수치모형이나 Grid 자료의 입출력으로 가장 많이 사용되는 라이브러리이다. NetCDF는 미국 기상 연구대학 연합(University Corporation for Atmosphere Research, UCAR) 소속인 Unidata가 주로 국립 과학 재단(National Science Foundation)의 기금을 바탕으로 개발, 유지되고 있다. NetCDF는 자료를 등간격의 Grid 형태로 저장, 배포하기 때문에 간단한 변환과정을 거치면 R의 `image`와 같은 `grid` 데이터 기반의 함수에서 사용이 가능하다. 현재 R은 NetCDF 자료를 불러올 수 있는 여러 패키지들이 제안되어 있다. 이중 대표적인 패키지는 `ncdf` (Pierce, 2014)와 `ncdf4` (Pierce, 2014), `RNetCDF` (Michna, 2014)가 있다. 다음은 패키지 `RNetCDF`를 이용하여 NOAA의 National Geophysical Data Center(NGDC)에서 제공하는 Grid of Earth's surface가 저장된 NetCDF 자료를 불러와 한반도 주변 지표면 고도 지도를 작성하는 R 스크립트이다.

```
# Figure 2 (a)
if(curlstate){
  download.file(paste(gitadd2, "kssletter/master/etopo1.nc", sep=""),
    destfile="etopo1.nc", method="curl", extra="-L -k ", quiet=TRUE)
```

```

} else {
  library(RCurl)
  download.file.Bin(paste(gitadd2, "kssletter/master/etopo1.nc", sep=""), "etopo1.nc")
}

library(RNetCDF)
nc <- open.nc("etopo1.nc")
tmp <- read.nc(nc)
names(tmp) <- c("crs", "y", "x", "z")
close.nc(nc)

ocean.pal <- colorRampPalette(
  c("#000000", "#000209", "#000413", "#00061E", "#000728", "#000932", "#002650", "#00426E",
    "#005E8C", "#007AAA", "#0096C8", "#22A9C2", "#45BCBB", "#67CFB5", "#8AE2AE", "#ACF6A8",
    "#BCF8B9", "#CBF9CA", "#DBFBDC", "#EBFDDE")
)

land.pal <- colorRampPalette(
  c("#336600", "#F3CA89", "#D9A627", "#A49019", "#9F7B0D", "#996600", "#B27676", "#C2B0B0",
    "#E5E5E5", "#FFFFFF")
)

zbreaks <- seq(min(tmp$z), max(tmp$z), by=5)
cols <- c(ocean.pal(sum(zbreaks<=0)), land.pal(sum(zbreaks>0)-1))

library(fields)
image.plot(tmp, col=cols, asp=1, breaks=zbreaks, useRaster=TRUE, xlab="", ylab="", axes=FALSE,
           legend.width = 2, axis.args=list(cex.axis=1.5), legend.shrink=.8, legend.mar=6)

```

NGDC의 Grid Extract 사이트(<http://maps.ngdc.noaa.gov/viewers/wcs-client/>)에서는 전 세계의 1분 단위 지표면 고도 자료를 제공하고 있으므로 추출하고자 하는 영역을 설정하여 Output Format을 NetCDF로 설정하여 앞의 R 스크립트에 사용된 **etopo1.nc**란 이름의 ‘Grid of Earth’s surface’ 자료를 다운로드 받을 수 있다. 스크립트의 실행 결과는 그림 2 (a)와 같다. 그림 2 (b)는 전 세계 지표면 고도 자료를 불러와 3D 시각화 패키지인 **rgl** (Adler et al., 2014)을 이용하여 3D 구의 형태로 표출된 모습을 보여주고 있다. 그림 2 (b)를 위한 R 스크립트는 다음과 같다.

```

# Figure 2 (b)
if(curlstate){
  download.file("https://www.dropbox.com/s/6opewjw02wn0fj0/SurfaceWorld.xyz?dl=0",
                destfile="SurfaceWorld.xyz", method="curl", extra="-L -k ", quiet=TRUE)
} else {
  library(httr)
  sfbin <- GET("https://www.dropbox.com/s/6opewjw02wn0fj0/SurfaceWorld.xyz?dl=1")
  writeBin(sfbin$content, "SurfaceWorld.xyz")
  rm(sfbin)
}
library(colorRamps)

```

```

xyz <- read.delim("SurfaceWorld.xyz", header=F)
vals <- matrix(xyz[,3], 2161,1081)
long <- pi*matrix(xyz[,1], 2161,1081)/180
lati <- pi*matrix(xyz[,2], 2161,1081)/180
vals <- vals[nrow(vals):1,]
z <- (vals-min(vals))/diff(range(vals))*4) + 1
r <- z
z <- r*sin(lati)
y <- r*cos(lati)*cos(long)
x <- r*cos(lati)*sin(long)
nlevel <- 1024
tmp <- round((nlevel-1)*vals/diff(range(vals)) + 1.5)
n1 <- -min(tmp)
n2 <- nlevel -n1 #max(tmp)
col1 <- blue2red(2*n1)[1:n1]
col2 <- blue2red(2*n2)[(n2+1):(2*n2)]
col <- c(col1, col2)
col <- col[tmp+n1]
library(rgl)
persp3d(x, y, z, col=col, specular="black", axes=FALSE, box=FALSE, xlab="", ylab="", zlab="")

```

3 Raster image

패키지 `mapdata`에 내장되어 있는 지도 자료는 R에서 제공하는 기본 `plot`이나 `lines`, `polygon`과 같이 그림과 관련된 함수들에서 손쉽게 사용할 수 있는 `list` type이므로 각 구역을 점들로 표현하고 이들은 선으로 이어서 영역을 구분하는 벡터 이미지를 만들어 내는데 주로 활용이 된다. 이러한 방식은 고해상도 지도를 그리는데 적합하나 지도 위에 표시해야 하는 점들이 늘어나는 경우 이를 전송하고 저장하기 위해서는 많은 공간을 필요로 하게 되는 단점이 있다.

벡터 기반의 지도를 서비스 하는 방식은 서버와 개인 사용자간에 엄청난 네트워크의 트래픽을 발생시키기 때문에 Google Map나 Daum, Naver 지도와 같은 웹 기반의 지도 사업자들은 해상도에 따라 여러 레벨의 지도 이미지를 미리 작성하여 이를 작은 tile들로 쪼개어 서버에 저장하고 개인 사용자들의 요청 영역만을 전송하여 open layer 같은 툴을 이용하여 이를 결합하고 웹 브라우저에서 표출하는 방식을 사용하고 있다. 이러한 방식을 GIS 프로그램에서는 raster 이미지 기반의 서비스라고 부른다.

3.1 Google Maps

Google은 개발자들이 쉽게 자신들의 Google Maps를 사용할 수 있도록 Open Application Programming Interface(API)를 제공하고 있다. 이 Open API를 이용하여 Goole Maps에서 제공하는 png 파일 형태의 지도 자료 R을 통해 불러와 이를 배경 지도로 활용하여 그 위에 다양한 정보를 표출할 수 있다. Google Maps을 R에서 사용할 수 있도록 개발된 패키지는 대단히 많으며 그 중 대표적인 패키지는 `RgoogleMaps` (Loecher, 2014)과 `ggmap` (Kahle and Wickham, 2013)이다.

`RgoogleMaps`는 지도 type을 `roadmap` 과 `mobile`, `satellite`, `terrain`, `hybrid`, `mapmaker-roadmap`, `mapmaker-hybrid` 을 지원하고 있으나 대한민국의 영역에서는 현재 `roadmap`과 `satellite`, `terrain`만을 제공하고 있다. 그림 3는 패키지 `RgoogleMaps`을 활용하여 R에서 그린 Google Maps의 예들을 보여주고 있다.

패키지 `RgoogleMaps`을 통해 R에서 Google Maps을 사용하기 위해서는 `GetMap` 함수를 이용하여 지도의 `center`와 해상도 `zoom`을 지정하면 된다. 그리고 `PlotOnStaticMap` 함수를 이용하여 다운받은 Google Maps를 불러와 지도를 그릴 수 있으며,

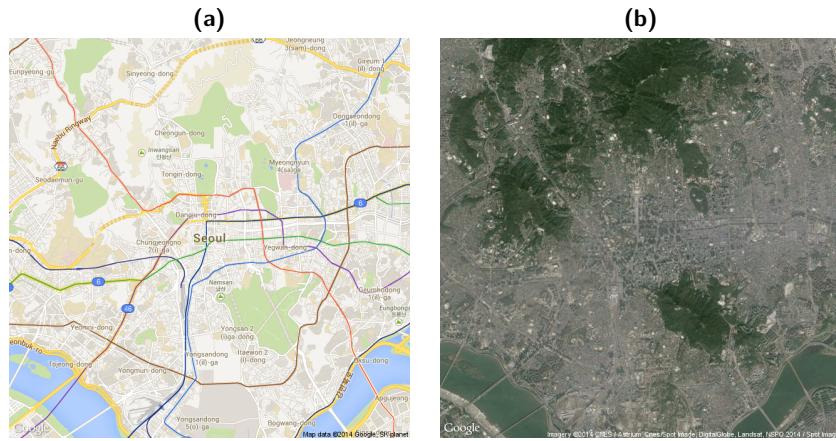


Figure 3: 패키지 `RgoogleMaps`을 이용하여 가져온 zoom 12인 서울 시청 중심의 Google Maps. (a) Roadmap, (b) Satellite.

`PlotOnStaticMap`에 있는 다양한 옵션들을 활용하여 지도 위에 점이나 선과 같은 방식으로 다양한 정보를 추가하는 것도 가능하다.

```
# Figure 3
library(RgoogleMaps)

MyMap <- GetMap(center = c(37.5665,126.978), zoom =13, size = c(640, 640),
                 destfile = "seoul1.png", maptype="roadmap")
PlotOnStaticMap(MyMap, size = c(640, 640))

# (a)
MyMap <- GetMap(center = c(37.5665,126.978), zoom =13, size = c(640, 640),
                 destfile = "seoul1.png", maptype="roadmap")
PlotOnStaticMap(MyMap, size = c(640, 640))

# (b)
MyMap <- GetMap(center = c(37.5665,126.978), zoom =13, size = c(640, 640),
                 destfile = "seoul2.png", maptype="satellite")
PlotOnStaticMap(MyMap, size = c(640, 640))

# not included in the letter
MyMap <- GetMap(center = c(37.5665,126.978), zoom =13, size = c(640, 640),
                 destfile = "seoul3.png", maptype="terrain")
PlotOnStaticMap(MyMap, size = c(640, 640))
```

`RgoogleMaps` 패키지는 Open API를 이용하기 때문에 zoom에 따라서 지도의 크기가 변화지 않으며 고해상도의 지도의 경우 매우 좁은 영역만을 반영하게 된다는 단점이 있다.

패키지 `ggmap`은 요즘 널리 쓰이는 layer 타입의 데이터 시각화 패키지 `ggplot2` (Wickham and Chang, 2014) 기반으로 만들어져 있다. 패키지 `ggmap`은 `RgoogleMaps`과 달리 지도의 여러 부분 tile들을 가져와서 조합하기 때문에 `RgoogleMaps`에 비해 넓은 영역을 고해상도로 표현할 수 있다. 패키지 `ggmap`에 있는 `qmap`이나 `get_map` 함수를 이용하여 Google Maps나 OpenStreetMap, Stamen Maps에서 지도를 가져와 사용할 수 있으며 API 키를 발급받으면 CloudMade maps 또한 사용이 가능하다. 이들의 사용 방법은 다음 R 스크립트와 같으며 그 실행 결과를 그림 4가 보여주고 있다.

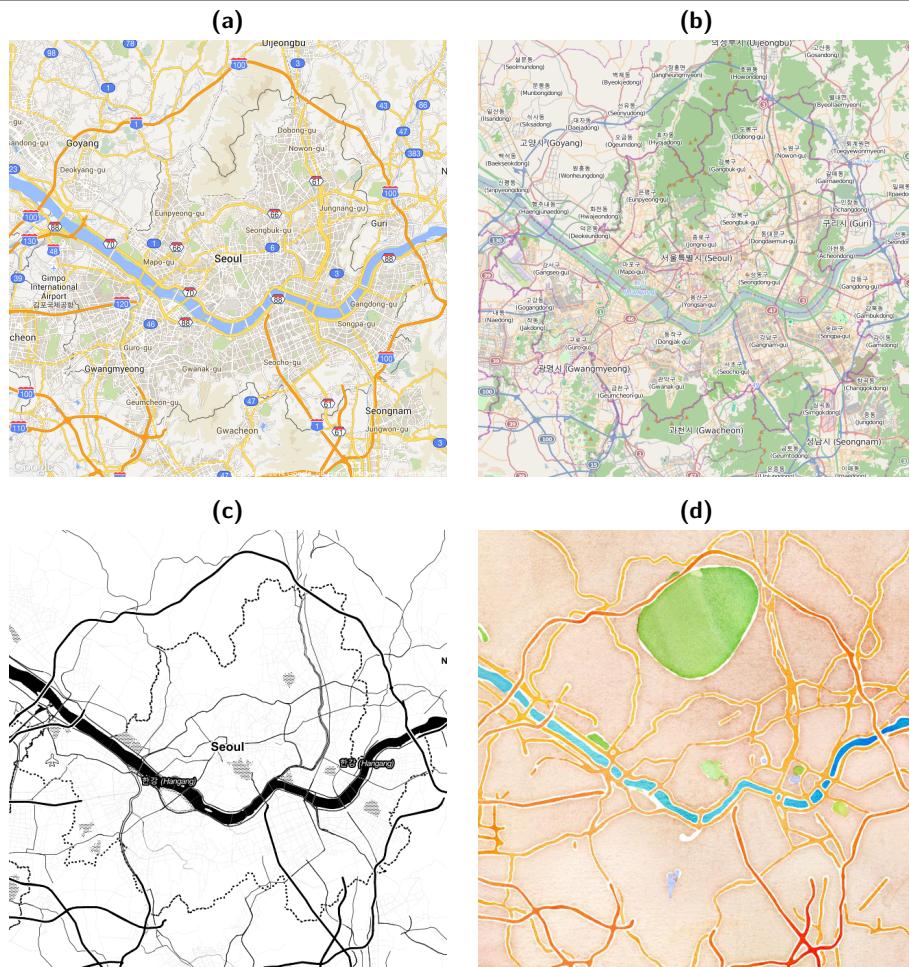


Figure 4: R 패키지 `ggmap`을 이용하여 나타낸 다양한 형태의 수도권 지도. (a) Google Maps의 Hybird type, (b) OpenStreetMap, (c) Stamen Maps의 toner type, (d) Stamen Maps의 water color type.

```
# Figure 4
library(ggmap)
library(jpeg)
library(png)
library( plyr)
source_https(paste(gitadd2, "GoogleMap/master/StamenWatercolor.R", sep=""))
# (a)
qmap("seoul", zoom = 11, maptype = 'hybrid')
# (b)
qmap("seoul", zoom = 11, source = 'osm')
# (c)
qmap("seoul", zoom = 11, maptype = 'toner', source = 'stamen')
# (d)
qmap("seoul", zoom = 11, maptype = 'watercolor', source = 'stamen')
```

최근 Stamen Maps 서버의 업데이트로 인해 `qmap`의 watercolor type은 오류가 있는 상황이다. 패키지 `ggmap`의 업데이트 전까지는 <https://github.com/dongikjang/GoogleMap/blob/master/StamenWatercolor.R>에 올려져 있는 수정 코드를 활용하여 watercolor type의 Stamen Maps의 사용이 가능하다.

패키지 `ggmap`은 `ggplot2` 기반이기 때문에 `ggplot`에 활용되는 문법들을 그대로 이용하여 지도 위에 다양한 layer를 추가할 수 있을 뿐만 아니라 기본적인 `image`와 같은 함수와 결합 하에 매쉬업에 활용이 가능하다. 그림 5는 Stamen Maps의 toner type을 배경 layer로 사용하여 나타낸 서울 지역 대기오염도 현황을 보여주고 있다. 서울특별시 기후환경본부 (<http://cleanair.seoul.go.kr>)에서는 서울 도심지역 25개 지점의 관측소에서 PM-10, PM-2.5, NO₂, O₃, SO₂, CO과 같은 대기오염 현황을 측정하여 매시간 발표하고 있다. 2014년 4월 서울 전 지역의 평균 대기오염 현황을 살펴보기 위하여 25개 지점의 관측값을 활용하여 공간통계적 기법으로 관측하지 못한 지역의 대기오염 현황을 추정하였다. 지역별 오염의 정도를 확인하기 위하여 Stamen Maps을 바탕 지도로 활용하였다. 그림 5를 위한 R 스크립트는 다음과 같다.

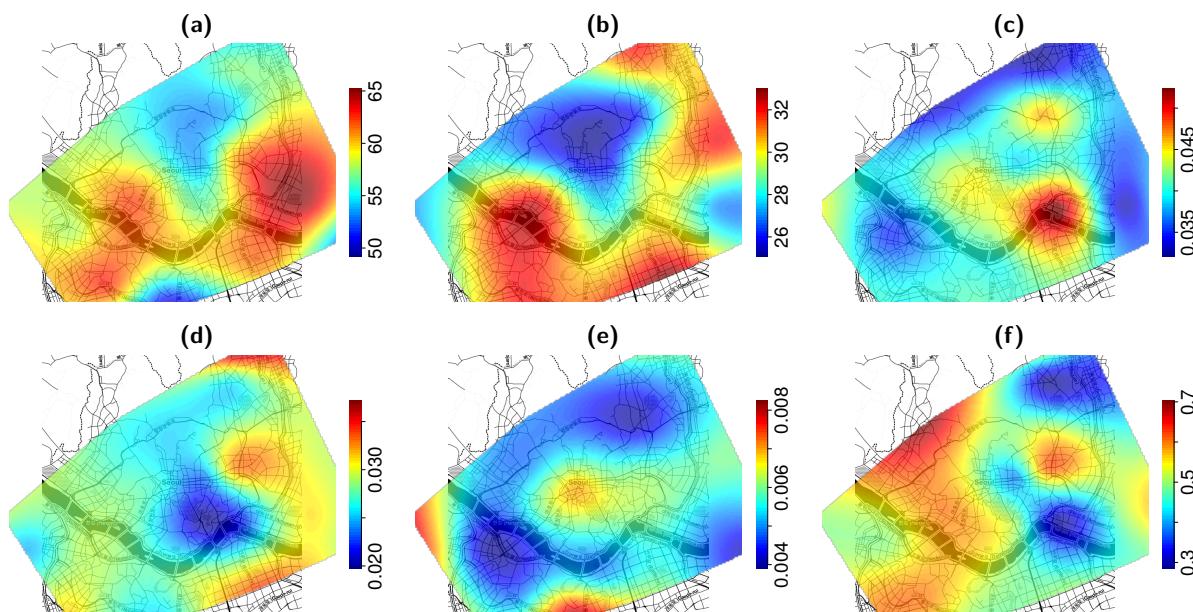


Figure 5: Stamen Maps의 toner type을 배경 layer로 사용하여 나타낸 2014년 4월 서울 지역 대기오염도 현황. (a) PM-10($\mu\text{g}/\text{m}^3$), (b) PM-2.5($\mu\text{g}/\text{m}^3$), (c) NO₂(ppm), (d) O₃(ppm), (e) SO₂(ppm), (f) CO(ppm).

```
# Figure 5
library(ggmap)
if(curlstate){
  download.file(paste(gitadd2, "kssletter/master/cleanair.csv", sep=""), destfile="cleanair.csv",
                method="curl", extra="-L -k ", quiet=TRUE )
  if(Encoding("a") == "unknown" & .Platform$OS.type =="windows"){
    pm10 <- read.csv("cleanair.csv", stringsAsFactors = FALSE, fileEncoding = "UTF-8",
                     encoding = "EUC-KR")
  } else {
    pm10 <- read.csv("cleanair.csv", stringsAsFactors = FALSE, fileEncoding = "UTF-8",
                     encoding = "UTF-8")
  }
}
```

```

} else {
  library(RCurl)
  cleanair <- getURL(paste(gitadd2, "kssletter/master/cleanair.csv", sep=""),
                      cainfo = system.file("CurlSSL", "cacert.pem", package = "RCurl"))
  if(Encoding("a") == "unknown" & .Platform$OS.type == "windows"){
    pm10 <- read.csv(textConnection(cleanair), stringsAsFactors = FALSE,
                     fileEncoding = "UTF-8", encoding = "EUC-KR")
  } else {
    pm10 <- read.csv(textConnection(cleanair), stringsAsFactors = FALSE,
                     fileEncoding = "UTF-8", encoding = "UTF-8")
  }
}

mstodeg <- function(x){
  x <- as.numeric(x)
  x[1] + x[2]/60 + x[3]/3600
}

pm10$경도 <- unlist(lapply(strsplit(pm10$경도, ":"), mstodeg))
pm10$위도 <- unlist(lapply(strsplit(pm10$위도, ":"), mstodeg))

seoulmap <- qmap("seoul", zoom = 11, maptype = 'toner', source = 'stamen')
seoulmap + geom_point(mapping=aes(x =경도, y =위도), colour=brewer.pal(9, "Set1")[1],
                      cex=4, data = pm10)

# location of observation
gm <- get_googlemap(center = "seoul", zoom = 12, filename = "ggmapTemp")
location <- as.numeric(attr(gm, "bb"))[c(2, 1, 4, 3)]
out <- get_stamenmap(bbox = location, zoom = 12, maptype = "toner", crop = TRUE, messaging = FALSE,
                      urlonly = FALSE, filename = "ggmapTmp", color = "color")
xmin <- attr(out, "bb")$ll.lon
xmax <- attr(out, "bb")$ur.lon
ymin <- attr(out, "bb")$ll.lat
ymax <- attr(out, "bb")$ur.lat

outraster <- as.raster(out)

xymar <- data.frame(lon = c(xmin, xmax), lat=c(ymin, ymax))
coordinates(xymar) <- c("lon", "lat")
proj4string(xymar) <- CRS("+proj=longlat")
library(rgdal)
xymar <- spTransform(xymar, CRS("+proj=merc"))
xmin <- coordinates(xymar)[1, "lon"]; xmax <- coordinates(xymar)[2, "lon"]
ymin <- coordinates(xymar)[1, "lat"]; ymax <- coordinates(xymar)[2, "lat"]

```

```

pm10_longlat <- pm10[, c("경도", "위도")]
colnames(pm10_longlat) <- c("lon", "lat")
coordinates(pm10_longlat) <- c("lon", "lat")
proj4string(pm10_longlat) <- CRS("+proj=longlat")
pm10_merc <- spTransform(pm10_longlat, CRS("+proj=merc"))

library(fields)
fit1 <- Tps(coordinates(pm10_merc), pm10$pm10, lambda=0.0000001)
result1 <- predictSurface(fit1, grid.list = NULL, extrap = FALSE, nx = 200, ny = 200, drop.Z = TRUE)
fit2 <- Tps(coordinates(pm10_merc), pm10$pm2.5)
result2 <- predictSurface(fit2, grid.list = NULL, extrap = FALSE, nx = 200, ny = 200, drop.Z = TRUE)
fit3 <- Tps(coordinates(pm10_merc), pm10$no2, lambda=0.0000001)
result3 <- predictSurface(fit3, grid.list = NULL, extrap = FALSE, nx = 200, ny = 200, drop.Z = TRUE)
fit4 <- Tps(coordinates(pm10_merc), pm10$o3, lambda=0.0000001)
result4 <- predictSurface(fit4, grid.list = NULL, extrap = FALSE, nx = 200, ny = 200, drop.Z = TRUE)
fit5 <- Tps(coordinates(pm10_merc), pm10$so2)
result5 <- predictSurface(fit5, grid.list = NULL, extrap = FALSE, nx = 200, ny = 200, drop.Z = TRUE)
fit6 <- Tps(coordinates(pm10_merc), pm10$co, lambda=0.0000001)
result6 <- predictSurface(fit6, grid.list = NULL, extrap = FALSE, nx = 200, ny = 200, drop.Z = TRUE)

library(scales)
par(mar = c(0,0,0,0), xaxs = "i", yaxs = "i")
plot(c(xmin, xmax), c(ymin, ymax), type = "n", xlab = "", ylab = "")
rasterImage(outraster, xmin, ymin, xmax, ymax, interpolate = TRUE)
image(result1, add=TRUE, col=alpha(tim.colors(64), .7))
par(mar = c(0,0,0,0), xaxs = "i", yaxs = "i")
plot(c(xmin, xmax), c(ymin, ymax), type = "n", xlab = "", ylab = "")
rasterImage(outraster, xmin, ymin, xmax, ymax, interpolate = TRUE)
image(result2, add=TRUE, col=alpha(tim.colors(64), .7))

for(i in 1:6){
  par(mar = c(0,0,0,2), xaxs = "i", yaxs = "i")
  mat <- matrix(1:2, ncol=2)
  layout(mat, width=c(10,1))
  plot(c(xmin, xmax), c(ymin, ymax), type = "n", xlab = "", ylab = "", asp=1, axes=FALSE)
  #box()
  rasterImage(outraster, xmin, ymin, xmax, ymax, interpolate = TRUE)

  assign("result", eval(parse(text= paste("result", i, sep=""))))
  image(result, add=TRUE, col=alpha(tim.colors(64), .7), useRaster=TRUE)
  zrng <- range(result$z, na.rm=TRUE)
  zseq <- seq(zrng[1], zrng[2], , 64)
}

```

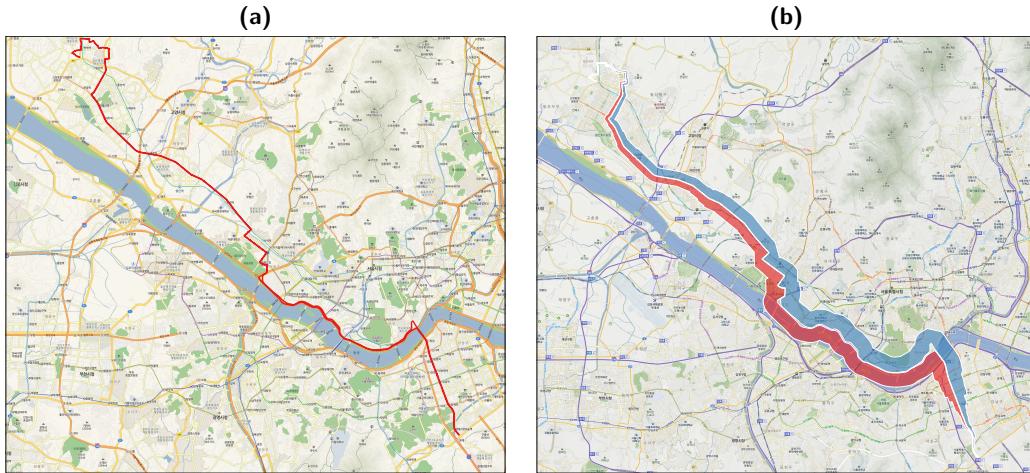


Figure 6: R을 이용하여 불러온 Daum과 Naver 지도 tile과 9711번 버스 노선도. (a)Naver 지도 , (b) Daum 지도

```
par(mar = c(6,0,6,3))
image(list(x=1, y=zseq, z=matrix(zseq, nrow=1)), useRaster=TRUE, axes=FALSE, col=tim.colors(64))
axis(4, cex.axis=3, padj=.5)
}
```

3.2 Daum과 Naver 지도

외국에서는 Google Map을 기반으로 한 다양한 패키지들이 개발되고 있으나 대한민국에서 많이 사용되는 Naver와 Daum 지도를 활용할 수 있는 R 용 패키지는 아직 개발이 되고 있지 않은 상황이다. 물론 Daum이나 Naver에서 제공하는 Open API를 활용하여 `RgoogleMaps`처럼 해상도에 따라 지도의 영역이 결정되는 매쉬업 함수를 쉽게 작성할 수 있다. 하지만 Open API를 이용하기 위해서는 각 사이트에 개발자 계정을 만들어 API 키를 발급받는 절차를 거쳐야 하며 한 API 키에 1일 요청 횟수에 제한이 걸려 있기 때문에 공개를 목적으로 하는 패키지의 개발에 Open API를 활용하기에는 제약이 있다.

실제 웹 브라우저를 통해 Daum이나 Naver 지도에 접속하면 open layer 기반으로 다양한 해상도의 여러 지도 tile을 조합하여 하나의 지도 서비스를 제공하는 것과 같이 R에서도 Daum이나 Naver 지도 tile들을 결합하여 다양한 해상도의 지도를 작성하는 것이 가능하다. 이미 이러한 방식으로 여러 GIS 프로그램에서 Daum이나 Naver 지도를 배경 이미지로 활용하고 있다. 현재 이러한 방식을 R에서 사용하기 위한 패키지를 개발 중이며, demo 함수를 <https://github.com/dongikjang>에서 다운받아 테스트 해볼 수 있다.

그림 6는 Daum과 Naver의 지도 tile들로 조합된 배경 지도 위에 경기도 일산에서 서울특별시 양재동으로 가는 9711번 버스 노선의 노선망과 2013년 3월 12일 하루 동안의 재차인원을 그린 그림을 보여주고 있다. 배경 지도를 위하여 Daum과 Naver에서 각각 가로 8개, 세로 9개, 총 72개의 지도 tile을 이용하였다.

그림 6을 위한 R 스크립트는 다음과 같다.

```
# Figure 6
if(curlstate){
  download.file(paste(gitadd2, "kssletter/master/9711.csv", sep=""), destfile="bus9711.csv",
                method="curl", extra="-L -k ", quiet=TRUE)
  bus9711 <- read.csv("bus9711.csv")
```

```

} else {
  library(RCurl)
  b9711csv <- getURL(paste(gitadd2, "kssletter/master/9711.csv", sep="")),
             cainfo = system.file("CurlSSL", "cacert.pem", package = "RCurl"))
  bus9711 <- read.csv(textConnection(b9711csv), stringsAsFactors = FALSE,
                      fileEncoding = "UTF-8", encoding = "UTF-8")
}

head(bus9711, 6)

#      x      y
# 1 126.7587 37.68823
# 2 126.7587 37.68825
# 3 126.7587 37.68825
# 4 126.7587 37.68823
# 5 126.7588 37.68826
# 6 126.7585 37.68837

library(RColorBrewer)
library(scales)
seoulmap <- qmap("seoul", zoom = 11, maptype = 'toner', source = 'stamen')
seoulmap + geom_path(mapping=aes(x = x, y = y), colour=brewer.pal(9, "Set1")[1], lwd=2, data = bus9711)

source_https(paste(gitadd2, "DaumMap/master/getDaummap.R", sep=""))
source_https(paste(gitadd2, "NaverMap/master/getNavermap.R", sep=""))

lon <- c(126.7405, 127.0398)
lat <- c(37.46889, 37.67667)
nmap <- getNaverMap(lon, lat, zoom=NA, mapproj = "Naver")
lon <- c(126.7405, 127.0398)
lat <- c(37.45889, 37.68667)
dmap <- getDaumMap(lon, lat, zoom=NA, mapproj = "Daum")

# (a)
plot(nmap)
lines(WGS842Naver(bus9711), col=brewer.pal(9, "Set1")[1], lwd=4)
# (b)
library(RColorBrewer)
library(scales)
source_https(paste(gitadd2, "kssletter/master/smartcardsource.R", sep=""))
if(curlstate){
  download.file(paste(gitadd2, "kssletter/master/9711vol.RData", sep=""),
                destfile="9711vol.RData",
                method="curl", extra="-L -k ", quiet=TRUE)
} else {

```

```

download.file.Bin(paste(gitadd2, "kssletter/master/9711vol.RData", sep=""),
                  "9711vol.RData")
}

load("9711vol.RData")

out <- pathvolpoly(res, pathvol, scl=1.2)
polygo <- out$polygo
polyback <- out$polyback
colval <- alpha(brewer.pal(9, "Set1"), .6)
plot(dmap)
polygon(polygo/2.5, col=colval[1], border="white")
polygon(polyback/2.5, col=colval[2], border="white")
rm(dmap, nmap)

```

4 shp 파일

shp 파일은 벡터(vector) 자료를 주고받을 때 가장 널리 사용되고 있는 자료 형태(format)이다. 이 파일 형태에 대한 정의는 ArcViewTM 와 ArcGISTM 개발사인 ESRITM에 의해서 상세하게 서술되어 있다(ESRI, 1998). Shp 파일은 자료를 표현하기 위하여 최소 3가지의 파일이 이용된다.

- *.shp: 도형의 기하학 정보를 저장하는 파일
- *.shx: 도형의 기하학 정보의 인덱스를 저장하는 파일
- *.dbf: 도형의 속성 정보를 제공하는 DBF III 파일. Feature table로서 표현

shp 포맷은 도형의 기하학적 정보와 속성정보 사이의 별도의 연결고리를 가지고 있지 않으며 shp 파일에 저장되어 있는 도형의 순서에 맞게 dbf에 그 속성이 저장되며 shx 파일에 저장되어 있는 위치 정보를 이용하여 shp 파일에 담긴 각 도형의 위치를 찾게 된다. 만약 같은 폴더에 *.proj이 있다면, 이 파일 안에는 Coordinate Reference System(CRS) 정보가 기록되어 있으며 이 파일은 지도의 투영법과 관련된 정보를 가지고 있다.

4.1 통계청 통계지리정보서비스(SGIS)

통계청 통계지리정보서비스(http://sgis.kostat.go.kr/statbd/statbd_03.vw)에서는 대한민국의 행정구역 shp 파일을 제공하고 있다. 시도 뿐만 아니라 시군구나 읍면동과 같은 자세한 행정구역 정보를 제공하고 있으면 이러한 shp파일은 SGIS 홈페이지에서 직접 다운로드하거나 Open API를 이용하여 다운로드할 수 있다. 또한 SGIS는 Open API를 통하여 인구주택총조사를 통해 조사된 각 집계구별 인구(인구총괄, 성/연령별, 교육정도별, 성/혼인상태별, 종교별), 가구(총가구, 평균가구원, 세대구성별, 젊유형태별, 난방시설별), 주택(주택총괄, 건축연도별, 주택유형별, 연건축면적 별) 등의 정보와 각 집계구의 영역을 KML과 geojson 파일로 제공하고 있다.

행정구역을 구분하는 shp파일을 사용하기 위하여 SGIS 홈페이지에서 2012년도 전국 시도 행정구역경계 파일([2012_1_0.zip](#))을 다운받아 압축을 풀게되면 이 폴더 안에는 temp.dbf와 temp.shp, temp.shx 파일이 들어있다. 이를 R을 통해 불러와 그림 8 (a)와 같이 전국적인 행정구역의 구분하는 지도를 작성할 수 있다. 또한 8 (b)처럼 인구주택 총조사에 활용되는 집계구와 같은 세부적인 구역정보와 Open API를 통해 제공되는 인구밀도($\text{명}/\text{km}^2$)를 결합하여 특정 지역의 사회경제지표를 나타내는 지도도 작성할 수 있다. 그림 8을 위한 R 스크립트는 다음과 같다.

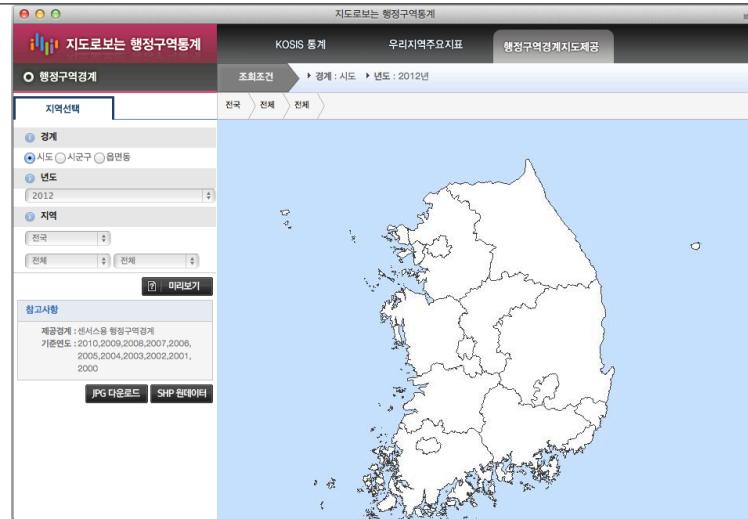


Figure 7: 통계청 통계지리정보서비스의 지도로 보는 행정구역 통계 홈페이지.

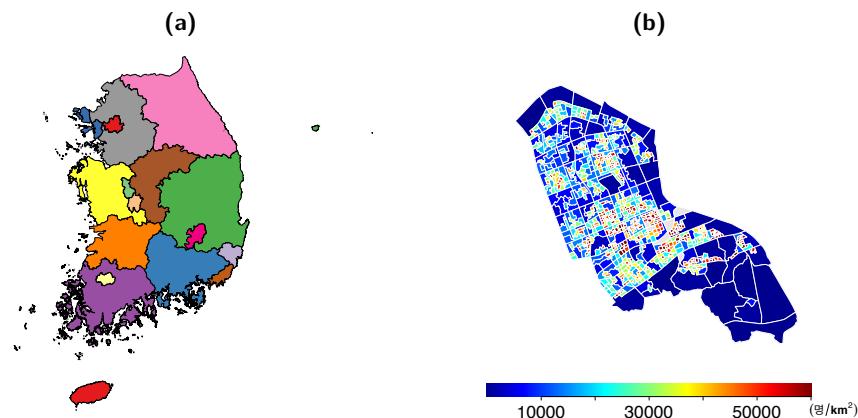


Figure 8: 통계청 통계지리정보서비스 제공하는 shp 파일. (a) shp 파일로 제공되는 대한민국 시도 행정구역경계, (b) Open API를 통해 제공되는 강남구 인구밀도 ($\text{명}/\text{km}^2$).

```
# Figure 8
library(maptools)
library(rgdal)
library(RColorBrewer)

# (a)
dir.create("2012_1_0")

if(curlstate){
  download.file(paste(gitadd2, "kssletter/master/2012_1_0/temp.shp", sep=""),
                destfile="2012_1_0/temp.shp", method="curl", extra=" -L -k ", quiet=TRUE)
  download.file(paste(gitadd2, "kssletter/master/2012_1_0/temp.dbf", sep=""),
                destfile="2012_1_0/temp.dbf", method="curl", extra=" -L -k ", quiet=TRUE)
}
```

```

    destfile="2012_1_0/temp.dbf", method="curl", extra=" -L -k ", quiet=TRUE)
download.file(paste(gitadd2, "kssletter/master/2012_1_0/temp.shx", sep=""),
              destfile="2012_1_0/temp.shx", method="curl", extra=" -L -k ", quiet=TRUE)
} else {
  download.file.Bin(paste(gitadd2, "kssletter/master/2012_1_0/temp.shp", sep=""),
                     "2012_1_0/temp.shp")
  download.file.Bin(paste(gitadd2, "kssletter/master/2012_1_0/temp.dbf", sep=""),
                     "2012_1_0/temp.dbf")
  download.file.Bin(paste(gitadd2, "kssletter/master/2012_1_0/temp.shx", sep=""),
                     "2012_1_0/temp.shx")
}

proj4val <- "+proj=tmerc +lat_0=38 +lon_0=127.0028902777778 +k=1 +x_0=200000 +y_0=500000
+ellps=bessel +units=m +no_defs
+towgs84=-115.80,474.99,674.11,1.16,-2.31,-1.63,6.43"
nc <- readShapePoly("2012_1_0/temp.shp", proj4string=CRS(proj4val))
nc <- spTransform(nc, CRS("+init=epsg:4326"))
cols <- c(brewer.pal(9, "Set1"), brewer.pal(7, "Accent"))
plot(nc, col=cols, lwd=.1, border=1)

# (b)
library(plyr)
val2col2 <- function(z, zlim, col = heat.colors(12), breaks){
  if(!missing(breaks)){
    if(length(breaks) != (length(col)+1)){stop("must have one more break than colour")}
  }
  if(missing(breaks) & !missing(zlim)){
    #breaks <- seq(zlim[1], zlim[2], length.out=(length(col)+1))
    breaks <- seq(zlim[1], zlim[2], length.out=(length(col)))
  }
  if(missing(breaks) & missing(zlim)){
    zlim <- range(z, na.rm=TRUE)
    breaks <- seq(zlim[1], zlim[2], length.out=(length(col)))
  }
  colorlevels <- col[((as.vector(z)-breaks[1])/range(breaks)[2]-range(breaks)[1])*(
    length(breaks)-1)+1]
  # assign colors to heights for each point
  colorlevels[z > zlim[2]] <- col[length(col)]
  colorlevels
}
library(data.table)

```

```

if(curlstate){

  download.file(paste(gitadd2, "kssletter/master/census2010_den_wgs1984.RData", sep=""),
                destfile="census2010_den_wgs1984.RData", method="curl", extra="-L -k ", quiet=TRUE)
} else {

  download.file.Bin(paste(gitadd2, "kssletter/master/census2010_den_wgs1984.RData", sep=""),
                    destfile="census2010_den_wgs1984.RData")
}

load('census2010_den_wgs1984.RData')
name <- unlist(lapply(out, function(x) x$name))
ind <- which(substring(name, 1, 5) == 11230)
out2 <- out[ind]

dir.create("2012_2_11230")
if(curlstate){

  download.file(paste(gitadd2, "kssletter/master/2012_2_11230/temp.shp", sep=""),
                destfile="2012_2_11230/temp.shp", method="curl", extra="-L -k ", quiet=TRUE)
  download.file(paste(gitadd2, "kssletter/master/2012_2_11230/temp.dbf", sep=""),
                destfile="2012_2_11230/temp.dbf", method="curl", extra="-L -k ", quiet=TRUE)
  download.file(paste(gitadd2, "kssletter/master/2012_2_11230/temp.shx", sep=""),
                destfile="2012_2_11230/temp.shx", method="curl", extra="-L -k ", quiet=TRUE)
} else {

  download.file.Bin(paste(gitadd2, "kssletter/master/2012_2_11230/temp.shp", sep=""),
                    destfile="2012_2_11230/temp.shp")
  download.file.Bin(paste(gitadd2, "kssletter/master/2012_2_11230/temp.dbf", sep=""),
                    destfile="2012_2_11230/temp.dbf")
  download.file.Bin(paste(gitadd2, "kssletter/master/2012_2_11230/temp.shx", sep=""),
                    destfile="2012_2_11230/temp.shx")
}

mat <- matrix(1:2, nrow=2)
layout(mat, height=c(9,2))
par(mar=c(1,4,0,2), family="NanumGothic")
proj4val <- "+proj=tmerc +lat_0=38 +lon_0=127.0028902777778 +k=1 +x_0=200000 +y_0=500000
+ellps=bessel +units=m +no_defs
+towgs84=-115.80,474.99,674.11,1.16,-2.31,-1.63,6.43"
nc <- readShapePoly("2012_2_11230/temp.shp", proj4string=CRS("+init=epsg:2097"))
nc <- spTransform(nc, CRS("+init=epsg:4326"))
plot(nc, col=gray(.9), border=gray(.9))
zlim <- range(unlist(lapply(out, function(x) x[[3]]))), na.rm=TRUE)
zlim[2] <- 60000
cols <- tim.colors(64) # colorRampPalette(c("green", "red"), space="rgb")(64)

```

```

l_ply(out2, function(x) plot(x$Geometry, col=alpha(val2col2(x$popdensity, zlim, col=cols), 1),
                            add=TRUE, border="white", lwd=.1))

par(mar=c(4,8,3.2,8))
image(matrix(seq(0,1,,200)), col=tim.colors(200), axes=FALSE, useRaster=TRUE)
seqval <- seq(floor(zlim[1]), ceiling(zlim[2]))
labval <- seqval[!duplicated(floor(seqval/10000)*10000)]
atval <- (labval - floor(zlim[1]))/(ceiling(zlim[2])-floor(zlim[1]))
labval <- labval[-1]
labval[length(labval)] <- paste(">", ceiling(zlim[2]), sep="")
atval <- atval[-1]
axis(1, at=atval, labels=labval, cex.axis=3, padj=.7)
mtext(expression(group("(", bold(명/km^2), ")")), side=1, line=2.1, at=1.07, cex=2)

```

SGIS가 국가에서 생산된 공식 행정구역계 shp 자료를 배포하는 곳 이라면 GIS 분석 커뮤니티에서 출발한 biz-gis(<http://www.biz-gis.com>)에서는 다양한 shp 파일들과 관련된 여러 가지 tool들을 제공하고 있으며 ESRITM 또한 다양한 shp 파일들을 제공하고 있다. 그리고 ITS 국가교통정보센터(<http://www.its.go.k>)에서는 전국의 도로망을 ITS 표준노드 링크의 shp 파일로 제공하고 있으며, 한국교통연구원 국가교통DB센터(<http://ktdb.go.kr>)에서는 교통시설물, 철도망, 교통존, 도로망, 도로경계 등의 shp 파일을 제공하고 있다. 그림 9는 국가교통DB센터에서 제공하는 서울 수도권 지하철 shp 파일과 통계청 SGIS에서 제공하는 서울특별시 행정구역 경계 shp 파일을 결합하여 작성된 서울-수도권 지하철 노선도이다.

```

# Figure 9 (based on Figure 8)

if(curlstate){

  download.file(paste(gitadd2, "kssletter/master/seoul_subway2.R", sep=""),
                destfile="seoul_subway2.R", method="curl", extra="-L -k ", quiet=TRUE)

} else {

  download.file.Bin(paste(gitadd2, "kssletter/master/seoul_subway2.R", sep=""),
                    destfile="seoul_subway2.R")

}

if(Encoding("a") == "unknown" & .Platform$OS.type =="windows"){

  if(curlstate){

    download.file(paste(gitadd2, "kssletter/master/SeoulSubwayShp2.zip", sep=""),
                  destfile="SeoulSubwayShp.zip", method="curl", extra="-L -k ", quiet=TRUE)

  } else {

    download.file.Bin(paste(gitadd2, "kssletter/master/SeoulSubwayShp2.zip", sep=""),
                      destfile="SeoulSubwayShp.zip")

  }

} else {

  if(curlstate){

    download.file(paste(gitadd2, "kssletter/master/SeoulSubwayShp.zip", sep=""),
                  destfile="SeoulSubwayShp.zip", method="curl", extra="-L -k ", quiet=TRUE)

  } else {

    download.file.Bin(paste(gitadd2, "kssletter/master/SeoulSubwayShp.zip", sep=""),
                      destfile="SeoulSubwayShp.zip")

  }

}

```

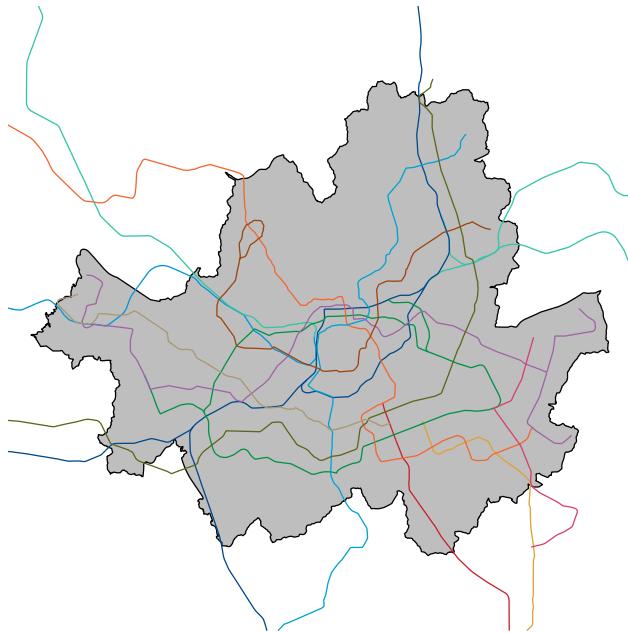


Figure 9: 한국교통연구원 국가교통DB센터에서 제공하는 shp 파일을 R에서 불러와 나타낸 서울-수도권 지하철 노선도.

```

}

}

unzip("SeoulSubwayShp.zip", exdir="SeoulSubwayShp")
shpfolder <- "SeoulSubwayShp"

library(maptools)
library(rgdal)

proj4val <- "+proj=tmerc +lat_0=38 +lon_0=127.0028902777778 +k=1 +x_0=200000 +y_0=500000
+ellps=bessel +units=m +no_defs
+towgs84=-115.80,474.99,674.11,1.16,-2.31,-1.63,6.43"
nc <- readShapePoly("2012_1_0/temp.shp", proj4string=CRS(proj4val))
nc <- spTransform(nc, CRS("+init=epsg:4326"))
nc$name <- as.factor(iconv(as.character(nc$name), "EUC-KR", "UTF-8"))
ncs <- subset(nc, name=="서울특별시")

plot(ncs, col=gray(.7))
source("seoul_subway2.R", encoding="UTF-8")

```

4.2 OpenStreetMap

편집이 가능한 전 세계 지도 자료 작성의 목적으로 2004년 영국에서 설립된 OpenStreetMap(OSM) (OSM Foundation, 2014) 프로젝트에서는 전 세계 다양한 format의 지도 자료를 제공하고 있다. OSM은 Google이나 Naver, Daum 지도와 같이 raster 이미지 형태의 지도를 제공할 뿐만 아니라 지도를 구성하고 있는 다양한 layer 또한 함께 다운로드 받을 수 있도록 하고 있어서

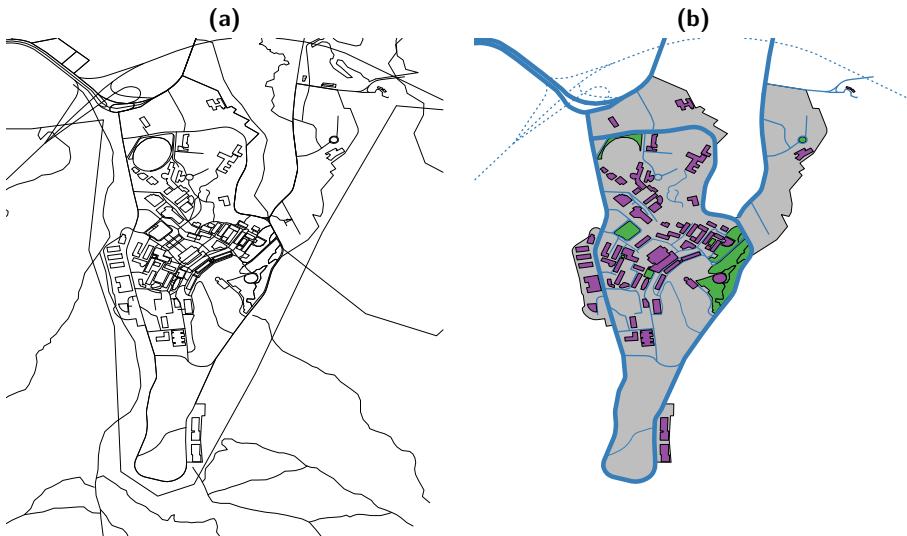


Figure 10: OpenStreetMap에서 추출한 서울대학교 지도. (a) OSM 원자료, (b) OSM 원자료를 특성값에 따라 구분하여 표출한 지도.

사용자들이 자연스럽게 변화된 구간을 업데이트하거나 오류를 수정할 수 있도록 하였다. 대한민국의 많은 영역의 지도들이 OMS 프로젝트에 참여하는 개별 사용자의 자발적 기여에 의해서 자료가 풍부해지고 있지만 아직 국가나 Naver나 Daum 등 민간 지도관련 회사에서 구축한 자료에 비해서는 열악한 게 현실이기도 하다.

앞에 소개한 패키지 `ggmap`에서 `source`를 ‘osm’으로 지정하면 OMS를 배경 layer로 사용하여 그 위에 다양한 layer를 추가할 수 있다. 또한 2014년 10월 현재 서울, 런던, 파리, 뉴욕 등 전 세계의 295개의 주요 도시에 대해서는 다양한 형태의 shp 파일을 제공하고 있다(<https://mapzen.com/metro-extracts/>).

OpenStreetMap을 R에서 활용하기 위해서는 개별 도시의 shp파일을 다운받아 사용하거나 패키지 `osmar` (Schlesinger and Eugster, 2013)을 이용하여 지도 자료를 OSM 서버에 직접 접속하여 다운로드 받을 수 있다. 아래 R 스크립트는 서울 지역 OSM 파일을 다운로드하여 R에서 불러오는 예이다.

```
# Figure 10 (a)
library(osmar)
api <- osmsource_api()
bbox <- corner_bbox(126.9450, 37.445, 126.963, 37.470)
osmmap <- get_osm(bbox, source = api)
plot(bbox[c(1,3)], bbox[c(2,4)], asp=1, xlab="", ylab="", type="n", axes=FALSE)
plot_ways(osmmap, add=TRUE)
```

OSM에서 제공하는 파일은 XML 파일 형태로 만들어져 있기 때문에 이를 `osmar` 패키지에 있는 `get_osm` 함수를 이용하여 R에서 사용이 가능한 형태로 변형하게 된다. 하지만 요소 각각에 대해서 `for` 문을 이용해서 반복이 행해지기 때문에 넓은 영역의 변환에는 매우 많은 시간이 걸리게 된다. 이러한 경우에는 `IMPOSM SHP` 파일을 다운로드 받아 각각의 shp 파일을 R에서 불러와서 사용하는 것이 좀 더 효과적이다. OSM에서 제공하는 주요 도시가 아닌 경우에는 OSM API를 이용하여 특정 영역을 다운로드 받을 수 있다. 단 한 번에 받을 수 있는 영역의 크기에 대한 제한이 있다. 그림 10 (a)는 패키지 `osmar`을 이용하여 다운로드 받은 서울대학교 부근의 지도이다. `get_osm` 함수는 `corner_bbox`를 통해 설정된 지역의 모든 OSM 자료를 다운 받게 되므로 그림 10 (a)처럼 건물, 도로, 등산로, 녹지 공간 등 지도를 구성하는 다양한 정보들을 포함하고 있다.

그림 10 (b)과 같이 OSM 자료에서 특정 key나 value 만을 만족하는 요소들을 추출하여 다양한 형태의 지도를 그릴 수

있다. 아래 R 스크립트처럼 `find`와 `find_down`, `subset` 함수를 이용하면 특정 요소만으로 구성된 node나 way를 추출할 수 있게 된다.

```
# Figure 10 (b)

buildingid <- find(osmmap, way(tags(id == 185936759 | k == "building" | v == "농업생명과학대학 (200)")))
ids <- find_down(osmmap, way(buildingid))
building <- subset(osmmap, ids=ids)
road1id <- find(osmmap, way(tags(v=="motorway" | v=="motorway_link")))
ids <- find_down(osmmap, way(road1id))
road1 <- subset(osmmap, ids=ids)
road2id <- find(osmmap, way(tags(k=="highway" & v=="secondary")))
ids <- find_down(osmmap, way(road2id))
road2 <- subset(osmmap, ids=ids)
road3id <- find(osmmap, way(tags(k=="highway" & v=="residential")))
ids <- find_down(osmmap, way(road3id))
road3 <- subset(osmmap, ids=ids)
road4id <- find(osmmap, way(tags(k=="highway" & (v=="tertiary" | v=="tertiary_link"))))
ids <- find_down(osmmap, way(road4id))
road4 <- subset(osmmap, ids=ids)
road5id <- find(osmmap, way(tags(k=="highway" & v=="living_street")))
ids <- find_down(osmmap, way(road5id))
road5 <- subset(osmmap, ids=ids)
parkid <- find(osmmap, way(tags((v=="wood" | v=="park" | v=="grass") &
                                id != 290466559 & id != 206049084)))
ids <- find_down(osmmap, way(parkid))
park <- subset(osmmap, ids=ids)
areaid <- find(osmmap, way(tags(v=="university" & id == 173556800)))
ids <- find_down(osmmap, way(areaid))
area <- subset(osmmap, ids=ids)

library(maps)
inpoly <- function(xx){
  any(maps:::in.polygon(as_sp(area, "polygons")@polygons[[1]]@Polygons[[1]]@coords,
                        xx@Polygons[[1]]@coords))
}

ids <- unlist(lapply(as_sp(park, "polygons")@polygons, inpoly))
park <- subset(as_sp(park, "polygons"), ids)

library(sp)
cols <- brewer.pal(9, "Set1")
plot(bbox[c(1,3)], bbox[c(2,4)], asp=1, xlab="", ylab="", type="n", axes=FALSE)
plot(as_sp(area, "polygons"), add=TRUE, col="grey")
plot(park, add=TRUE, col=cols[3])
```

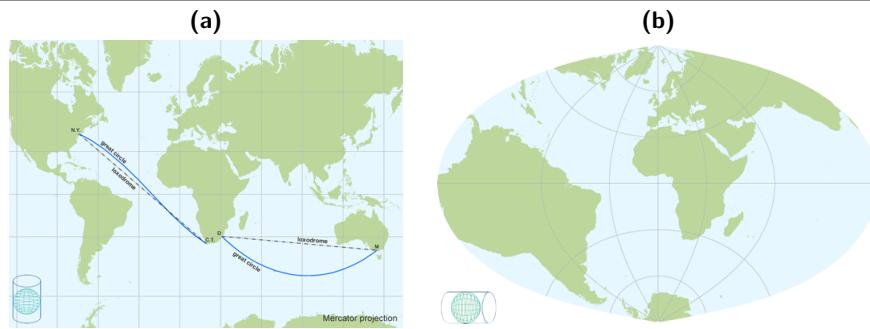


Figure 11: Projection 방법에 따른 세계지도 예시. (a) Mercator projection, (b) Transverse Mercator projection. 출처 : <http://kartoweb.itc.nl/geometrics/map%20projections/mappro.html>

```
plot_ways(road1, add=TRUE, lwd=2, col=cols[2], lty=3)
plot_ways(road2, add=TRUE, lwd=7, col=cols[2])
plot_ways(road3, add=TRUE, lwd=2, col=cols[2])
plot_ways(road4, add=TRUE, lwd=2, col=cols[2])
plot_ways(road5, add=TRUE, lwd=1, col=cols[2])
plot(as_sp(building, "polygons"), col = cols[4], add=TRUE)
```

그리고 패키지 `osmar`를 이용하여 추출한 도로망 정보와 패키지 `igraph` (igraph Developement Group, 2014)의 `netwrok` 분석 tool을 결합하면 두 지점사이의 최단거리를 구할 수 있다. 그러나 현재 대한민국의 영역의 OSM은 통일된 체계를 갖추고 있지 않고 많은 도로망들이 서로 단절되어 있어 연결된 도로망만을 이용하여 최단거리를 계산하는 오류가 있다. 패키지 `osmar`에 대한 자세한 설명은 Eugster and Schlesinger (2013)을 보기 바란다.

4.3 Ellipsoid 와 Projection

지구가 평면이 아닌 구의 형태를 이루기 때문에 이를 평면으로 표현하기 위한 다양한 방법들이 제안되어 왔다. 그러나 지구는 적도반지름이 극반지름보다 약간 긴 타원체이며, 실제 지표면은 그 형상이 매우 불규칙하기 때문에 일정한 규칙에 따라 지도를 제작하기 위하여 지표면의 수학적 모델인 지오이드(Geoid)와 가장 유사한 수학적 타원체를 가정하게 된다. 이를 지구타원체(Earth Ellipsoid)라고 한다. 현재 지구에 가장 적합한 지구타원체인 GRS80타원체와 WGS84타원체를 국제표준 타원체로 결정하여 GPS와 같이 전 세계를 대상으로 하는 좌표계에 사용되고 있다. 대한민국은 일본이 측량해 놓은 지도 체계를 기반으로 하여 지도가 제작되었기에 최근까지 Bessel 1941 타원체를 지구타원체로 사용하였으나 현재 세계적 표준인 GRS80이 WGS와 같은 타원체로 대체하고 있다. 하지만 수학적 타원체인 지구타원체는 지구의 모든 지역을 표현하기 위하여 설정이 되었기 때문에 지역별로 오차가 발생하게 된다. 이를 보정하기 위하여 각 지역별로 해당 지역의 지오이드 면에 적합한 지구타원체를 정의하여 사용하고 있다.

타원체는 여전히 3차원 좌표계나 위경도 좌표계를 등으로 위치를 나타내므로 이를 평면을 표현하기 위하여 투영(projection) 과정을 거친다. 이 투영 과정을 거쳐 지도는 또다시 여러 지도체계로 세분되게 된다. 현재 대한민국에서는 원통형 투영법인 merc 투영법(Mercator projection)나 tmrc 투영법(또는 tm: Transverse Mercator projection)과 같은 투영법이 널리 사용되고 있다. 그림 11은 전 세계 지도를 Mercator과 Transverse Mercator 투영법으로 제작한 지도를 보여 주고 있다. Mercator 투영법은 원통을 적도와 접하도록 수직으로 세우고 지구를 이 원통에 투영하는 방법이고 Transverse Mercator 투영법은 원통은 지구 자오선과 접하도록 수평으로 높고 지구를 투영하는 방법이다. 대한민국과 같이 국토가 남북으로 긴 영역을 지도로 표현하기 위해서는 주로 Transverse Mercator 투영법이 사용되고 있다.

앞에서 언급한 shp 파일들도 다양한 형태의 projection 값을 가지고 있다. 현재 통계청 SGIS에서 제공하는 행정구역경계

Table 1: 대한민국 주요 좌표계 EPSG 코드 및 Proj4 인자

Coordinate Reference System	Authority ID	Proj4 Parameter
WGS84	EPSG:4326	+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs
GRS80	EPSG:4019	+proj=longlat +ellps=GRS80 +no_defs
Google Mercator	EPSG:3857	+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null +wktext +no_defs
Korean 1985 / Korea Central Belt	EPSG:2097	+proj=tmerc +lat_0=38 +lon_0=127 +k=1 +x_0=200000 +y_0=500000 +ellps=bessel +units=m +no_defs +towgs84=-145.907,505.034,685.756,-1.162,2.347,1.592,6.342
Korean 1985 / Korea Central Belt (Adjusted)	EPSG:5174	+proj=tmerc +lat_0=38 +lon_0=127.00289 +k=1 +x_0=200000 +y_0=500000 +ellps=bessel +units=m +no_defs +towgs84=-145.907,505.034,685.756,-1.162,2.347,1.592,6.342
Korean 1985 / Unified CS (Road name address)	EPSG:5178	+proj=tmerc +lat_0=38 +lon_0=127.5 +k=0.9996 +x_0=1000000 +y_0=2000000 +ellps=bessel +units=m +no_defs +towgs84=-145.907,505.034,685.756,-1.162,2.347,1.592,6.342
Korea 2000 / Unified CS (Naver Map)	EPSG:5179	+proj=tmerc +lat_0=38 +lon_0=127.5 +k=0.9996 +x_0=1000000 +y_0=2000000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
Korea 2000 / Central Belt (Daum Map)	EPSG:5181	+proj=tmerc +lat_0=38 +lon_0=127 +k=1 +x_0=200000 +y_0=500000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
Korea 2000 / Central Belt 2010 (NGII standard)	EPSG:5186	+proj=tmerc +lat_0=38 +lon_0=127 +k=1 +x_0=200000 +y_0=600000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs

shp파일은 한국측지계 동경 127도, 북위 38도를 기준으로 하는 TM중부원점 좌표계를 사용하고 있다(통계청은 2015년 상반기부터는 세계측지계 기준으로 변경한다고 예고하고 있다).

R 패키지 `sp` (Pebesma et al., 2014)와 `rgdal` (Bivand et al., 2014)는 PROJ.4 projection system를 활용한 CRS 설정용 함수 `CRS`를 제공하고 있다. 또한 `spTransform` 함수를 이용하여 좌표계 간 변환이 가능하다. 현재 다양한 영역에서 각자의 CRS를 이용하여 지도를 제작하고 있기 때문에 R에서 shp 파일을 활용하기 위해서는 주의가 필요하다. 많은 GIS 프로그램에서는 shp 파일이 있는 폴더에 같은 이름을 가진 `*.proj` 파일이 있으면 해당 shp 파일에 해당하는 좌표계를 자동으로 인식하나 R에서는 `*.proj` 파일이 있더라도 Proj4 인자를 다시 정확하게 선언을 해주어야 한다. 표 1은 대한민국에서 널리 사용되고 있는 주요 좌표계와 Proj4 인자값을 보여주고 있다. 앞에서 언급한 Naver와 Daum과 같은 지도도 각자의 좌표계를 가지고 있으므로 이들 지도 위에서 통계량들을 중첩하여 표현하기 위해서는 각자의 좌표계로 변환을 해주어야 한다.

4.4 Node와 Link

지구를 평면의 지도로 표현하기 위한 다양한 타원체와 투영법이 있는 것처럼 그 지도를 구성하는 하나의 요소인 도로와 철도 등의 교통 네트워크의 표출 방식에서 다양한 방식이 존재하며 이러한 방식을 GIS에서는 node와 link 체계라고 한다. 예를 들어 도로의 차선 수나 제한속도가 변하는 지점이나 교차로와 같이 다른 길과 만나는 지점을 node라 하고 이들을 잇는 선들을 link라고 한다. 따라서 DB를 구축하는 각각의 목적에 따라 다양한 속성을 반영하는 node/link 체계가 존재한다. 따라서 점차 교통정보의 제공범위가 넓어지고 있으며 서로 다른 node/link 체계는 상호 호환이 거의 불가능하기 때문에 교통정보 관리주체 간 상호 교통정보 교환을 위하여 2004년부터 표준지도(전국 표준 node/link)를 구축하여 활용하고 있다. 표준 node/link는 shape 형태의 파일로 제작되어 있으며 ITS 국가교통정보센터(<http://www.its.go.kr/>)를 통하여 배포되고 있다.

차량이나 스마트폰의 GPS를 통해 구해지는 개별 이동 궤적은 각 지점의 좌표값 만을 알려주기 때문에 특정 사용자가 어떠한 길을 따라 이동하는지를 알아보기 위하여 각 지점을 표준 node/link 체계의 개별 link에 할당을 하는 과정을 거쳐 실제 어떤 network를 따라 이동하는지를 계산하게 된다. 이러한 link들에 평균 통행시간, 통행속도, 교통량 등 다양한 속성 정보들을 할당하여 차량의 최적경로와 검색이나 구간별 평균 속도 계산과 같은 영역에 활용이 되고 있다.

그림 12는 국가교통DB센터에서 구축한 서울특별시 관악구의 주중/주말의 혼잡지도를 보여주고 있다. 혼잡지도 구축을

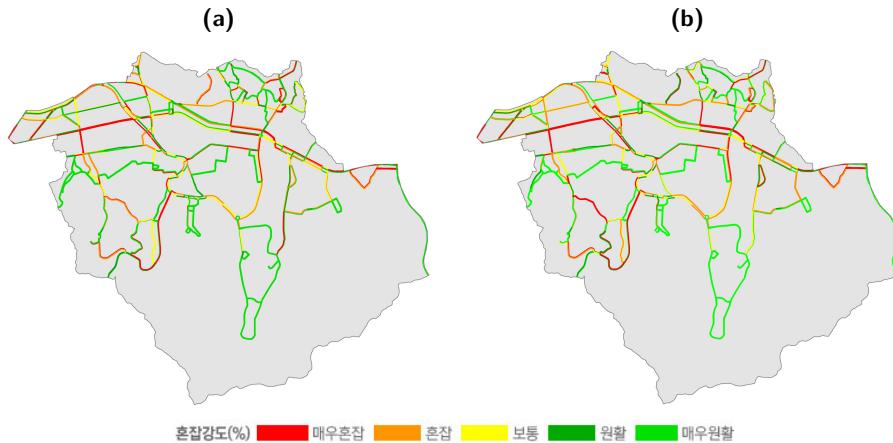


Figure 12: 각 link 통과 차량들의 내비게이션 속도 정보를 이용하여 구축한 서울특별시 관악구 혼잡지도 (a) 주중, (b) 주말 (출처: 국가교통DB센터, <http://ktedb.go.kr>).

위하여 각 link 통과 차량들의 내비게이션 속도 정보를 이용하여 속도를 관측값으로 하는 분포함수를 만들었다. 그리고 이를 기반으로 혼잡과 정상 속도를 나누는 군집분석을 활용하여 link 통과차량 중 혼잡 속도 기준 이하를 경험한 차량의 비율을 혼잡강도로 정의하여 link별 혼잡지도를 구축하였다.

5 Interactive map

R의 graphic은 논문과 같은 출판물에 삽입되는 고해상도의 static 이미지 제작을 주된 목적으로 하고있기 때문에 Flash나 Javascript을 이용하여 만들어진 interactive graphic을 만들기에는 많은 한계가 있었다. 이러한 문제를 해결하기 위하여 최근에 Javascript의 interactive visualizations을 만들어내기 위하여 R과 친숙한 lattice 스타일의 plotting interface를 가진 **rChart** (Thomas et al., 2013)나 **rMaps** (Vaidyanathan, 2014), **googleVis** (Gesmann et al., 2014)와 같은 패키지가 제안이 되었다. 이들 중 패키지 **rChart**는 최신의 open-source JavaScript 라이브러리인 **Leaflet** (Leaflet, 2014)을 이용할 수 있도록 지원하여 R에서 interactive dynamic 지도 제작이 가능해졌다. 그리고 최근에 개발된 패키지 **rMaps** (Jang, 2014)은 대한민국에서 널리 사용되는 Daum 과 Naver, VWorld에서 제공되고 있는 지도를 활용한 interactive map을 R에서 사용이 가능하도록 하였다. 그리고 RStudio (RStudio, 2014)에서 제작한 패키지 **shiny** (RStudio, Inc., 2014)를 이용하여 R에서 바로 interactive한 web applications을 개발할 수 있게 되었으며, Shiny Server를 이용하여 R이 설치된 서버에서 웹 서비스도 제공할 수 있다. 그림 13는 패키지 **shiny**와 **rCharts**를 이용하여 구축된 Bike Sharing Systems (<http://glimmer.rstudio.com/ramnathv/BikeShare/>)을 응용하여 interactive map 테스트 웹 서버에 구축된 경기도 고양시 공공자전거 스테이션 현황과 전국 교통량 조사지점과 일교통량 웹 페이지들이다. 패키지 **shiny**를 활용한 웹 어플리케이션 개발에 관한 자세한 내용은 Beeley (2013)을 보기 바란다.

6 마치며

이 글을 쓰게 된 주된 이유는 2006년 수행했던 프로젝트의 기억 때문이다. 당시 서울/수도권 아파트 평당 가격을 조사하여 이들의 시공간 패턴을 분석하여 표출하기 위하여 서울-수도권 지역의 행정구역경계 자료가 필요했다. 오랜 검색을 통해 겨우 찾아낸 서울특별시의 행정구역경계 자료가 불완전하여 특정 구들의 경계는 포털 사이트의 지도를 일일이 확인하며 경계 지점을 마우스로 하나씩 클릭하여 좌표를 확인하는 과정을 통해 지도를 완성하는 과정을 거쳐야 했다. 8년여가 지난 오늘날에는 R 사용자들이 급속히 넓어지면서 통계 뿐만 아니라 다양한 영역에서 이를 활용한 이들이 기여로 인해 많은 패키들이 개발이 이루어지고 있다. 이와 함께 R에서 지도를 사용하는 다양한 방법들 또한 제안이 되고 있다. 이를 활용하면 공간통계학 뿐만

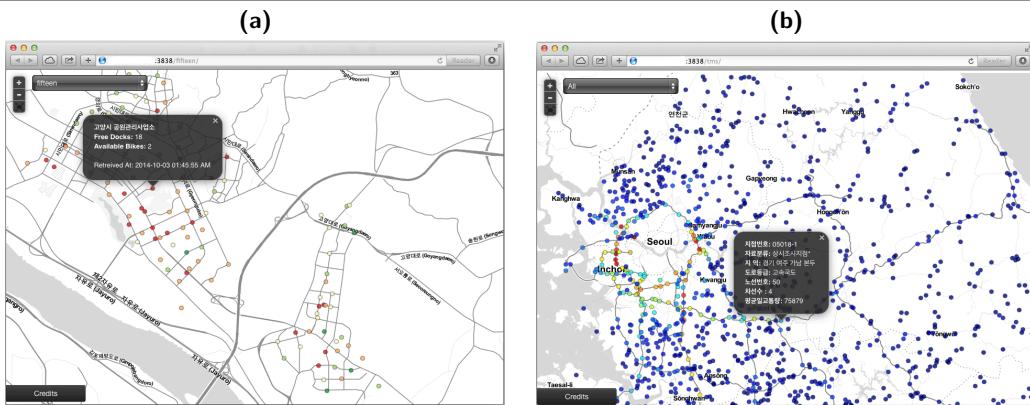


Figure 13: R 패키지 `shiny`를 이용한 interactive map. (a) 경기도 고양시 공공자전거서비스 스테이션 및 자전거 대여 현황, (b) 전국 교통량 조사지점 및 2013년 (연)평균 일교통량

아니라 다양한 통계학의 영역에서 좀 더 효과적인 성과를 이룰 수 있으리라 본다. 마지막으로 여기에 수록된 그림들을 위한 R 스크립트는 <https://github.com/dongikjang/kssletter>에서 다운로드 받을 수 있다.

참고문헌

Daniel Adler, Duncan Murdoch, Oleg Nenadic, Simon Urbanek, Ming Chen, Albrecht Gebhardt, Ben Bolker, Gabor Csardi, Adam Strzelecki and Alexander Senger (2014). **rgl**: *3D visualization device system (OpenGL)*, URL <http://CRAN.R-project.org/package=rgl>. R package version 0.94.1143.

Richard A. Becker, Allan R. Wilks and Ray Brownrigg (2014). **mapdata**: *Extra Map Databases*, URL <http://CRAN.R-project.org/package=mapdata>. R package version 2.2-3.

Chris Beeley (2013). *Web Application Development with R Using Shiny*. Packt Publishing, Birmingham.

Roger Bivand, Tim Keitt, Barry Rowlingson, Edzer Pebesma, Michael Sumner, Robert Hijmans and Even Rouault (2014). **rgdal**: Bindings for the Geospatial Data Abstraction Library, URL <http://CRAN.R-project.org/package=rgdal>, R package version 0.9-1.

ESRI (1998). *ESRI Shapefile Technical Description*, URL <http://dl.maptools.org/dl/shapelib/shapefile.pdf>.

Manuel J. A. Eugster and Thomas Schlesinger (2013), osmar: OpenStreetMap and R. *R Journal*, **5**, 53–63.

igraph Development Group (2014). **graph**: *Network analysis and visualization*, URL <http://CRAN.R-project.org/package=igraph>, R package version 0.7.1.

Markus Gesmann, Diego de Castillo and Joe Cheng (2014). **googleVis**: Interface between R and Google Charts. URL <http://www.googlevis.org>.

[//CRAN.R-project.org/package=googleVis](http://CRAN.R-project.org/package=googleVis) R package version 0.5.5

David Kahle and Hadley Wickham (2013). **ggmap**: *A package for spatial visualization with Google Maps and OpenStreetMap*, URL <http://CRAN.R-project.org/package=ggmap>. R package version 2.3.

Leaflet (2014). Leaflet. An Open-Source JavaScript Library for Mobile-Friendly Interactive Maps, URL <http://leafletjs.com/>.

Markus Loecher (2014). **RgoogleMaps**: Overlays on Google Map Tiles in R, URL <http://CRAN.R-project.org/package=RgoogleMaps>. R package version 1.2.0.6.

Dongik Jang (2014). **rMapsKr**: Interactive South Korea Maps using Leaflet Library, URL <https://github.com/dongikjang/rMapsKr>. R package version 0.0.1.

- Pavel Michna (2014). **RNetCDF**: *R Interface to NetCDF Datasets*, URL <http://CRAN.R-project.org/package=RNetCDF>, R package version 1.6.3-1.
- OSM Foundation (2014). *The OpenStreetMap Project*, URL <http://openstreetmap.org>.
- Edzer Pebesma, Roger Bivand, Barry Rowlingson and Virgilio Gomez-Rubio (2014). **sp**: *classes and methods for spatial data*, URL <http://CRAN.R-project.org/package=sp>, R package version 1.0-15.
- David Pierce (2014) . **ncdf**: *Interface to Unidata netCDF data files*, URL <http://CRAN.R-project.org/package=ncdf>, R package version 1.6.8.
- David Pierce (2014) . **ncdf4**: *Interface to Unidata netCDF (version 4 or earlier) format data files*, URL <http://CRAN.R-project.org/package=ncdf4>, R package version 1.13.
- RStudio (2014). RStudio, Inc, URL <http://www.rstudio.com>.
- RStudio, Inc. (2014). **shiny**: *Web Application Framework for R*, URL <http://www.rstudio.com/shiny/>, R package version 0.101.
- Thomas Schlesinger and Manuel J. A. Eugster (2013). **osmar**: *OpenStreetMap and R*, URL <http://CRAN.R-project.org/package=osmar>, R package version 1.1-7.
- Reinholdsson Thomas, Russell Kenton and Vaidyanathan Ramnath (2013). **rCharts**: *Interactive Charts using Javascript Visualization Libraries*, URL <http://rcharts.io/>, R package version 0.4.5.
- Ramnath Vaidyanathan (2014). **rMaps**: *Interactive Maps from R*, URL <http://rmaps.github.io>, R package version 0.1.1.
- Hadley Wickham and Winston Chang (2013). **ggplot2**: *An implementation of the Grammar of Graphics*, URL <http://CRAN.R-project.org/package=ggplot2>, R package version 1.0.0.