

Lab 4：地理空间数据和空间分析

什么是空间分析？

空间分析是针对空间数据的数据分析技能。空间数据也被称为地理数据或地理信息，是指与特定地理位置相关联的信息。这类数据通常包含两个主要组成部分：位置（经纬度坐标、街道地址等）和属性（与该位置关联的描述性信息）。空间数据是空间分析的基础。空间分析允许我们对空间数据进行复杂的处理和查询。这种类型的数据分析可以帮助我们理解地理现象和要素在空间内存在的模式、趋势和关系，进而对地球科学、生态农业、公共卫生、城市规划、资源管理、交通运输、建设选址等领域提供决策支持。

空间分析可能包括以下几个方面：

- 叠加分析：通过将不同的空间数据（地图图层）叠加在一起，找出共享了某些属性的区域。
- 几何计算：计算地理要素的面积、长度、距离等参数。
- 缓冲区分析：在一个特定的范围内，探寻地理位置、要素或事件的分布情况。
- 表面分析：对空间中的连续数据场进行分析和展示，比如利用数字高程模型显示海拔变化。
- 网络分析：对网络状的线性数据（路网、河道网络、人口流动等）进行分析，常用于优化路线等问题。
- 地统计学：使用统计方法分析地理数据的分布模式，包括热点分析、聚类分析、地理加权回归分析等。

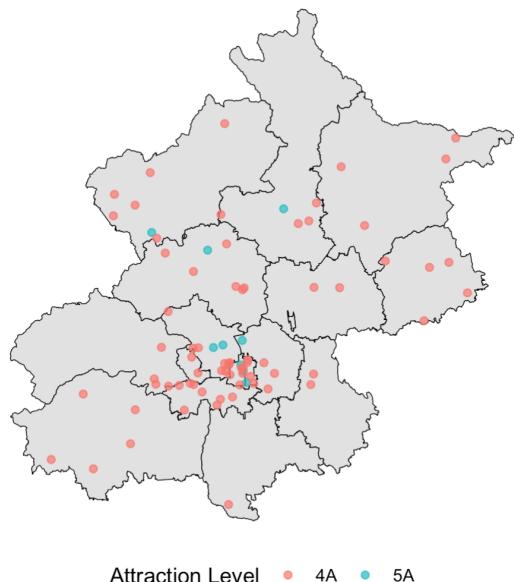
以上方面都可以用 R 语言实现，并进行相应的可视化呈现。本课程将介绍叠加分析、几何计算、缓冲区分析和表面分析的 R 语言基本操作。网络分析和地统计学不在本课程要求掌握的范围内。

例 1：叠加分析

1. 添加需要用到的 library，包括 `readr`、`ggplot2`、`sf` 和 `dplyr` 等。导入北京区县的 `Shapefile` 数据（和 Lab 3 案例中的数据相同），并导入 `beijing_tourism.csv` 数据集，该数据集为北京市重要景区数据集，包含了北京市境内 4A 和 5A 级风景区的名单（Name）、级别（Level）及经纬度信息（Lat/Lon）。我们首先将景区的 `csv` 数据集转换成 `Shapefile` 格式，并绘制一幅简单的景区分布图。

```
# 导入北京行政区划 shapefile
beijing <- sf::st_read("data/beijing_districts/beijing_districts.shp")
# 导入北京著名景区数据，并转为 shapefile 格式
tourism <- read_csv("data/beijing_tourism.csv")
tourism_sf <- st_as_sf(tourism, coords = c("Lon", "Lat"), crs=4326)
# 创建地图
p <- ggplot() +
  geom_sf(data = beijing, color="black") +
  geom_sf(data = tourism_sf, aes(color=Level), alpha=0.7) +
  theme_void() +
  theme(legend.position = "bottom") +
  labs(color = "Attraction Level")
# 绘制地图
p
```

输出效果：



2. 在原本的北京市重要景区数据集中，并未包含每个景区所在的区县。要想得知每个景区所对应的区县，我们需要进行“空间连接”（`spatial join`）操作。空间连接会根据空间关系，将地理要素 A 连接到目标要素 B，并将要素 A 的属性写进要素 B 中。在这个例子里，北京区县数据为要素 A，它提供我们所需要的属性数据，即区县名称，而景区数据则为目标要素 B，是属性的需求侧。

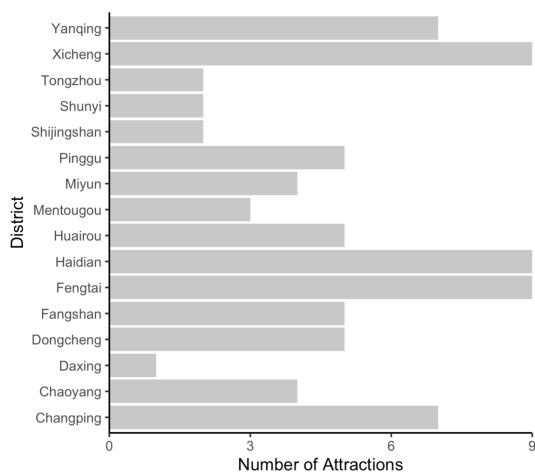
在 R 语言中，`spatial join` 的操作如下：

```
# 使用st_join功能，在北京景区数据和区县数据之间完成spatial join
# 获得一个新的Shapefile型数据框
beijing_tourism <- st_join(
  x=tourism_sf,
  y=beijing,
  join = st_intersects)
View(beijing_tourism)
```

查看后可知，原本位于北京区县数据中的区县 `id` 和区县名称信息，已经被作为新的属性添加到了景区的属性之后。我们可以画一张条形图来展示各区县的景区数量：

```
# 统计各区的景区分布
distribution <- data.frame(table(beijing_tourism$district))
# 创建一幅条形图
p <- ggplot() +
  geom_bar(data=distribution, aes(x=Var1, y=Freq),
           stat="identity", fill="gray80")+
  theme_classic()+
  coord_flip()+
  scale_y_continuous(expand=c(0,0),breaks=c(0,3,6,9))+
  labs(y="Number of Attractions", x="District")
# 绘图
p
```

输出效果：

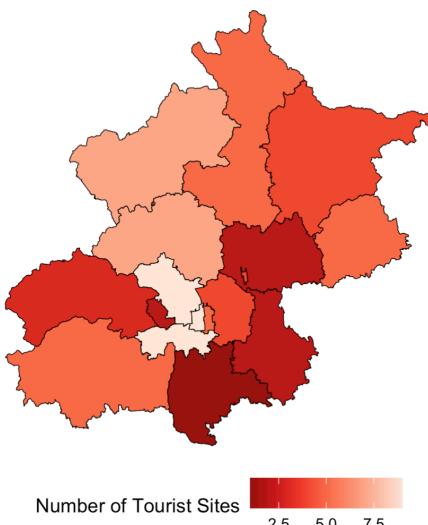


3. 现在我们有一个新的需求：对每个区县拥有的景区数量在地图上进行可视化。我们有两个方法可以达到这一目的，第一个方法是利用 `merge` 函数将上一步生成的 `distribution` 数据框与原本的区县地图数据进行合并（注意：首先要改变 `distribution` 数据框两列的名称），然后再用合并后的地图数据进行绘图。这种方法的所有技术都已经介绍过，这里略过。

第二个方法是回到地图数据上，采用“空间交叉”（`spatial intersect`）操作，让每一个区县对应的多边形找到其包含的景区数据点，再进行计数统计。这样可以直接把计数的结果作为一个新的属性赋予到地图数据的后面。R 代码如下：

```
#为北京区县数据新建一个属性
#利用sf中的st_intersects函数完成区县数据和景区数据之间的空间交叉
#用lengths函数统计每个区县匹配成功的景区数量
beijing$tour_sites <- lengths(st_intersects(beijing, beijing_tourism))
#创建各区县景区数量地图
p <- ggplot() +
  geom_sf(data = beijing, aes(fill=tour_sites), color="black") +
  theme_void() +
  theme(legend.position = "bottom") +
  labs(fill = "Number of Tourist Sites") +
  scale_fill_distiller(palette = "Reds")
#绘制地图
p
```

输出效果：



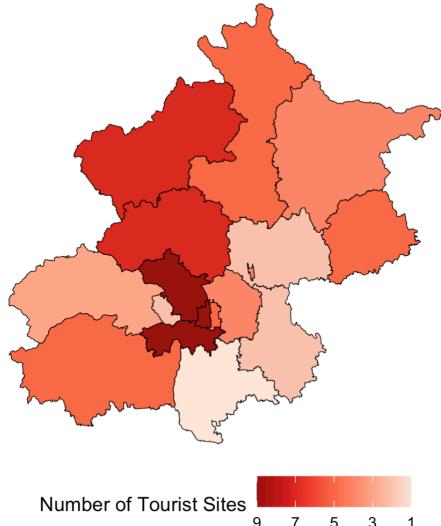
4. 这幅地图能够呈现出每个区县拥有的景区数量。但是在观察这幅地图后，我们发现了两个瑕疵。1) 图例中，色带上的标注有 2.5、5.0、7.5 三个数字，出现了小数点，这显然不符合“景区”的数据特征，因为一般不会存在“半个景区”的情况。2) 色带中，较深的红色对应较小的数据值，较浅的红色对应较大的数据值，这不符合类似的“计数”型专题地图的常见着色逻辑，我们希望用较深的颜色对应景区较多的区县。

更改的代码如下：

```
#反转色带的深浅，并调整色带的标签，使其为整数
p <- ggplot() +
  geom_sf(data = beijing, aes(fill=tour_sites), color="black") +
  theme_void() +
  theme(legend.position = "bottom") +
  labs(fill = "Number of Tourist Sites") +
  scale_fill_distiller(palette = "Reds", trans = 'reverse',
                        labels = c("1", "3", "5", "7", "9"),
                        breaks = c(1, 3, 5, 7, 9))

#绘制地图
p
```

输出效果：

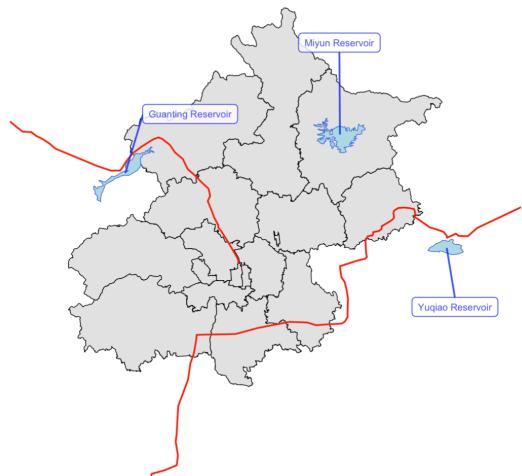


5. 导入两个新的地图数据集：routes.shp（国道 G110 和 G230 的部分路段）和 reservoirs.shp（北京附近的大型水库），绘制基本的地图。

```
#导入国道和水库数据
routes <- sf::st_read("routes/routes.shp")
reservoirs <- sf::st_read("reservoirs/reservoirs.shp")
#创建基本地图
p <- ggplot() +
  geom_sf(data = beijing, color="black") +
  geom_sf(data = reservoirs, fill="lightblue", color="royalblue") +
  geom_sf(data = routes, color="red") +
  geom_label_repel(
    data = reservoirs,
    aes(label = name, geometry = geometry),
    size = 2, box.padding = 2, stat = "sf_coordinates",
    min.segment.length = 0.5, colour = "blue",
    segment.colour = "blue", alpha = 0.8
  ) +
  theme_void()

#绘制地图
p
```

输出效果：

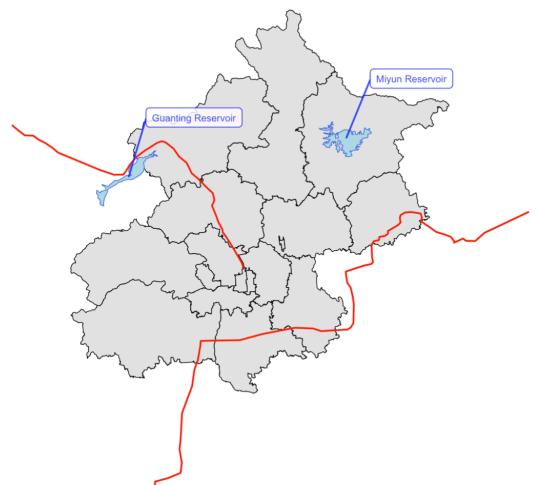


地图上的两条红线为国道。北京到宣化的 G110 只有部分路段在北京市境内，而从雄安到遵化的 G230 先后有两处路段在北京境内。此外，三个水库中，只有密云水库完全位于北京境内，官厅水库有一部分位于北京境内，而于桥水库完全位于北京以外。

6. 下一步，我们希望删除完全位于北京之外的于桥水库，可以在 R 中调用 `st_intersects`，计算每个水库和北京的几个区县有重合，而完全位于北京之外的于桥水库在这项计数上为 0，可以据此筛选。

```
#计算每个水库和北京的几个区县有重合
reservoirs$touch_beijing <- lengths(st_intersects(reservoirs, beijing))
#保留和至少1个北京区县有重合的水库，去掉其余的
reservoirs_touch_beijing <- reservoirs[which(reservoirs$touch_beijing>=1),]
#创建基本地图
p <- ggplot() +
  geom_sf(data = beijing, color="black") +
  geom_sf(data = reservoirs_touch_beijing, fill="lightblue", color="royalblue") +
  geom_sf(data = routes, color="red") +
  geom_label_repel(
    data = reservoirs_touch_beijing,
    aes(label = name, geometry = geometry),
    size = 2, box.padding = 2, stat = "sf_coordinates",
    min.segment.length = 0.5, colour = "blue",
    segment.colour = "blue", alpha = 0.8
  )+
  theme_void()
#绘制地图
p
```

输出效果：



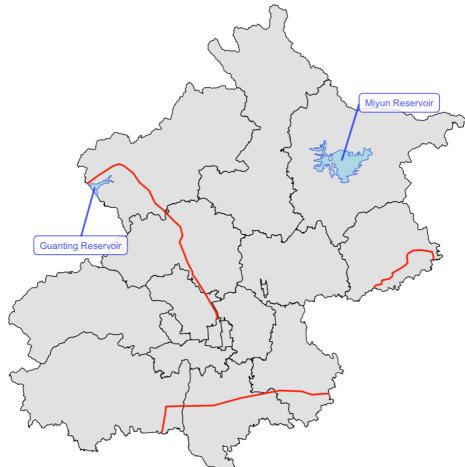
7. 接下来，我们希望只保留位于北京境内的国道路段，官厅水库也只保留位于北京境内的部分。我们可以调用 `st_intersection` 函数（注意和 `st_intersects` 函数相区别）来实现这一需求。

```
# 获取国道和水库完全位于北京境内的部分
reservoirs_within_beijing <- st_intersection(reservoirs, beijing)
routes_within_beijing <- st_intersection(routes, beijing)

# 创建基本地图
p <- ggplot() +
  geom_sf(data = beijing, color = "black") +
  geom_sf(data = reservoirs_within_beijing, fill = "lightblue", color = "royalblue") +
  geom_sf(data = routes_within_beijing, color = "red") +
  geom_label_repel(
    data = reservoirs_within_beijing,
    aes(label = name, geometry = geometry),
    size = 2, box.padding = 2, stat = "sf_coordinates",
    min.segment.length = 0.5, colour = "blue",
    segment.colour = "blue", alpha = 0.8
  ) +
  theme_void()

# 绘制地图
p
```

输出效果：



这种操作在 ArcGIS 等软件里被称为“要素剪切”(clip)。

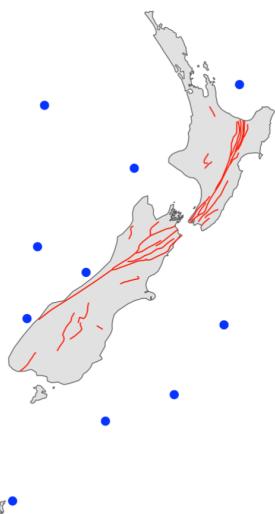
例 2：几何计算和缓冲区分析

有时候，我们需要对地理要素的几何属性进行计算，比如线性要素的长度、多边形的面积、要素之间的距离等。需要注意的是，在 R 中计算地理要素的几何属性时，默认情况下采用的是 WGS 84 大地参考系。例 2 要用到三个数据集：`newzealand.shp` 为新西兰主要岛屿数据（多边形），`nz_faults.shp` 为新西兰主要活跃断层数据（线性数据）以及 `ocean_site.csv` 为新西兰附近的海洋观测站的经纬度坐标（点数据）。

```
# 导入新西兰主要岛屿和断层的Shapefile数据
nz <- sf::st_read("newzealand/newzealand.shp")
faults <- sf::st_read("nz_faults/nz_faults.shp")
# 导入海洋观测站的位置数据，并将其转换为shapefile
sites <- read_csv("ocean_site.csv")
sites_sf <- st_as_sf(sites, coords = c("lon", "lat"), crs=4326)
# 创建地图
p <- ggplot() +
  geom_sf(data = nz) +
  geom_sf(data = faults, color = "red", linewidth = 0.3) +
  geom_sf(data = sites_sf, color = "blue") +
  theme_void()

# 绘制地图
p
```

输出效果：



2. 计算各个岛屿的面积，使用 `st_area()` 函数。

```
# 打印每一个岛屿的面积，并告知单位（为平方米）
print(st_area(nz))
# 将面积作为新的属性添加到新西兰的地图数据中，并改为平方公里
nz$area_km2 <- as.numeric(test/1000000)
```

打印结果：

```
Units: [m^2]
[1] 115356481164 153169815963 1700725798 13774269 16098950 85962823 4643536
[8] 4262299 21592498 29607910 17795280 297913408 19712141 480837942
[15] 7365273 93872145 103138234
```

添加属性后：

FID	name	geometry	area_km2
0	North Island	POLYGON ((174.5198 -38.8996...))	1.153565e+05
1	South Island	POLYGON ((168.853 -46.63342...))	1.531698e+05
2	Stewart Island	POLYGON ((167.7511 -46.7094...))	1.700726e+03
3	Codfish Island	POLYGON ((167.6002 -46.7561...))	1.377427e+01
4	Rapanui Island	POLYGON ((168.5284 -46.7414...))	1.609895e+01

同理，可以使用 `st_length()` 函数，计算每一条断层的长度。

```
# 打印每一条断层的长度，并告知单位（为平方米）
print(st_length(faults))
# 将长度作为新的属性添加到断层的地图数据中，并改为公里
faults$length_km <- as.numeric(test/1000)
```

还可以用 `st_distance()` 函数计算地理要素之间的最近距离（球面距离）。

```
# 各个海洋观测站到各个岛屿之间距离的矩阵
st_distance(sites_sf, nz)
# 海洋观测站“A”到南岛之间的距离
st_distance(sites_sf[which(sites_sf$name == "A"),], nz[which(nz$name == "South Island"),])
```

注：本次练习使用的数据为低分辨率的示例数据，计算出的面积和长度精确度不高，仅用于展示方法。

3. 接下来，我们进行缓冲区分析。缓冲区分析是地理信息学的一种常见空间分析技术。它主要用于找出某个地理要素附近一定距离以内的区域。

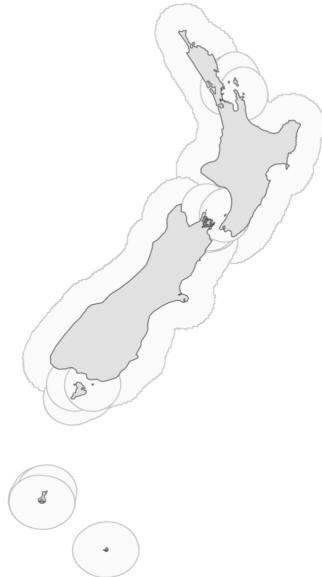
在具体操作中，缓冲区分析会向外（或向内）延伸指定地理要素（点、线、多边形）的外围，按照给定的距离，创建新的图形要素，这个新图形就是“缓冲区”（buffer zone）。例如，我们可以为某条河流创建一个 100 米宽的缓冲区，该缓冲区表示出河流两侧各 100 米范围内的地带。

通过这种方式，我们可以识别出距离地理要素某一特定距离范围内存，在哪些其他地理要素或事件。比如，在上述例子中，我们可以分析出距离河岸 100 米的缓冲区内有多少建筑。这种空间分析方法常用于选址，比如要为一个化工厂选址，根据要求，它到任何天然水域的距离必须在 5 千米以上，距离任何居民区的距离必须在 10 千米以上，而距离原料产地的距离必须在 20 千米以内。于是，我们可以分别在天然水域、居民区和原料产地的要素周围按相应的距离建立缓冲区，然后找出谓语原料产地的缓冲区内，但谓语天然水域和居民区缓冲区以外的区域，作为潜在的化工厂选址地区。

现在，我们要找出哪些海洋观测站属于近岸观测站（距离新西兰岛屿海岸线的距离小于 100 千米）。首先，我们使用 `st_buffer` 函数给岛屿建立一个 100 千米的缓冲区。注意，该功能的默认单位是米。

```
#在岛屿外围添加100千米的缓冲区（函数默认单位是米）
nz_buffer100km <- st_buffer(nz, 100000)
#创建地图查看
p <- ggplot() +
  geom_sf(data = nz_buffer100km, fill = "snow", color="gray") +
  geom_sf(data = nz) +
  theme_void()
#绘制地图
p
```

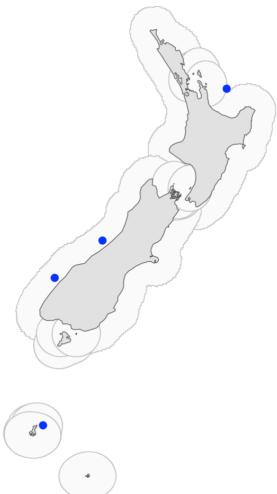
输出效果：



然后，利用 `st_filter` 函数直接过滤海洋观测站数据。过滤后，新数据集里保留下的观测站为近岸观测站，距离岛屿不足 100 千米；而没被留下的为远洋观测站，距离岛屿 100 千米以上。

```
#找到距离新西兰陆地100千米以内的观测站
sites_nearshore <- st_filter(sites_sf, nz_buffer100km)
#创建地图
p <- ggplot() +
  geom_sf(data = nz_buffer100km, fill = "snow", color="gray") +
  geom_sf(data = nz) +
  geom_sf(data=sites_nearshore,color="blue")+
  theme_void()
#绘制地图
p
```

输出效果：



在可视化时，我们还可以用 `st_union` 函数把缓冲区合并为单个多边形，减少地图的复杂度，突出视觉效果。

```
#将缓冲区合并，让地图更清晰
nz_buffer100km <- st_union(nz_buffer100km)
p <- ggplot() +
  geom_sf(data = nz_buffer100km, fill = "snow", color="gray") +
  geom_sf(data = nz) +
  geom_sf(data=sites_sf, aes(color="> 100km from the coast"))+
  geom_sf(data=sites_nz,aes(color="Within 100km from the coast"))+
  ggtitle("New Zealand") +
  labs(color="Oceanic research sites")+
  theme_bw()+
  scale_x_continuous(breaks=c(164,168,172,176))
#绘制地图
p
```

输出效果：

```
#将缓冲区合并，让地图更清晰
nz_buffer100km <- st_union(nz_buffer100km)
#创建地图
p <- ggplot() +
  geom_sf(data = nz_buffer100km, fill = "snow", color="gray") +
  geom_sf(data = nz) +
  geom_sf(data=sites_nearshore,color="blue")+
  theme_void()
#绘制地图
p
```

输出效果：

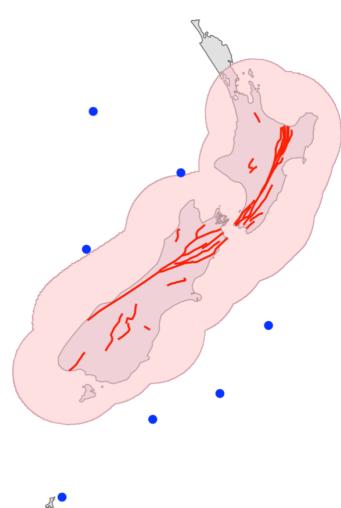


同样的道理，我们可以分析哪些观测站距离断层的距离大于 200 千米。

```
#在断层两侧添加200千米的缓冲区
faults_buffer200km <- st_buffer(faults, 200000)
##因为我们要找200千米以外的点位，因此不能直接用st_filter
#用st_join功能，将观测站和缓冲区的交叉状况作为新的属性添加给观测站数据
#再根据这个新属性来subset海洋观测站数据
#新属性为空的观测站位于缓冲区之外
sites_class <- st_join(sites_sf, faults_buffer200km)
sites_chosen <- sites_class[which(is.na(sites_class$id.y)),]
#合并缓冲区，让地图更清晰
faults_buffer200km <- st_union(faults_buffer200km)
#创建地图
p <- ggplot() +
  geom_sf(data = nz) +
  geom_sf(data = faults_buffer200km, fill = "pink", color = "pink", alpha = 0.5) +
  geom_sf(data = faults, color = "red") +
  geom_sf(data=sites_chosen,color="blue")+
  theme_void()
#绘制地图
p
```

注意：这里用来筛选的属性是 `id.y` (`join` 后自动添加的 `y`)。其他来自 `buffer` 图层的属性也可以用来筛选。

输出效果：



例 3：表面分析

表面分析是空间分析的一种类型，主要用于研究和理解连续的数据或现象（如温度、湿度、气压、地形等）在空间中的变化。这种分析通常基于栅格数据（raster data），因为栅格数据能很好地表示连续的表面。

栅格数据是一种常见的地理空间数据模型，它将地球表面划分为规则的、矩形形状的网格单元（cell，即栅格，或称像素），并在每个单元内存储一个值。这些值可以代表各种类型的信息，如海拔高度、土壤类型、温度、降水量、遥感波段反射率等。

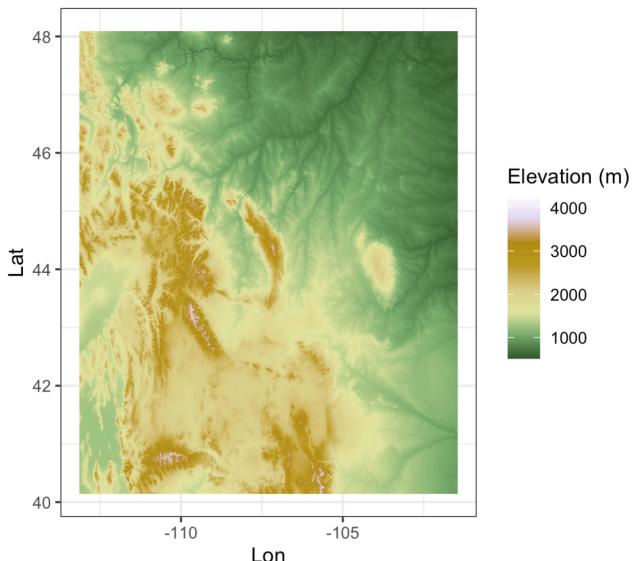
具体来说，栅格数据由行和列构成，并且每个网格单元都有唯一的行列索引。每个网格单元包含一个数值，表示某种特定属性或测量结果。例如，在数字高程模型（Digital Elevation Model, DEM）中，每个网格单元包含了其所在位置的海拔高度信息。

相比于另一类主要地理空间数据—矢量数据（点、线、多边形），栅格数据更适合表示在空间里连续变化的现象或模式（比如温度、植被覆盖、地形等），其精确性取决于所用网格单元大小：栅格越小，分辨率越高，能够提供更详细准确的信息，但数据体量也就越大，处理时间越长。

1. 安装并导入 raster 包，读取栅格数据 `terrain.tif`。这是一个 GeoTiff 格式的栅格数据，即带有地理空间坐标 Tiff 图像。本数据为美国黄石国家公园附近的落基山脉部分区域（包括提顿山脉、尤因他山脉、风河山脉、大角山脉和黑山，以及几个山间盆地，包括绿河盆地、大角盆地、炮灰河盆地、雷霆盆地等）的海拔数据。

```
library(raster)
# 读取栅格数据
terrain <- raster("terrain.tif")
# 将GeoTiff栅格数据转化为数据框格式
terrain_df <- as.data.frame(terrain, xy = TRUE)
# 创建地图，定义海拔高度的经典色带
terrain_color_ramp = c("#33592c", "#78ad6d", "#e3e6a1", "#dbcc86",
                        "#bd9c26", "#b38b15", "#e2d5f2", "white")
p <- ggplot() +
  geom_raster(data = terrain_df, aes(x = x, y = y, fill = terrain)) +
  scale_fill_gradientn(colours=terrain_color_ramp) +
  labs(fill="Elevation (m)", x = "Lon", y = "Lat") +
  theme_bw()
# 绘制地图
p
```

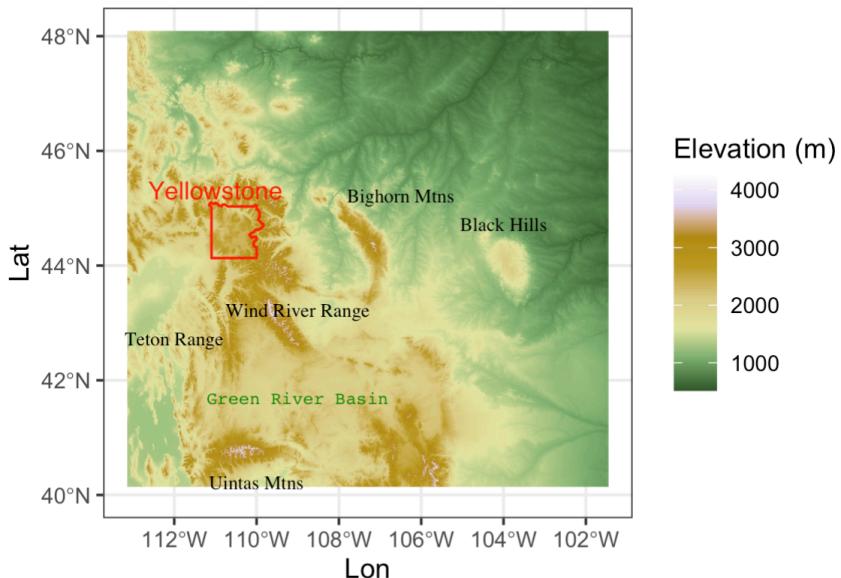
输出效果：



2. 和矢量地图一样，栅格地图也是一个图层，可以和其他图层相互叠加。比如，我们可以导入黄石国家公园的轮廓数据，并且批量添加一些标注。

```
# 导入黄石公园的矢量数据
yellowstone <- sf::st_read("yellowstone/yellowstone.shp")
# 创建地图，并批量添加标注
p <- ggplot() +
  geom_raster(data = terrain_df, aes(x = x, y = y, fill = terrain)) +
  geom_sf(data = yellowstone, fill = NA, color = "red", linewidth = 0.5) +
  scale_fill_gradientn(colours=terrain_color_ramp) +
  labs(fill="Elevation (m)", x = "Lon", y = "Lat") +
  annotate("text",
    y = c(45, 44.5, 45, 43, 42.5, 40, 41.5),
    x = c(-111, -104, -106.5, -109, -112, -110, -109),
    label = c("Yellowstone", "Black Hills", "Bighorn Mtns", "Wind River Range",
              "Teton Range", "Uintas Mtns", "Green River Basin"),
    color = c("red", "black", "black", "black", "black", "black", "green4"),
    family=c("Helvetica", "Times", "Times", "Times", "Times", "Times", "Courier"),
    size = c(3.5, 2.7, 2.7, 2.7, 2.7, 2.7, 2.5),
    hjust = 0.5, vjust = -0.5) +
  theme_bw()
# 绘制地图
p
```

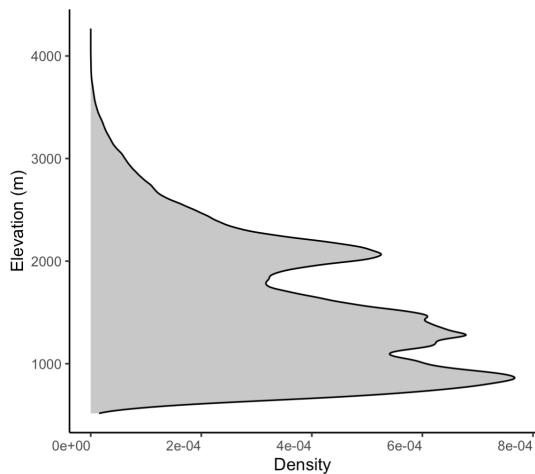
输出效果：



3. 我们可以对转换成数据框格式的地形数据做一些基本的统计。比如查看海拔分布：

```
# 通过密度图查看该区域内的海拔分布(即各海拔对应的网格单元的数量)
# 因为我们统计的是海拔，所以可以把坐标轴翻转，更符合阅读习惯
p <- ggplot() +
  geom_density(data=terrain_df, aes(x=terrain), fill="gray80") +
  theme_classic() +
  labs(x="Elevation (m)", y="Density") +
  coord_flip() +
  theme(axis.text.x = element_text(hjust=1))
p
```

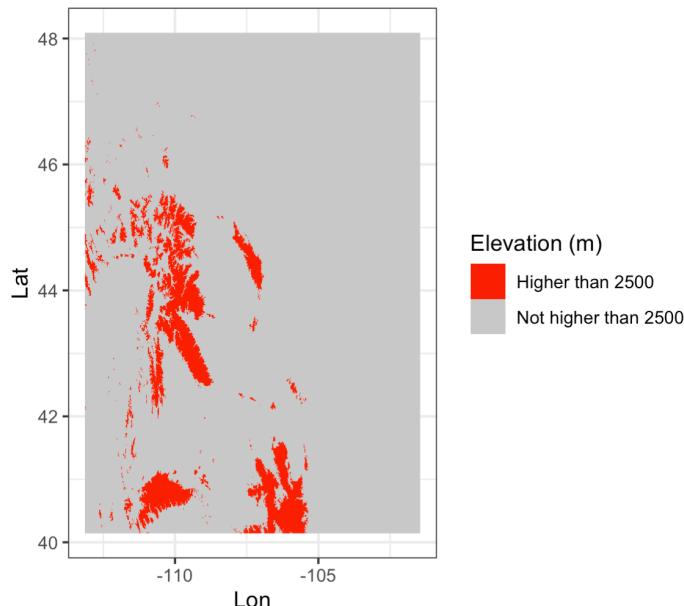
输出效果：



或者可以把地形数据集按海拔高度分类，寻找符合特定数据的区域在空间中的分布模式（是离散化的一种，在 ArcGIS 等软件内叫做 `reclassify`）。

```
#重新分类 (reclassify)，找出海拔高于2500米的高山区域
library(dplyr)
terrain_df <- terrain_df %>
  mutate(reclassify = ifelse(terrain > 2500, "Higher than 2500", "Not higher than 2500"))
#生成地图
p <- ggplot() +
  geom_raster(data = terrain_df, aes(x = x, y = y, fill = reclassify))+
  scale_fill_manual(values=c("Higher than 2500"="red", "Not higher than 2500"="gray80"))+
  labs(fill="Elevation (m)", x = "Lon", y = "Lat")+
  theme_bw()
#绘制地图
p
```

输出效果：



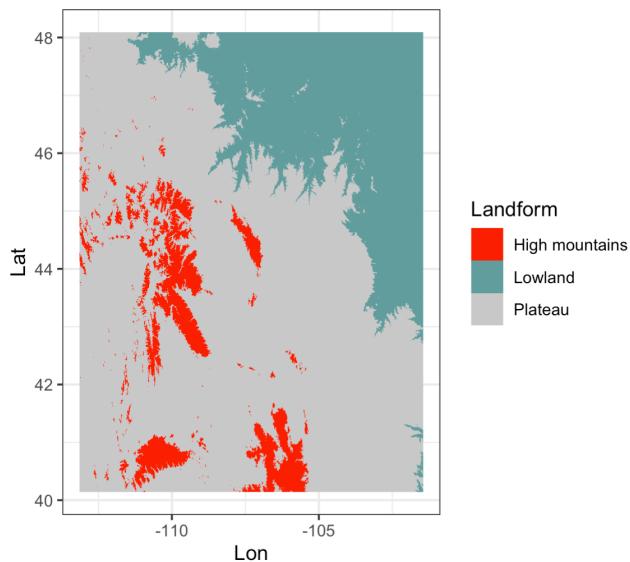
分更多的类别，可以在 `mutate` 函数中进行持续操作：

```
#重新分为3类：海拔高于2500米的高山区域，低于1000米的低洼区域，以及中间的普通高地
terrain_df <- terrain_df %>%
  mutate(reclassify = ifelse(terrain > 2500, "High mountains",
                            ifelse(terrain > 1000, "Plateau", "Lowland")))

#生成地图
p <- ggplot() +
  geom_raster(data = terrain_df, aes(x = x, y = y, fill = reclassify)) +
  scale_fill_manual(values=c("High mountains"="red",
                            "Plateau"="gray80",
                            "Lowland"="cadetblue")) +
  labs(fill="Landform", x = "Lon", y = "Lat") +
  theme_bw()

#绘制地图
p
```

输出效果：

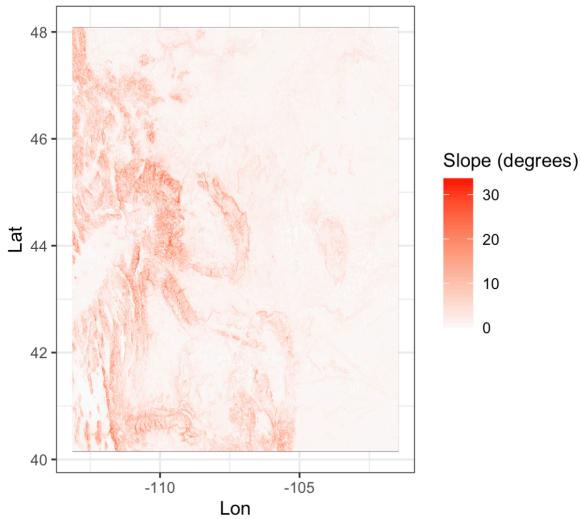


4. 除了直接对栅格数据进行分类以外，我们还可以寻找栅格数据集中数据变化最快的地方，对于本例题中的地形数据来说，即寻找坡度较大的区域。此时我们需要用到 `terrain` 函数（注意：例子里的地形数据集也被命名为 `terrain`，请仔细阅读代码，勿混淆）。

```
#使用terrain函数计算坡度，获得一个新的栅格数据
slope <- terrain(terrain, opt = "slope", unit="degrees")
#将新得到的栅格数据转化为数据框格式
slope_df <- as.data.frame(slope, xy = TRUE)
#创建地图
p <- ggplot() +
  geom_raster(data = slope_df, aes(x = x, y = y, fill = slope)) +
  scale_fill_gradientn(colours=c("snow","red")) +
  labs(fill="Slope (degrees)", x = "Lon", y = "Lat") +
  theme_bw()

#绘制地图
p
```

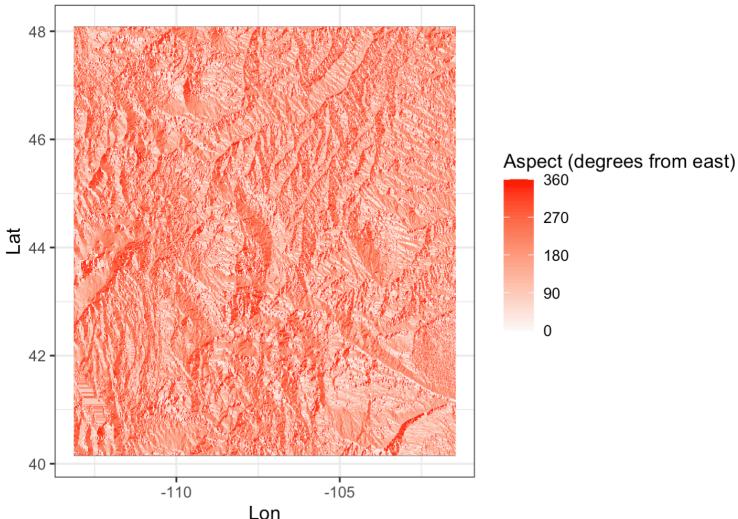
输出效果：



除了坡度大小以外，我们还可以计算坡度的朝向（和正东方向的夹角）：

```
# 使用 terrain 函数计算朝向，获得一个新的栅格数据
aspect <- terrain(terrain, opt = "aspect", unit="degrees")
# 将新得到的栅格数据转化为数据框格式
aspect_df <- as.data.frame(aspect, xy = TRUE)
# 创建地图
p <- ggplot() +
  geom_raster(data = aspect_df, aes(x = x, y = y, fill = aspect))+
  scale_fill_gradientn(colours=c("snow","red"),
                        breaks = c(0,90,180,270,360))+
  labs(fill="Aspect (degrees from east)", x = "Lon", y = "Lat")+
  theme_bw()
# 绘制地图
p
```

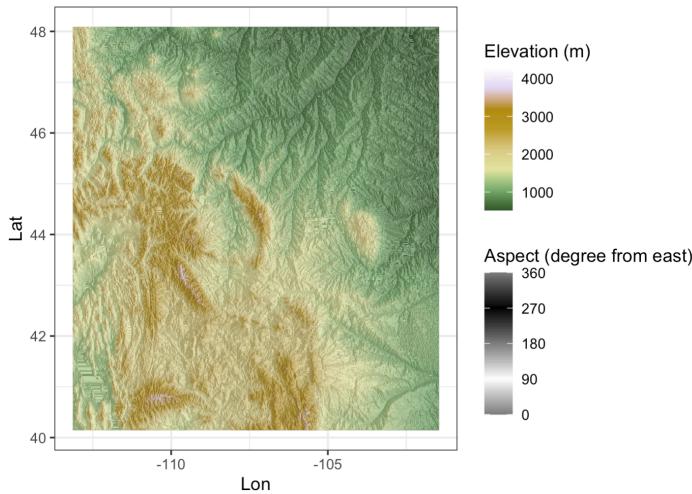
输出效果：



获得的坡度朝向数据除了可以用于 `reclassify` 以及其他相关的数据分析以外，还有一个重要作用是模拟太阳光的照射，增强对地形数据的表现力。北半球的正午时分，阳光来自正南，我们可以调整色带，让 90° 的亮度最大、 270° 的亮度最小，来体现阳光下的地貌阴影。然后我们把朝向数据叠加到海拔数据之上，并设置透明度（通常 $0.3-0.5$ ，可以自己调试观察效果）让它们在视觉效果上形成融合（这种操作叫做 `hillshade`）。因为我们需要在同一张图像里用到不同的色带，因此需要引入 `ggnewscale` 安装包，它能让我们在绘制不同图层的时候重复定义色带。

```
#利用aspect创建地图，突出立体感
library(ggnewscale)
p <- ggplot() +
  geom_raster(data = terrain_df, aes(x = x, y = y, fill = terrain))+ 
  scale_fill_gradientn(colours=terrain_color_ramp)+ 
  labs(fill="Elevation (m)")+ 
  new_scale_fill() + 
  geom_raster(data = aspect_df, aes(x = x, y = y, fill = aspect),alpha=0.3)+ 
  scale_fill_gradientn(colours=c("gray50","white","gray50","black","gray50"), 
                        breaks = c(0,90,180,270,360))+ 
  labs(fill="Aspect (degree from east)", x = "Lon", y = "Lat")+
  theme_bw()
#绘制地图
p
```

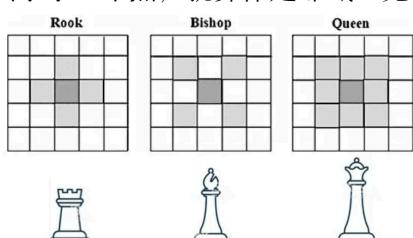
输出效果（可以和例 3 第一幅图对比）：



此外，`terrain` 函数还可以分析流向 (`flow direction`) 和表面起伏度 (`ruggedness`)。流向是栅格数据中数值下降最快的方向，而表面粗糙度又分为 `TRI` 和 `TPI` 两种算法，`TRI` 对比的是某个栅格单元的数值和其所有相邻单元 (`neighbors`, 邻域) 平均值的差距，而 `TPI` 对比某个栅格单元的数值和邻域之间的最大差值。另有表面粗糙度 (`roughness`) 的算法，它计算某个栅格单元及其邻域的所有值之间的极差。表面起伏度和表面粗糙度越大，表示数据在局部区域内越不稳定。

```
#计算流向
flowdir <- terrain(terrain, opt = "flowdir", neighbors=8)
#计算表面起伏度 (TRI算法)
tri <- terrain(terrain, opt = "tri", neighbors=8)
#计算表面起伏度 (TPI算法)
tpi <- terrain(terrain, opt = "tpi", neighbors=8)
#计算表面粗糙度
roughness <- terrain(terrain, opt = "roughness", neighbors=8)
```

这几个函数中，`neighbors` 参数定义相邻单元的选取范围，有 4 和 8 两种输入。当 `neighbors` 为 4 时，邻域定义采取 `rook` 规则，只选择有共享边的单元作为邻域；当 `neighbors` 为 8 时，邻域定义采取 `queen` 规则，只要共享了一个点，就算作是邻域。见下图：

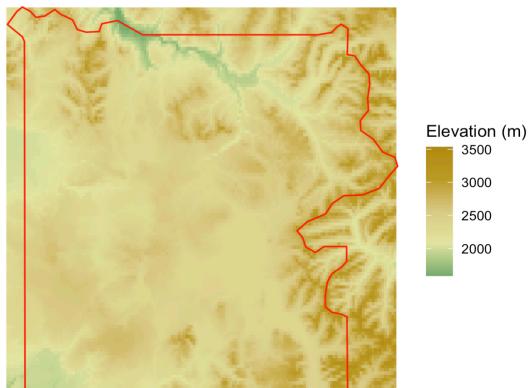


此图为空间分析中的三种定义邻域的方法：车（rook）、相（bishop）和皇后（queen），这三个名字都来源于国际象棋。国际象棋里，车的走法是横竖直线，类比到空间邻域中，如果两个多边形共享了任何边线，按照 rook 的定义它们就是相邻的。相走对角线，因此按照 bishop 的定义法，两个多边形如果只共享了一个端点，则是相邻的，这种定义使用场景十分稀少，一般不作考虑。皇后的走法是车和相的结合，因此按照 queen 的定义法，两个多边形共享了一个端点或一条边，则为相邻的。在规整的栅格数据里，rook 有 4 个邻域，queen 有 8 个邻域。

5. 很多时候，原始的栅格数据覆盖范围远大于我们的研究区域。我们可以用 `crop` 或 `mask` 两种剪切方法清除不需要的额外数据，以节省计算资源。比如，我们将例题里的栅格数据缩减到黄石公园的范围。用 `crop` 函数：

```
#将栅格数据剪切到和黄石国家公园相同的经纬度范围
terrain_ys_crop <- crop(terrain, extent(yellowstone))
#转换格式为data frame
terrain_ys_crop_df <- as.data.frame(terrain_ys_crop, xy = TRUE)
p <- ggplot() +
  geom_raster(data = terrain_ys_crop_df, aes(x = x, y = y, fill = terrain))+
  geom_sf(data = yellowstone, fill = NA, color = "red", linewidth = 0.5) +
  scale_fill_gradientn(colours=c("#78ad6d", "#e3e6a1", "#dbcc86", "#bd9c26", "#b38b15"))+
  labs(fill="Elevation (m)", x = "Lon", y = "Lat")+
  theme_void()
#绘制地图
p
```

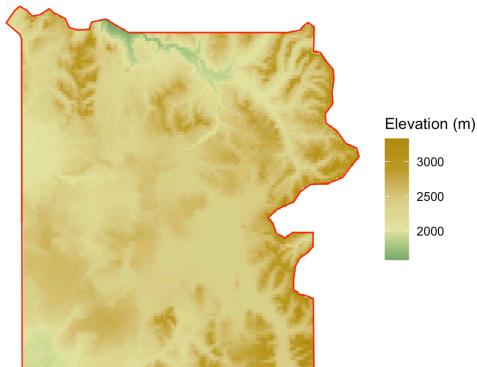
输出效果：



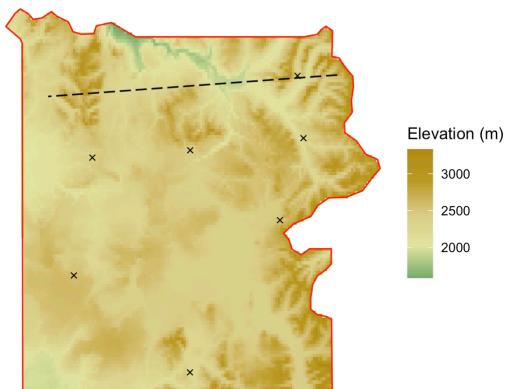
或者用 `mask` 函数，让缩减后的栅格数据完全符合黄石公园的轮廓（易错点：`terrain_ys_mask_df$terrain` 中的 `terrain` 不是数据框的名称，而是前一步生成的变量名，请注意观察）：

```
#将栅格数据剪切到和黄石国家公园相同的轮廓（轮廓外的数据值设为空值）
terrain_ys_mask <- mask(terrain, as(yellowstone, "Spatial"))
#转换格式为data frame，并清除空值
terrain_ys_mask_df <- as.data.frame(terrain_ys_mask, xy = TRUE)
terrain_ys_mask_df <- terrain_ys_mask_df[which(!is.na(terrain_ys_mask_df$terrain)),]
#创建地图
p <- ggplot() +
  geom_raster(data = terrain_ys_mask_df, aes(x = x, y = y, fill = terrain))+
  geom_sf(data = yellowstone, fill = NA, color = "red", linewidth = 0.5) +
  scale_fill_gradientn(colours=c("#78ad6d", "#e3e6a1", "#dbcc86", "#bd9c26", "#b38b15"))+
  labs(fill="Elevation (m)", x = "Lon", y = "Lat")+
  theme_void()
#绘制地图
p
```

输出效果：



6. 数据集 `yellowstone_sites.csv` 为黄石公园内的一些野外考察站坐标。矢量数据 `line.shp` 为黄石公园内的一条考察路线。导入这两组数据，并绘制地图查看（代码略）。



在科研或商业分析中，我们通常想知道某些具体位置对应的栅格数据中的值是多少。例如，我们希望得知考察站各自的海拔。此时我们可以用 `extract` 函数达到目的：

```
# 导入考察站数据，并转换为Shapefile格式
sites <- read_csv("yellowstone_sites.csv")
sites_sf <- st_as_sf(sites, coords = c("lon", "lat"), crs=4326)
# 提取考察站所在地的海拔，作为新的属性添加到考察站数据里
sites_sf$elev_m <- extract(terrain_ys_mask, as(sites_sf, "Spatial"))
print(sites_sf)
```

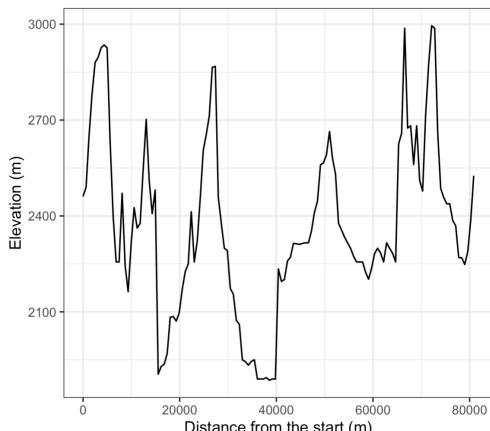
输出结果：

	<code>id</code>	<code>name</code>	<code>geometry</code>	<code>elev_m</code>
*	<code><dbl></code>	<code><chr></code>	<code><POINT [°]></code>	<code><dbl></code>
1	1	A	(-110.1839 44.56977)	2477
2	2	B	(-110.1007 44.77616)	2301
3	3	C	(-110.5019 44.7457)	2438
4	4	D	(-110.1217 44.93297)	2256
5	5	E	(-110.5027 44.18689)	2267
6	6	F	(-110.914 44.43064)	2537
7	7	G	(-110.8483 44.72718)	2438

此外，我们希望得知考察路线沿途的海拔变化情况，此时也可以利用 `extract` 对路线沿途进行采样，并绘制距离 `vs` 海拔的折线图。

```
#读取路线沿途的海拔数据(按照线数据的方向性排列, 可参见LINESTRING属性中的起止点坐标)
line <- sf::st_read("line/line.shp")
elev_line <- extract(terrain_ys_mask, line, method = "bilinear")
#创建一个数据框, 储存路线沿途各采样点的海拔, 以及各点到起点的距离
#采样点即为线要素经过的栅格数据单元
profile <- data.frame(
  distance = seq(from = 0, #出发点
                 to=as.numeric(st_length(line)), #终点到出发点的距离
                 by = as.numeric(st_length(line))/(length(elev_line[[1]])-1)), #分配给采样点-1个区间
  elevation = as.numeric(elev_line[[1]]))
#画图
p <- ggplot()+
  geom_line(data=profile,aes(x=distance,y=elevation))+
  theme_bw()+
  labs(x="Distance from the start (m)", y="Elevation (m)")
p
```

输出结果:

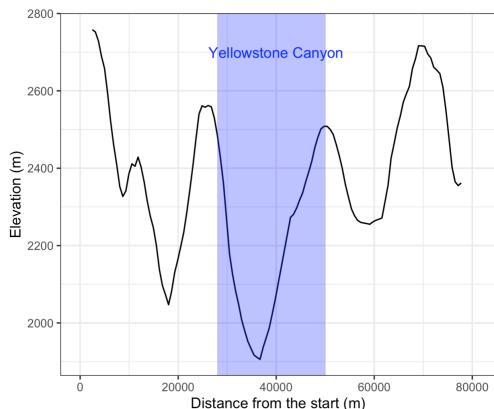


有时候我们不需要如此复杂的地形剖面, 而只需要看大致的趋势, 此时我们可以用安装包 zoo 里的 rollmean 函数 (即统计学中的移动平均法) 来让曲线平滑。下面代码中的 10 为移动窗口的宽度, 这个数越大, 结果越简略。

```
#用rollmean来化简地形图, 并标注黄石大峡谷的范围。
require(zoo)
p <- ggplot()+
  geom_line(data=profile,aes(x=distance,y=rollmean(elevation, 10,
                                                       na.pad = TRUE,
                                                       align = "center")))+

  theme_bw()+
  geom_rect(aes(xmin=28000,xmax=50000,ymin=-Inf,ymax=Inf),fill="blue",alpha=0.3)+
  geom_text(aes(x=40000,y=2700,label="Yellowstone Canyon"),color="blue")+
  labs(x="Distance from the start (m)", y="Elevation (m)")
p
```

输出结果:



作业部分

数据描述

sichuan.zip

包含 `sichuan.shp` 及相关文档，为四川省各地级行政区的多边形地图数据。

chongqing.zip

包含 `chongqing.shp` 及相关文档，为重庆市辖区的多边形轮廓。

sichuan_basin.tif

GeoTiff 格式的栅格数据，为四川盆地及周边地区的海拔高度（米）。

temp.tif

GeoTiff 格式的栅格数据，为四川盆地及周边地区 2009 年某日的最高气温（摄氏度）。

yangtze.zip

包含 `yangtze.shp` 及相关文档，为长江流域主要河道的线性矢量数据。

section.zip

包含 `section.shp` 及相关文档，为连接宁夏固原（黄土高原）和贵州黔西（云贵高原）的直线，纵贯四川盆地。

hospitals.csv

四川省三甲医院的经纬度坐标列表。

cities.csv

研究区域部分城市的经纬度列表。

总体要求

根据题目的描述绘制地图或其他图表，每道题 10 分。将图像粘贴到答题区，或作为附件上传。除特殊要求的题目外，其他题目不需要在答题区填写文字。

题目

1. 绘制一张柱状图（或条形图），展示四川省各个地级行政区的面积大小 [7 分]。面积单位设为 **平方千米** [3 分]。
2. 已知西安的坐标为北纬 34.3° ，东经 109° 。绘制一张箱线图与小提琴图的**叠加图**（参见 Lab 2），展示西安到四川省各个地级行政区边界的最近距离的分布情况 [10 分]。提示：当只需要绘制一组小提琴图或箱线图时，`aes()` 中的 `x` 可以自定义。例如 `aes(x="Sichuan Cities", y=.....)`。

3. 绘制一张地图，展示重庆市境内有哪些城市（`cities.csv` 给定范围）距离主要河流（`yangtze.shp` 给定范围）的距离在 15 千米以上（注意单位）。要求：a) 地图上呈现河流，且河流不超出重庆市的边界范围 [3 分]；b) 地图上只呈现符合要求的城市并标注城市名称 [4 分]；c) 地图上呈现缓冲区，缓冲区需要合并以保持画面整洁 [3 分]。提示 1：可以先在草稿纸上画一下思路图，厘清 `buffer`、`intersection`、`spatial join` 和 `union` 一共有几步及其先后顺序。提示 2：`union` 会导致空间信息的部分损失，在这里只是为了保持画面简洁，所以应于最后一步完成。

4. 四川省南充市（Nanchong）位于四川盆地内部，阿坝藏族羌族自治州（Ngawa）位于青藏高原边缘。绘制两条半透明的密度曲线图（在同一张图表中），对比展示南充市和阿坝州的海拔分布情况 [每条密度曲线 4 分]，并在图中标记出两个行政区各自的平均海拔 [各 1 分]（提示：先用 `mask` 方法将栅格数据切割到行政区的范围）。

5. 绘制一张地图，展示四川省各地级行政区拥有的三甲医院数量。要求：a) 地图能正确地呈现数据 [3 分]；b) 地图中需展现医院的所在位置 [2 分]；c) 自定义一个多色配色方案，并符合色盲保护原则 [3 分]；d) 地图的色带需要有图例，色带图例的数字标签不少于 4 个，且需符合数据特征 [2 分]。

6. 绘制一条起于宁夏固原，终于贵州黔西（沿着 `section.shp`）的地形剖面图 [3 分]，该图横坐标单位为千米 [1 分]。采用宽度为 20 的移动窗口平滑剖面图的曲线 [2 分]。这条线沿途经过了丰富的地形区，请结合初中地理常识（可对照网上的地形图），在剖面上用半透明的阴影区标注秦岭和大娄山（四川盆地的南缘）的大致范围 [各 2 分]。（提示：用 `ggsave` 输出图像时可以适当拉长 `width` 以容纳更多的标签）。

7. 绘制地图展现 2009 年某日（`temp.tif`）最高温度在 20~26 摄氏度之间的区域（包含端点） [5 分]。在给定城市（`cities.csv`）中，筛选当日最高温度在 20~26 摄氏度之间的城市，将它们添加到这张地图上（可以不标注城市名） [4 分]。（提示：`temp.tif` 为不规则形状，在转换为数据框后可以先清除空值）

8. 绘制地图展现 2009 年某日（`temp.tif`）最高温度在空间上的变化最剧烈的区域 [6 分]（提示：类比到地形图上，变化最剧烈的区域为坡度最陡的区域）。为地图合理添加图例 [3 分]。在答题区用一句话总结这些区域在四川盆地内的分布特征 [1 分]。

9. 绘制一幅四川省凉山彝族自治州（Liangshan）的地形图。要求：a) 合理地定义色带并添加图例 [3 分]；b) 地形边界不超出自治州行政区的边界 [3 分]；c) 添加正午的 `hillshade` 以增强对地貌的视觉呈现 [4 分]。（提示：`aspects` 转换成数据框后可能也需要清除空值，才能被 `mask` 到凉山州的范围）

10. Lab 6 开放性作业的主题是否已完成构思？需要什么样的地理数据（`Shapefile` 或栅格数据）？是否需要地理数据上的协助？如有，请及时发邮件到 `yes@cugb.edu.cn`，不要等到 Lab 4 的截止日期。[调查性问题，请如实填写，10 分]

提交内容

将 1–9 题的图表或地图分别填入学习通答题区的相应位置。有文字回答要求的题目在相应位置填写答案。

所有用到的 R 代码汇总到一个文档里，作为附件上传到第一题。R 文档请命名为“Lab4+姓名+学号”，比如“Lab4 张三 1010101010.R”。

截止时间：2023 年 12 月 24 日北京时间 23:59。