Handling Geospatial Data with R-Programming

Adam Quek

2025-07-14

Base Map

Maps used in spatial analysis generally fall into two categories: **raster** and **vector**.

- Raster maps resemble common image formats (e.g. `.jpg`, `.png`), where the map is stored pixel by pixel, each representing colour or intensity. While visually rich, raster maps are less suitable for direct object manipulation.
- Vector maps, in contrast, represent geographical features (points, lines, polygons) mathematically. This makes them highly scalable and suitable for analysis, such as identifying boundaries, aggregate population data, or linking to spatial attributes.

For thie session, we will use publicly available administrative boundaries from data.gov.sg. Specifically, we'll use the **URA Master Plan 2019 Subzone Boundary (NO Sea)** dataset in **GeoJSON** format.

GeoJSON (Geospatial JavaScript Object Notation) is a widely used open format that stores both geometries and associated attributes in a JSON structure. It is highly compatible with the `sf` package in R.

```
# load the simple features (sf) package
library(sf)
```

```
## Linking to GEOS 3.11.2, GDAL 3.8.2, PROJ 9.3.1; sf_use_s2() is TRUE
```

```
# Read GeoJSON file into an sf object
geo_data <- st_read("./data/MasterPlan2019SubzoneBoundaryNoSeaGEOJSON.geojson")
```

```
## Reading layer 'MasterPlan2019SubzoneBoundaryNoSeaGEOJSON' from data source
##   'C:\hsrc\admin\gms5204\data\MasterPlan2019SubzoneBoundaryNoSeaGEOJSON.geojson'
##   using driver 'GeoJSON'
## Simple feature collection with 332 features and 2 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY, XYZ
## Bounding box:  xmin: 103.6057 ymin: 1.158699 xmax: 104.0885 ymax: 1.470775
## z_range:       zmin: 0 zmax: 0
## Geodetic CRS:  WGS 84
```

`geo_data` is a geospatial vector data with both **geometrical** and **attribute** information.

- It contains 332 polygon features, each corresponding to a **subzone** defined under Singapore's **URA Master Plan 2019**.
- The geometry type is `MULTIPOLYGON`, indicating that some subzones may consist of multiple disjoint polygons.
- The spatial data uses the **WGS 84** coordinate reference system (CRS), which is standard for global geographic data.

```r
# Glimpse of first feature
geo_data[1,]
```

```
## Simple feature collection with 1 feature and 2 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XYZ
## Bounding box:  xmin: 103.8013 ymin: 1.280037 xmax: 103.8177 ymax: 1.284103
## z_range:       zmin: 0 zmax: 0
## Geodetic CRS:  WGS 84
##    Name
## 1 kml_1
##
## 1 <center><table><tr><th colspan='2' align='center'><em>Attributes</em></th></tr><tr bgcolor="#E3E3F
##                          geometry
## 1 MULTIPOLYGON Z (((103.8145 ...
```

Each row in the `geo_data` object corresponds to a single **subzone feature**, consisting of two components:

1. `Description`: This field stores HTML-formatted metadata, which includes key identifiers such as:
    - `SUBZONE_N`: Subzone name (e.g., *DEPOT ROAD*)
    - `PLN_AREA_N`: Planning area (e.g., *BUKIT MERAH*)
    - `REGION_N`: Region name (e.g., *CENTRAL REGION*)
    - Other administrative and system-related codes (e.g., `SUBZONE_C`, `PLN_AREA_C`, `REGION_C`, `INC_CRC`, and `FMEL_UPD_D` for update timestamp)

    These attributes are essential for joining this geospatial layer with population, infrastructure, or health datasets based on spatial identifiers.
2. `geometry`: This contains the actual **MULTIPOLYGON Z** geometry, specifying a set of (longitude, latitude, altitude) points that enclose the boundaries of the subzone. The `Z` indicates that altitude (z-dimension) is recorded, although in most map-based applications, it's typically ignored.

This structure allows for flexible manipulation and spatial analysis: for example, you can visualize subzones, filter by planning areas, or spatially join this map to population or facility point data.

Description wrangling

Parsing description from html format into viewable table format.

```r
library(sf)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(rvest)
library(purrr)
library(stringr)
library(tidyr)
```

```r
# Function to parse Description column
parse_description <- function(html_string) {
  html <- read_html(html_string)
  keys <- html %>% html_nodes("th") %>% html_text(trim = TRUE)
  values <- html %>% html_nodes("td") %>% html_text(trim = TRUE)

  # Make lengths equal if mismatch occurs
  n <- min(length(keys), length(values))
  keys <- keys[1:n]
  values <- values[1:n]

  setNames(as.list(values), keys)
}


# Apply parser to all rows and combine as tibble
description_df <- geo_data %>%
  st_drop_geometry() %>%
  mutate(parsed = map(Description, parse_description)) %>%
  select(parsed) %>%
  unnest_wider(parsed)

# Combine with geometry
geo_data_clean <- bind_cols(description_df, st_geometry(geo_data)) %>%
  st_as_sf()
```

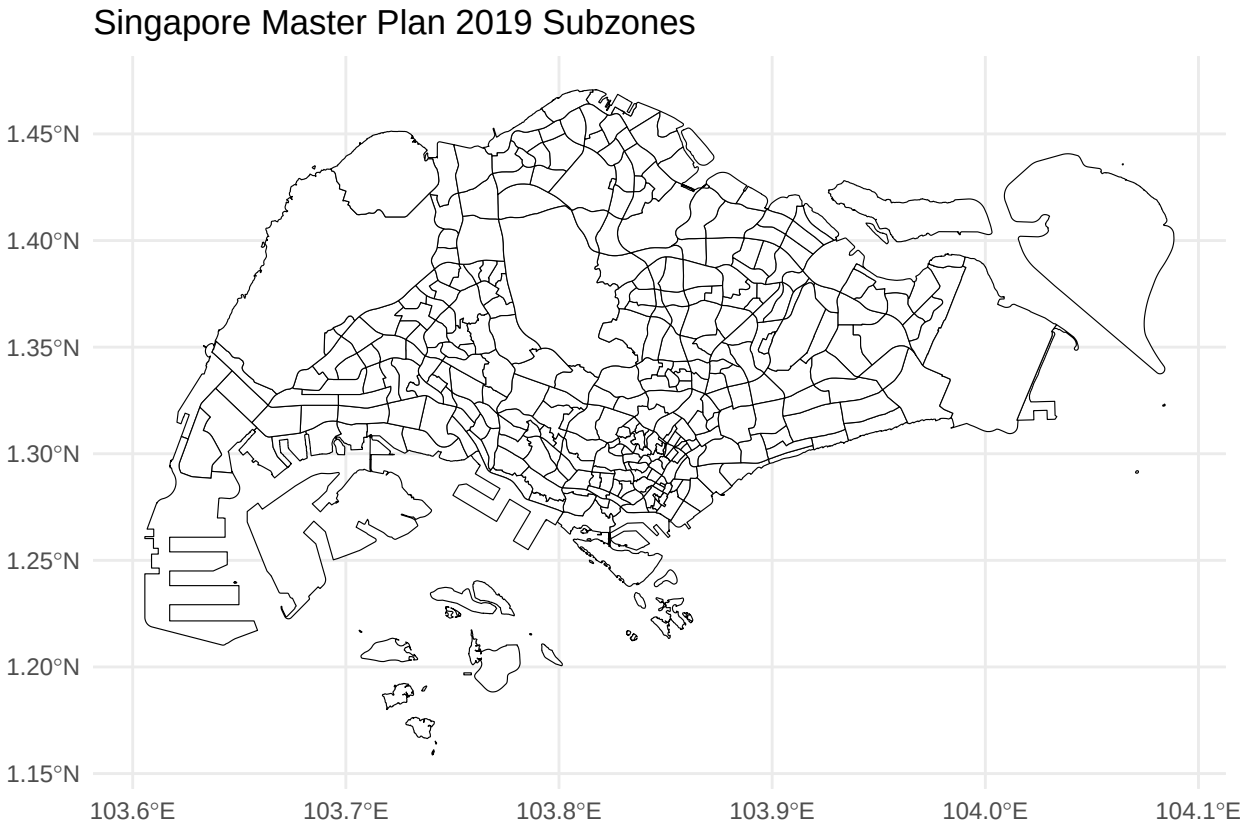
```
## New names:
## * `` -> `...11`
```

```r
# Drop Z/M dimensions
geo_data_clean_2d <- st_zm(geo_data_clean, drop = TRUE, what = "ZM")

# View result
print(head(geo_data_clean_2d))
```

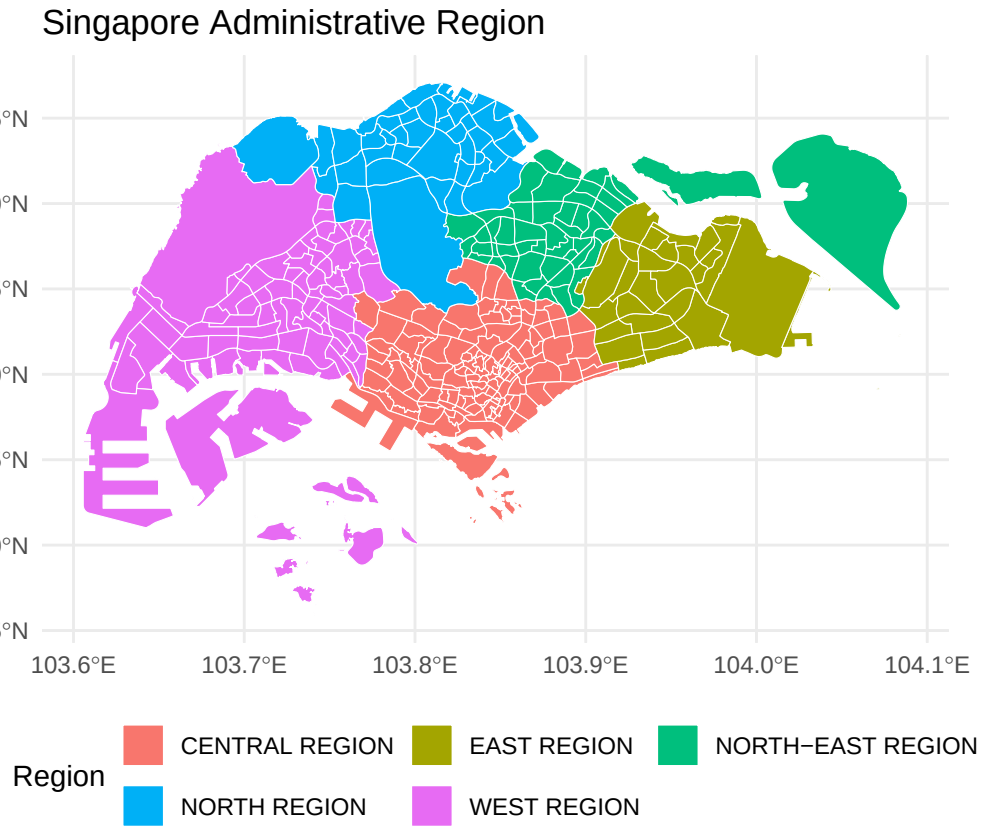
```
## Simple feature collection with 6 features and 10 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 103.8013 ymin: 1.274155 xmax: 103.8532 ymax: 1.286517
## Geodetic CRS:  WGS 84
## # A tibble: 6 x 11
##   Attributes SUBZONE_NO    SUBZONE_N SUBZONE_C CA_IND      PLN_AREA_N PLN_AREA_C
##   <chr>      <chr>         <chr>     <chr>     <chr>       <chr>      <chr>
## 1 12         DEPOT ROAD    BMSZ12    N         BUKIT MERAH BM         CENTRAL R~
## 2 2          BUKIT MERAH   BMSZ02    N         BUKIT MERAH BM         CENTRAL R~
## 3 3          CHINATOWN     OTSZ03    Y         OUTRAM      OT         CENTRAL R~
## 4 4          PHILLIP       DTSZ04    Y         DOWNTOWN C~ DT         CENTRAL R~
## 5 5          RAFFLES PLACE DTSZ05    Y         DOWNTOWN C~ DT         CENTRAL R~
## 6 4          CHINA SQUARE  OTSZ04    Y         OUTRAM      OT         CENTRAL R~
## # i 4 more variables: REGION_N <chr>, REGION_C <chr>, INC_CRC <chr>,
## #   ...11 <MULTIPOLYGON [°]>
```

Simple Visualisation of Base Map

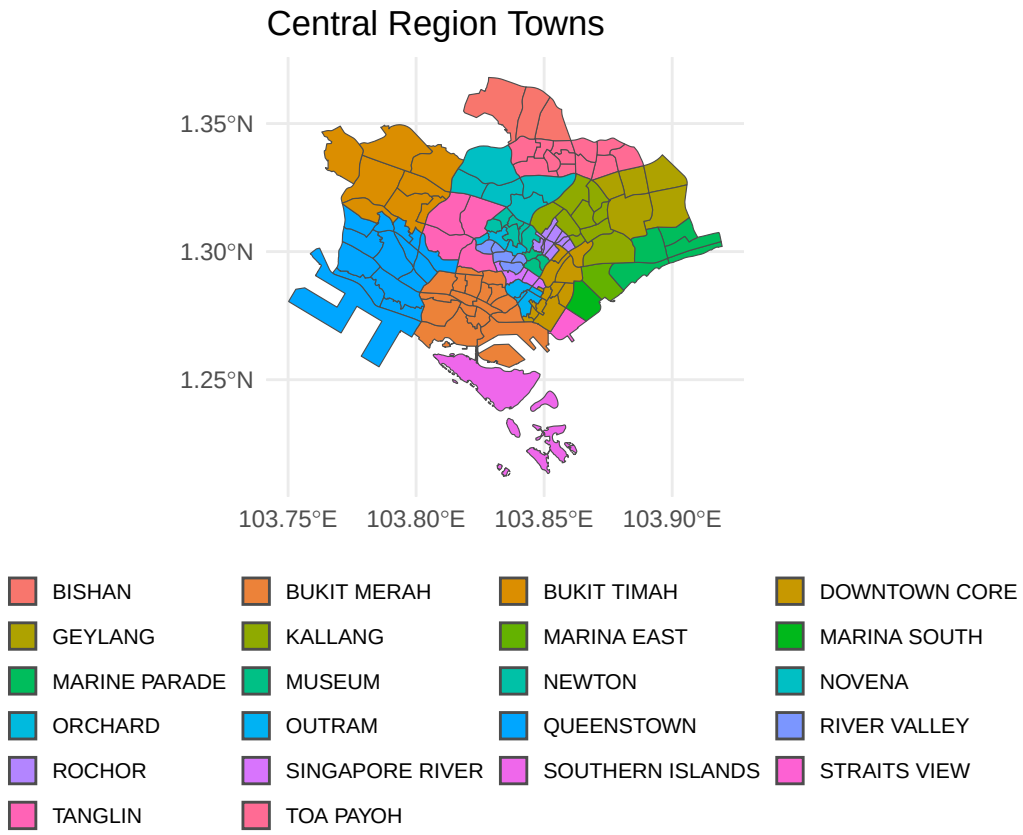
```
library(ggplot2)

# visualisation of all 332 subzones in Singapore
ggplot(geo_data_clean_2d) +
  geom_sf(fill = "white", color = "black", size = 0.2) +
  theme_minimal() +
  labs(title = "Singapore Master Plan 2019 Subzones")
```



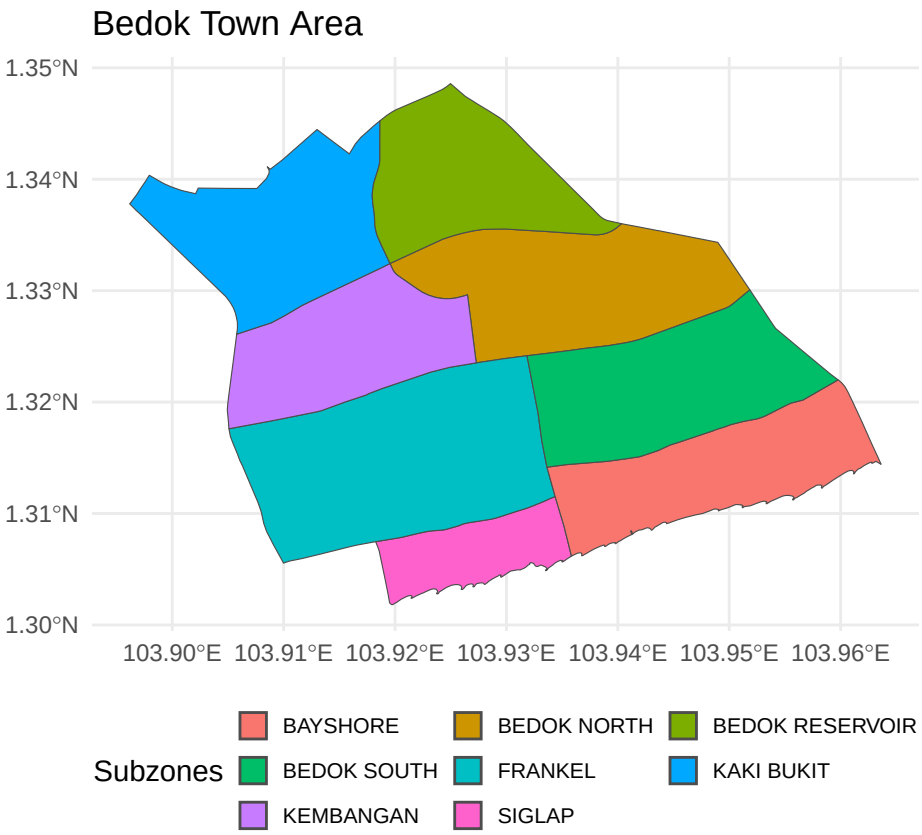
```r
# visulisation of subzones by region
ggplot(geo_data_clean_2d) +
  geom_sf(aes(fill = PLN_AREA_C), color = "white") +
  theme_minimal() +
  labs(title = "Singapore Administrative Region", fill = "Region") +
  theme(legend.position = "bottom") +
  guides(fill = guide_legend(nrow = 2, byrow = TRUE))
```



```r
# plot all towns in central region
central <- geo_data_clean_2d %>% filter(REGION_N == "CR")

ggplot(central) +
  geom_sf(aes(fill = CA_IND), color = "grey30") +
  theme_minimal() +
  labs(title = "Central Region Towns", fill = "") +
  theme(legend.position = "bottom",
        legend.key.size = unit(0.4, "cm"),
        legend.text = element_text(size = 8)) +
  guides(fill = guide_legend(ncol = 4, byrow = TRUE))
```



```r
# plot all subzones in Bedok town area
bedok <- geo_data_clean_2d %>% filter(CA_IND == "BEDOK")

ggplot(bedok) +
  geom_sf(aes(fill = SUBZONE_NO), color = "grey30") +
  theme_minimal() +
  labs(title = "Bedok Town Area", fill = "Subzones") +
    theme(legend.position = "bottom",
        legend.key.size = unit(0.4, "cm"),
        legend.text = element_text(size = 8)) +
  guides(fill = guide_legend(ncol = 3, byrow = TRUE))
```



Point Feature

Base maps provide the foundational context for interpreting spatial data. However, to derive meaningful insights, we often overlay them with additional layers—such as point, line, or polygon features—that represent real-world entities or phenomena.

Think of using Google Maps to find directions: the app overlays a path (line feature) on top of the base map and highlights the origin and destination (point features). These points help us understand spatial relationships like distance, accessibility, and proximity.

Similarly, in healthcare geospatial analysis, point features represent precise locations of interest. These can include:

- Hospitals and clinics
- Ambulance dispatch points (SCDF fire stations)
- COVID-19 testing or vaccination sites
- Locations of reported cases (e.g., dengue, tuberculosis)
- Residential addresses of patients (with proper anonymization)

By overlaying point features onto a base map (e.g., town subzones), we can answer important questions:

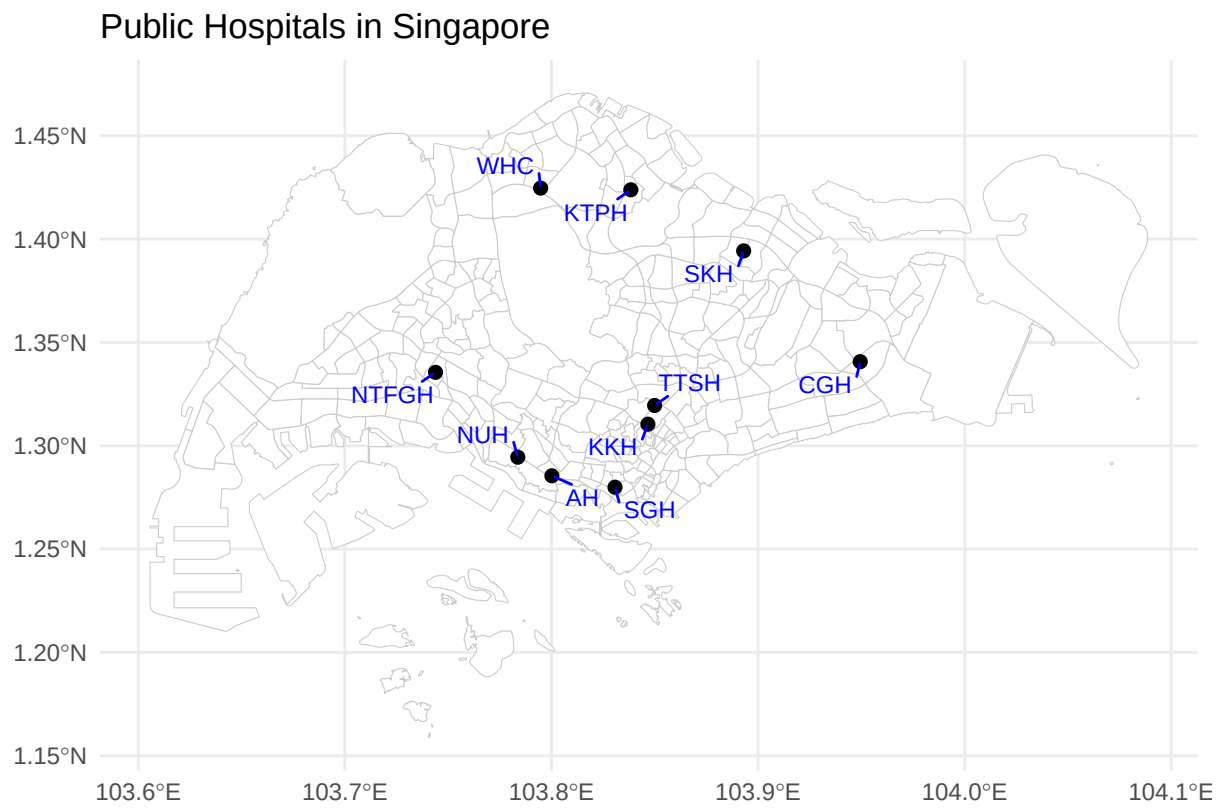
- Are healthcare facilities equitably distributed across regions?
- What is the nearest facility to a high-risk population?
- How clustered are emergency calls within a specific district?

In R, such point features can be represented using sf objects with POINT geometry, created either from latitude-longitude coordinates (e.g., CSVs) or directly from shapefiles or GeoJSON layers.

```
library(ggrepel)

# Coding location of hospital as data frame
hospitals_df <- data.frame(
  hospital_name = c("sgh", "ttsh", "cgh", "ntfgh", "ktph", "skh", "whc", "ah", "nuh", "kkh"),
  hospital_latitude = c(1.279939688, 1.319499928, 1.340695382, 1.335533826, 1.423864587,
                        1.394299417, 1.424681383, 1.28548113, 1.294488602, 1.310448532),
  hospital_longitude = c(103.8307218, 103.8499006, 103.9494212, 103.7438655, 103.8384081,
                         103.8930013, 103.7947438, 103.8001809, 103.7836841, 103.8466501)
)

# Convert hospitals_df to sf POINT object
hospitals_sf <- st_as_sf(
  hospitals_df,
  coords = c("hospital_longitude", "hospital_latitude"),
  crs = 4326  # WGS84
)

# Visualisation of hospital location onto basemap
ggplot() +
  geom_sf(data = geo_data_clean_2d, fill = "white", color = "grey80") +
  geom_sf(data = hospitals_sf, colour="black", size = 2) +
  geom_text_repel(
    data = hospitals_sf,
    aes(geometry = geometry, label = toupper(hospital_name)),
    colour = "blue",
    stat = "sf_coordinates",  # Convert sf geometry to x/y for plotting
    size = 3,
    min.segment.length = 0,
```

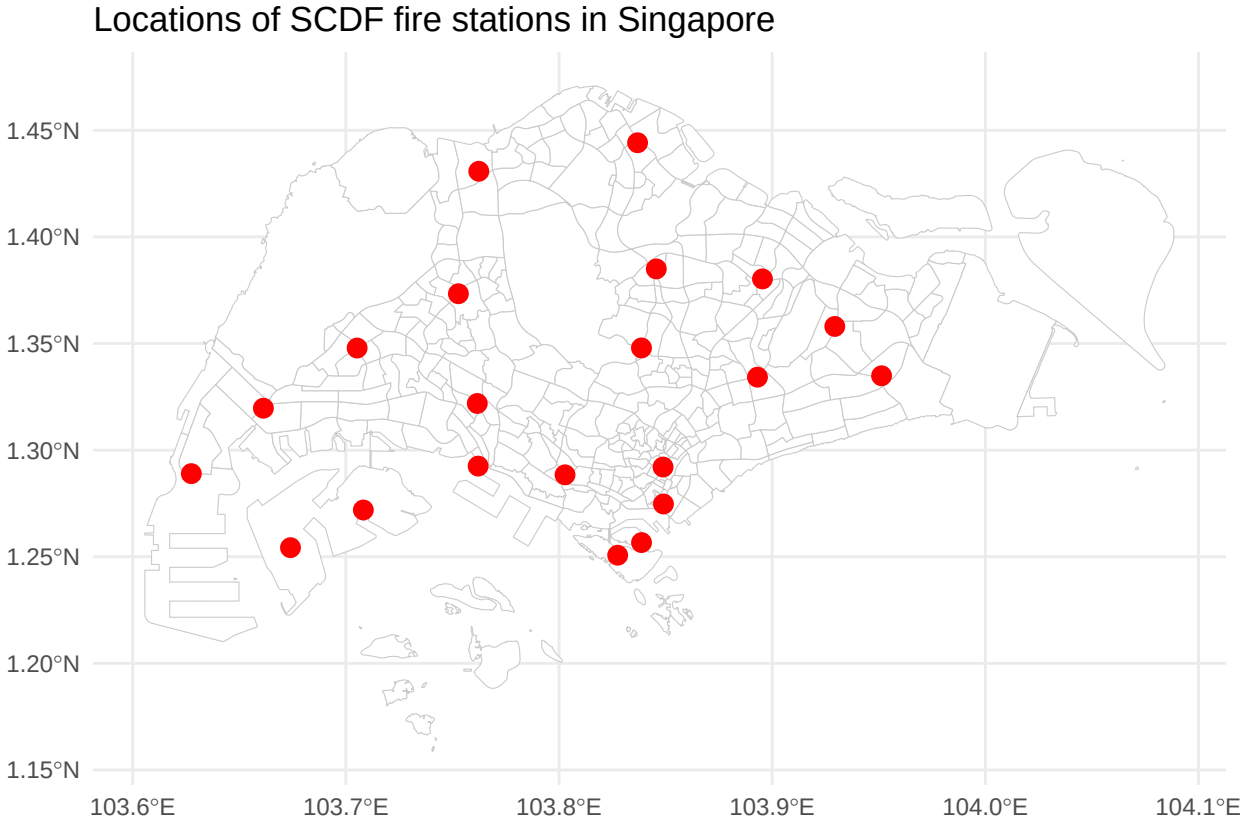
```
    box.padding = 0.3
  ) +
  theme_minimal() +
  labs(title = "Public Hospitals in Singapore", x = "", y = "")
```

```
## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may not
## give correct results for longitude/latitude data
```



Some of the point features may be directly downloadable from open source data. For instance, locations of SCDF fire stations, where ambulances are primarily dispatched from, are available as KML format on Fire Stations (KML). KML stands for **Keyhole Markup Language**, which is used to represent geographic data and commonly used in Google products and widely compatible with any GIS software.

```
# Load KML file (replace path if needed)
fire_stations <- st_read("./data/FireStationsKML.kml", quiet = TRUE)

ggplot() +
  geom_sf(data = geo_data_clean_2d, fill = "white", color = "grey80", size = 0.2) +
  geom_sf(data = fire_stations, colour="red", size=3) +
  theme_minimal() +
  labs(title = "Locations of SCDF fire stations in Singapore")
```



Line Features/ Calculating distances between point

To understand **spatial proximity** between entities, e.g. *distance between a patient's residence and the nearest clinic*, we often use line features or distance calculations between point geometries. These measurements are foundational for evaluating accessibility, service coverage, and healthcare equity.

Why it matters in healthcare?

- Access to care: Distance to the nearest hospital or polyclinic can affect patient adherence, follow-up rates, or emergency response.
- Health equity: Long travel distances may disproportionately affect the elderly, disabled, or low-income populations.
- Resource allocation: Helps planners identify underserved areas and optimize the placement of mobile clinics or new facilities.

Measuring Distance in R with `sf`

The following example demonstrates how to calculate the distance between patient locations and a set of hospitals, and identify the nearest facility for each patient. For accurate measurement in meters, we use Singapore's projected coordinate system **EPSG:3414 (SVY21)**.

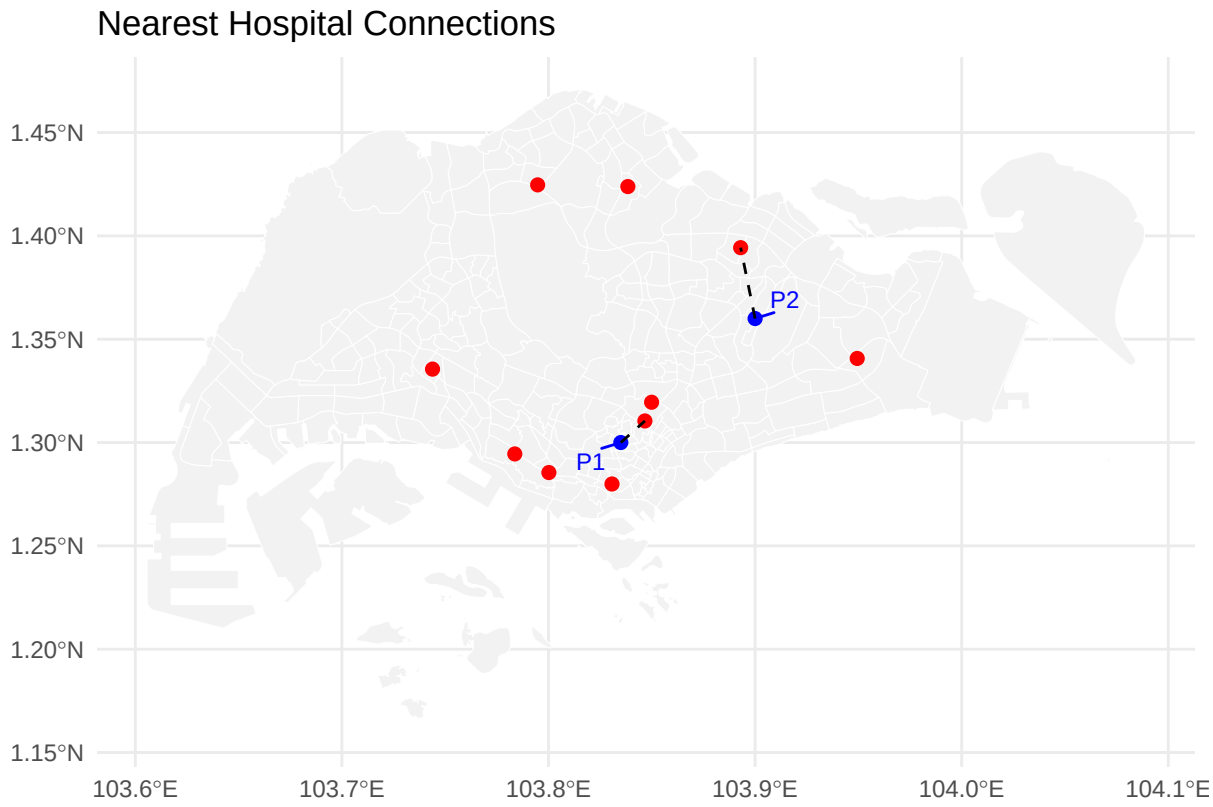
```
# Simulated patient locations (example)
patients_df <- data.frame(
  patient_id = c("P1", "P2"),
  lon = c(103.835, 103.900),
  lat = c(1.300, 1.360)
)

# Convert to sf object
patients_sf <- st_as_sf(patients_df, coords = c("lon", "lat"), crs = 4326)

# Ensure both geometries use the same projection
# Optionally, transform to a projected CRS (e.g., EPSG:3414 for Singapore) for meters
patients_proj <- st_transform(patients_sf, 3414)
hospitals_proj <- st_transform(hospitals_sf, 3414)

# Calculate distances (in meters) between patients and hospitals
dist_matrix <- st_distance(patients_proj, hospitals_proj)

# Convert to a data frame for easy viewing
dist_df <- as.data.frame(dist_matrix)
rownames(dist_df) <- patients_sf$patient_id
colnames(dist_df) <- hospitals_sf$hospital_name

print(dist_df)
```

```
##             sgh         ttsh           cgh        ntfgh          ktph
## P1  2268.69 [m] 2720.139 [m] 13505.655 [m] 10876.86 [m] 13701.565 [m]
## P2 11739.40 [m] 7151.335 [m]  5899.738 [m] 17585.38 [m]  9841.329 [m]
##              skh          whc            ah          nuh          kkh
## P1 12263.410 [m] 14496.28 [m]  4194.461 [m]  5743.43 [m] 1736.623 [m]
## P2  3871.807 [m] 13724.51 [m] 13831.241 [m] 14833.80 [m] 8079.134 [m]
```

We now determine the **nearest hospital** for each patient and draw a **line feature** connecting them.

```
nearest_hospitals <- apply(dist_df, 1, function(x) names(x)[which.min(x)])


lines_sf <- st_sfc(mapply(function(p, h) st_cast(st_union(p, h), "LINESTRING"),
                          st_geometry(patients_proj),
                          st_geometry(hospitals_proj[apply(dist_df, 1, which.min), ]),
                          SIMPLIFY = FALSE),
                   crs = 3414)

connections <- st_sf(
  patient_id = patients_sf$patient_id,
  nearest_hospital = nearest_hospitals,
  geometry = lines_sf
)

ggplot() +
  geom_sf(data = geo_data_clean_2d, fill = "grey95", color = "white") +
  geom_sf(data = hospitals_proj, color = "red", size = 2) +
  geom_sf(data = patients_proj, color = "blue", size = 2) +
  geom_sf(data = connections, linetype = "dashed", color = "black") +
  geom_text_repel(
    data = patients_proj,
    aes(geometry = geometry, label = patient_id),
    colour = "blue",
    stat = "sf_coordinates",  # Convert sf geometry to x/y for plotting
    size = 3,
    min.segment.length = 0,
    box.padding = 0.3
  ) +
  theme_minimal() +
  labs(title = "Nearest Hospital Connections", x="", y="")
```

```
## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may not
## give correct results for longitude/latitude data
```


Nearest Hospital Connections
1.45°N
1.40°N
1.35°N
1.30°N
1.25°N
1.20°N
1.15°N
103.6°E
103.7°E
103.8°E
103.9°E
104.0°E
104.1°E
P1
P2

Choropleth Mapping with External Attributes

In spatial analysis, especially in healthcare, maps become more insightful when external socioeconomic or demographic attributes are layered onto geographical units. This allows analysts to explore potential gradients, disparities, or clustering patterns in relation to health outcomes or service access.

One such indicator is the Socioeconomic Advantage Index (SAI) developed by Earnest et al. (2015) in the study "Derivation of indices of socioeconomic status for health services research in Asia". This index was specifically designed to adapt to the Singaporean urban context, based on 2015 Census data based on (i) proportion of residents in owner-occupied housing; (ii) educational attainment levels; (iii) median household income; and (iv) employment status.

```
sai_summary <- data.frame(
  planning_area = toupper(c(
    "Tanglin", "River Valley", "Newton", "Bukit Timah", "Marine Parade", "Novena",
    "Bishan", "Serangoon", "Pasir Ris", "Clementi", "Bedok", "Bukit Batok",
    "Queenstown", "Choa Chu Kang", "Jurong East", "Hougang", "Tampines",
    "Bukit Panjang", "Toa Payoh", "Ang Mo Kio", "Downtown Core", "Sembawang",
    "Geylang", "Kallang", "Yishun", "Bukit Merah", "Jurong West", "Sengkang",
    "Woodlands", "Rochor", "Outram", "Changi"
  )),
  sai = c(
    126.7, 123.7, 123.5, 122.2, 107.4, 105.8,
    103.4, 102.7, 100.2, 99.5, 98.8, 98.0,
    97.2, 95.7, 95.6, 95.5, 95.4,
    95.2, 95.1, 95.0, 95.0, 94.9,
    94.4, 94.4, 94.3, 94.3, 93.5, 93.5,
    93.3, 93.1, 91.5, 91.0
  )
)

print(sai_summary)
```

```
##    planning_area   sai
## 1         TANGLIN 126.7
## 2    RIVER VALLEY 123.7
## 3          NEWTON 123.5
## 4     BUKIT TIMAH 122.2
## 5   MARINE PARADE 107.4
## 6          NOVENA 105.8
## 7          BISHAN 103.4
## 8       SERANGOON 102.7
## 9       PASIR RIS 100.2
## 10       CLEMENTI  99.5
## 11          BEDOK  98.8
## 12    BUKIT BATOK  98.0
## 13     QUEENSTOWN  97.2
## 14 CHOA CHU KANG  95.7
## 15    JURONG EAST  95.6
## 16        HOUGANG  95.5
## 17       TAMPINES  95.4
## 18 BUKIT PANJANG  95.2
## 19      TOA PAYOH  95.1
## 20     ANG MO KIO  95.0
## 21 DOWNTOWN CORE  95.0
```

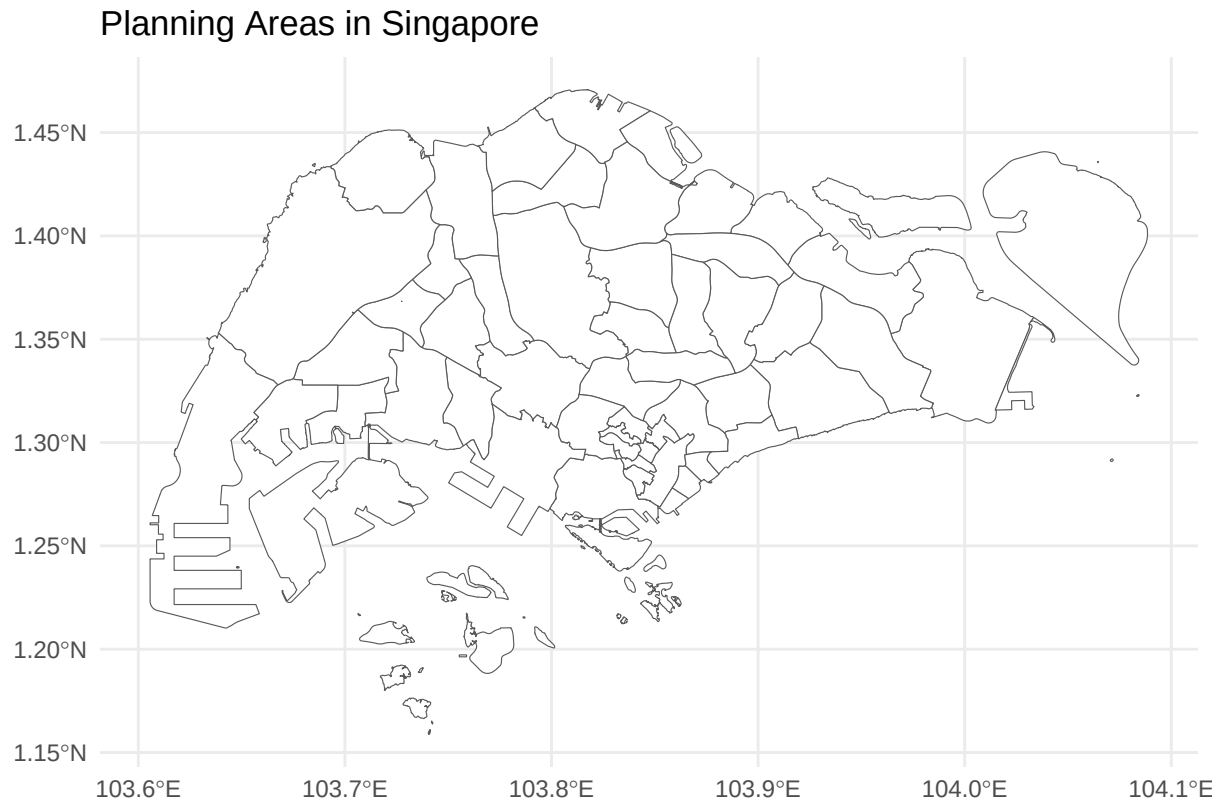
```
## 22     SEMBAWANG  94.9
## 23       GEYLANG  94.4
## 24       KALLANG  94.4
## 25        YISHUN  94.3
## 26   BUKIT MERAH  94.3
## 27   JURONG WEST  93.5
## 28      SENGKANG  93.5
## 29     WOODLANDS  93.3
## 30        ROCHOR  93.1
## 31        OUTRAM  91.5
## 32        CHANGI  91.0
```

Transforming the base-map from 332 subzones into 55 planning areas.

```
# check geometry validity
invalid <- !st_is_valid(geo_data_clean_2d)
sum(invalid)  # how many invalid geometries?
```

```
## [1] 6
```

```
# fix invalid geometries
geo_data_valid <- st_make_valid(geo_data_clean_2d)

# CA_IND corresponds to Planning Area Name (e.g., "BEDOK")
geo_data_pa <- geo_data_valid %>%
  group_by(CA_IND) %>%
  summarise(.groups = "drop")  # auto-union

ggplot(geo_data_pa) +
  geom_sf(fill="white") +
  theme_minimal() +
  labs(title = "Planning Areas in Singapore", x = "", y = "")
```



Merging SAI into the transformed map with planning area.

```
geo_data_sai <- geo_data_pa %>%
  left_join(sai_summary, by = c("CA_IND" = "planning_area"))

# check planning areas that are not matched
setdiff(unique(geo_data_pa$CA_IND), sai_summary$planning_area)
```

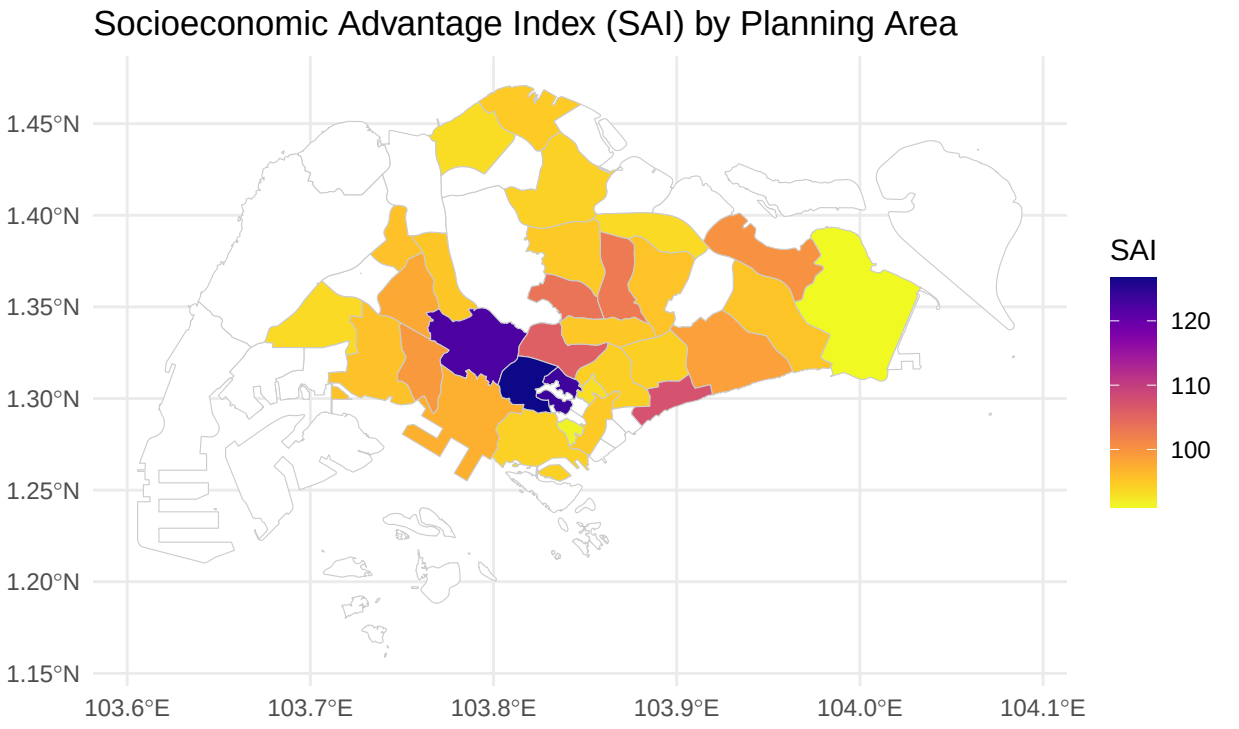
```
##  [1] "BOON LAY"                "CENTRAL WATER CATCHMENT"
##  [3] "CHANGI BAY"              "LIM CHU KANG"
##  [5] "MANDAI"                  "MARINA EAST"
##  [7] "MARINA SOUTH"            "MUSEUM"
##  [9] "NORTH-EASTERN ISLANDS"   "ORCHARD"
## [11] "PAYA LEBAR"              "PIONEER"
## [13] "PUNGGOL"                 "SELETAR"
## [15] "SIMPANG"                 "SINGAPORE RIVER"
## [17] "SOUTHERN ISLANDS"        "STRAITS VIEW"
## [19] "SUNGEI KADUT"            "TENGAH"
## [21] "TUAS"                    "WESTERN ISLANDS"
## [23] "WESTERN WATER CATCHMENT"
```

```
library(viridis)
```

```
## Loading required package: viridisLite
```

```
ggplot(geo_data_sai) +
  geom_sf(aes(fill = sai), color = "grey80") +
```

```
  scale_fill_viridis(
    name = "SAI",
    option = "C",
    direction = -1,
    na.value = "white"
  ) +
  labs(title = "Socioeconomic Advantage Index (SAI) by Planning Area") +
  theme_minimal()
```

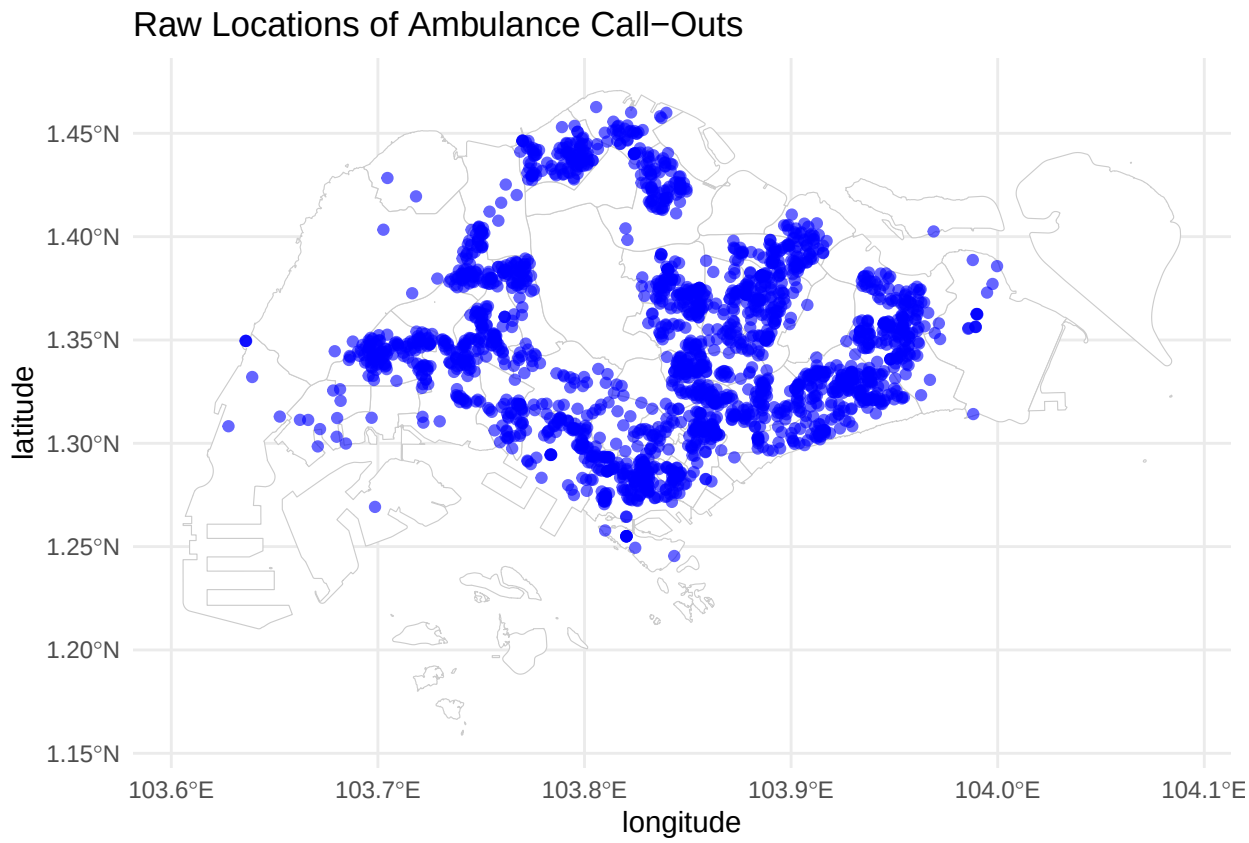


Hotspot Detection and Clustering

One of the key applications of geospatial analysis in healthcare is to **identify spatial clusters** or **hotspots** of events. This can be useful to detect high-need areas, allocate healthcare resources effectively, or evaluate equity of access.

In this example, we use a synthetic dataset representing **patient ambulance call-out locations** to demonstrate basic clustering and heatmap generation.

```
# Load and visualisa patient location
patient_location <- read.csv("./data/patient_location.csv")

# Basic map: raw patient locations
ggplot() +
  geom_sf(data = geo_data_pa, fill="white", colour="grey80")+
  geom_point(data = patient_location, aes(x = longitude, y = latitude),
             color = "blue", alpha = 0.6) +
  labs(title = "Raw Locations of Ambulance Call-Outs") +
  theme_minimal()
```

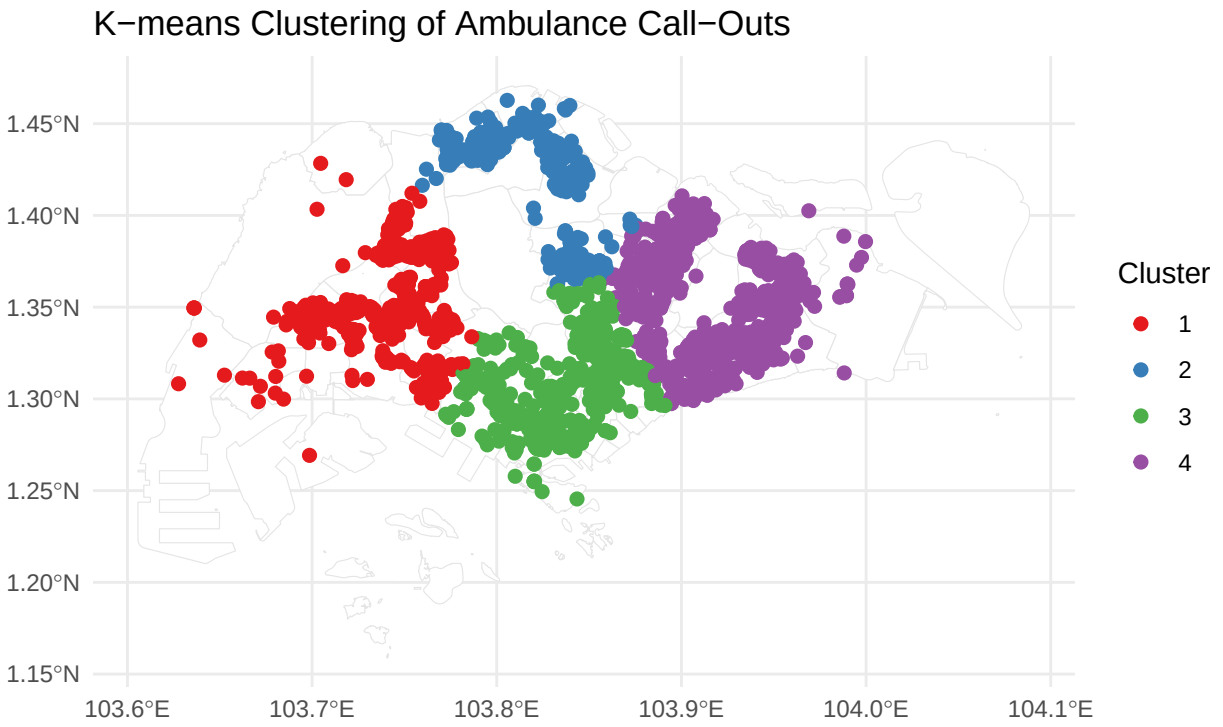


K-means clustering

K-means is a simple clustering technique that groups points into `k` clusters by minimizing the within-cluster variance. While it does not account for underlying spatial autocorrelation, it is useful as an intuitive first-pass for hotspot grouping.

```
# Convert to sf and project to SVY21 (EPSG:3414 in meters)
patients_sf <- st_as_sf(patient_location, coords = c("longitude", "latitude"), crs = 4326) %>%
```

```r
  st_transform(3414)

# Extract planar coordinates
coords <- st_coordinates(patients_sf)

# Perform K-means clustering (adjust centers as needed)
set.seed(42)
kmeans_result <- kmeans(coords, centers = 4)

# Attach cluster assignment
patients_sf$cluster <- as.factor(kmeans_result$cluster)

# Visualize clusters
ggplot() +
  geom_sf(data = geo_data_pa %>% st_transform(3414), fill = "white", color = "grey90") +
  geom_sf(data = patients_sf, aes(color = cluster), size = 2) +
  scale_color_brewer(palette = "Set1") +
  theme_minimal() +
  labs(title = "K-means Clustering of Ambulance Call-Outs", color = "Cluster")
```

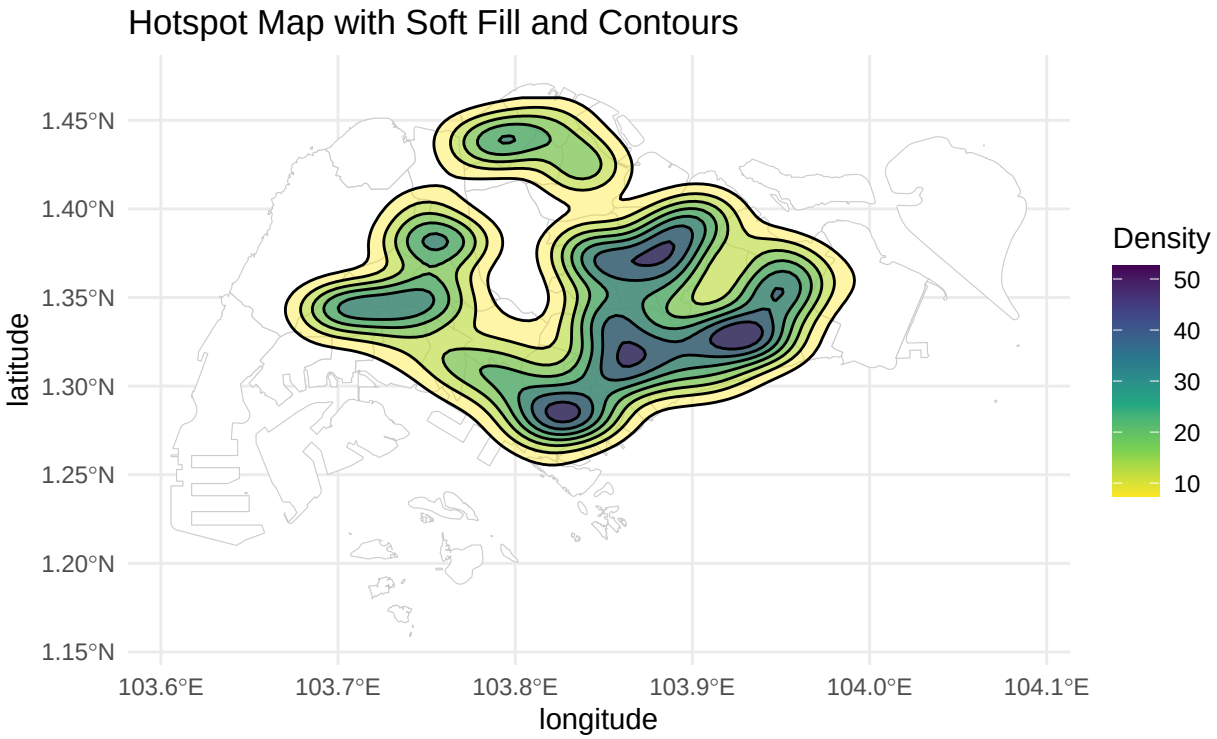


Kernel Density Estimation (KDE)

A more spatially appropriate approach is **Kernel Density Estimation**, which visualizes the intensity of point concentrations, producing a smooth hotspot surface.

```r
# Kernel density plot with filled contours
ggplot() +
  geom_sf(data = geo_data_pa %>% st_transform(4326), fill = "white", color = "grey80") +
  stat_density_2d(data = patient_location,
                  aes(x = longitude, y = latitude, fill = ..level..),
                  color = "black",
                  alpha = 0.4,
                  bins = 8,
                  geom = "polygon") +
  scale_fill_viridis_c(option = "D", direction = -1) +
  theme_minimal() +
  labs(title = "Hotspot Map with Soft Fill and Contours", fill = "Density")
```

```
## Warning: The dot-dot notation (`..level..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(level)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



Summary and Next Steps

In this hands-on session, you learned how to:

- Load and visualize geospatial polygon (planning area) and point data (patient and hospital locations)
- Perform spatial joins and transformations using the sf package
- Derive accessibility insights through nearest facility analysis
- Conduct basic spatial clustering with kmeans and visualize density hotspots using kernel density estimation

These methods represent an introductory workflow for spatial health equity analysis. They are especially useful for identifying underserved regions, planning mobile health deployments, or evaluating spatial determinants in healthcare utilization.