# Geospatial Data Carpentry Workshop for Urbanism: : **CHEATSHEET**

## Basics

### USING LIBRARIES

install.package("here")

library(here)

### ASSIGNMENT

x <- "apple"

### DATA TYPES

| | | |
|---|---|---|
| **as.character(x)** | "1" , "2", "one" | Character strings |
| **as.numeric(x)** | 1, 2, 1 | Numbers |
| **as.logical(x)** | TRUE, FALSE, T | Boolean |
| **as.factor(x)** | "1", "2", "1" Levels: "1", "2" | Strings with preset levels |

### VECTORS

Function **c()** joins elements of the same data type:

```
c(1, 2, 4, 5)
c(y, x)
```

**Combine vectors to create a new one**

### MISSING VALUES

| | |
|---|---|
| **is.na(x)** | Is missing |
| **!is.na(x)** | Not missing |

### OPERATORS

| | |
|---|---|
| x == y<br>x **!=** y | x equal to y<br>X not equal to y |
| x **>** y  x **<** y | x greater than y<br>x less than y |
| x **>=** y<br>x **<=** y | x greater than or equal to y<br>x less than or equal to y |
| x == y **&**<br>x == z | Logical 'AND' operator |
| x == y **\|**<br>x == z | Logical 'OR' operator |
| x **%in%** y | x belongs to a vector (y) |

Rbanism
Reproducibility.
Automation.
Scalability.

### EXPLORE DATASETS

| | |
|---|---|
| **head**(df,n) | First **n** rows of dataset **df** |
| **summary**(df) | Summary stats of **df** |
| **nrow**(df) | Number of rows in **df** |
| **ncol**(df) | Number of columns in **df** |

## Data manipulation

### DPLYR

**Pipes**

**x %>% f(y)** or **x |> f(y)** becomes **f(x,y)**

### SUBSET & FILTER

Select columns by name.

df %>% **select**(variables)

Extract rows meeting logical condition

df %>% **filter**(condition)

### CREATE NEW VARIABLE

Compute new columns

df %>% **mutate**(x=mean(y))

### SUMMARISE

Summarize data into summary table

df %>% **summarize**(x=mean(y))

### GROUP CASES

to create summaries by category

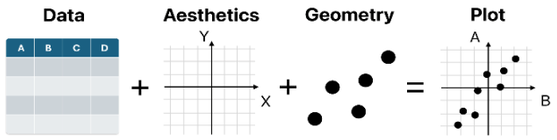
```
df %>%
group_by(variable) %>%
summarize(x=mean(y)) %>%
ungroup()
```

**Remove grouping**

## Data visualization

### GGPLOT2

**ggplot2** is based on the grammar of graphics - idea that plots are build based on three components: data set, coordinate system, and geoms—visual marks that represent data points.




**data**

**aesthetics**

```
ggplot(dataset,
       aes(x=variable) +
       geom_histogram()
```

**geometry**

The **geom_** functions define shape of a plot.

**SCATTER PLOT**
ggplot(df, aes(x = var1, y = var2) +
**geom_point()**

**HISTOGRAM**
ggplot(df, aes(x = var1) +
**geom_histogram()**

**BAR CHART**
ggplot(df, aes(x = var1, y = var2) +
**geom_col()**

### TITLES AND LABELS

**labs()** function allows naming axes, adding titles and useful legend names

```
plot1 +
labs( title = "Plot title",
      subtitle = "Plot subtitle"
       x  = "Axis X",
       y = " Axis Y",
      color = "Legend title ")
```

## Vector data

### SF BASICS

**st_read**(dsn, layer, ...) Read file or database vector dataset as a sf object

**st_geometry_type**(x, by_geometry = TRUE) Return the geometry type of an object

**st_crs**(x, …) Set or retrieve coordinate reference system (CRS) from an sf object

**Long output with all CRS info**

**Short output with specific parts of CRS**

**st_crs**(x, …)$Name – Get CRS name
**st_crs**(x, …)$epsg – Get EPSG code

**st_bbox**(obj, ...) Return bounding box of an sf object as an object of class bbox with xmin, ymin, xmax and ymax values

**st_transform**(x, crs, ...) Convert coordinates of an sf, sfc, sfg or bbox object

**st_length**(x, ...) Compute the length of a LINESTRING or MULTILINESTRING geometry in a projected CRS like Amersfoort / RD New (EPSG:28992)

**st_write**(obj, dsn, layer = NULL, ...) Write sf object to file

### VISUALISING SF OBJECTS

**geom_sf**() visualise sf objects with ggplot2

**coord_sf**() ensures that all layers use the same CRS, either specified with the crs parameter or taken automatically from the first layer that defines a CRS

**sf object**

**No need to specify x and y**

```
ggplot(data) +
 geom_sf() +
 coord_sf(datum = st_crs(28992))
```

**rainbow**(n) Create a vector of n colors, optionally customized with the palette parameter (e.g., palette = "viridis")

# Geospatial Data Carpentry Workshop for Urbanism: : **CHEATSHEET**

## Raster Data

### TERRA BASICS

**describe**(x, ...) Describe the properties
of spatial data in a file.

**rast**(x, nrows, ncols,…) Create a SpatRaster, from scratch, from a filename, or from another object.

**summary**(x, …) Compute summary statistics (min, max, mean, and quartiles). A sample of cells is used for very large files.

**values**(x,…) Get the cell values of a SpatRaster or the attributes of a SpatVector.

### DATA WRANGLING

If TRUE, coordinates are included

**as.data.frame**(x, xy=FALSE, geom, na.rm=NA, …) Coerce a SpatRaster or SpatVector to a data.frame.

**minmax**(x, …) and **setminmax**(x, …) Get or compute the min and max cell values.

**nlyr**(x, …) Get the number of rows (**nrow**), columns (**ncol**), cells (**ncell**), layers (**nlyr**), resolution (**res**), and other dimensions of a SpatRaster.

**ext**(x,…) Get a SpatExtent of a SpatRaster, SpatVector, or other spatial objects.

### PROJECTIONS

**crs**(x, …) Get or set the coordinate reference system of a SpatRaster or SpatVector.

**project**(x, y, …) Change the coordinate reference system ("project") of a SpatVector, SpatRaster or a matrix with coordinates.

### PLOT

**plotRGB**(x,filename,…) Make a Red-Green-Blue plot based on three layers in a SpatRaster.

### EXPORT

**writeRaster**(x,filename,…) Write a SpatRaster to a file.

## Visualisation

**ggplot2::geom_raster**(x, aes(fill=z), …) Draw a raster plot.

**ggplot2::coord_equal**(ratio = 1, xlim = NULL, ylim = NULL, …) Cartesian coordinates with fixed "aspect ratio"

**Terrain.colors**(n, alpha, rev=FALSE) Create a vector of n contiguous colors

## Open Street Map

### BOUNDING BOX

With the OSMdata package, it is possible to geocode a spatial text using the Nominatim API. The function `getbb` returns the coordinates of its bounding box: xmin, xmax, ymin and ymax.

```
osmdata::getbb("place name")
```

### OVERPASS QUERY

To extract and download Open Street Map (OSM) data into R, we access the Overpass API using a query, to which we add OSM features defined by hierarchical tags called keys and values. To download data about greenhouses for example, the key is "building" and the value "greenhouse".

```
osmdata::opq(bbox) |>
  add_osm_feature(key, value)|>
  osmdata_sf()
```

Format of resulting object (sf object)

The result of this query can contain **points**, **lines** and/or **polygons**, each described by a data frame.

## Interactive mapping

The **leaflet** package provides a way to create map with interactive features such as zoom, popups, image overlay, etc.


```
leaflet(x) |>
  addTiles() |>
  addPolygons()
```

Background map

Added geometries from x

## Geoprocessing

### BUFFER

A buffer corresponds to a circular polygon around an 'x' feature with a specified distance 'dist'

```
sf::st_buffer(x,dist)
```

### UNION

A union corresponds to the combination of polygons by removing internal boundaries

```
sf::st_union(x,y,...) |>
  sf::st_cast(to = "POLYGON") |>
  sf::st_as_sf()
```

Format of resulting object

Type of resulting object

### CENTROID

A centroid corresponds to the centre of mass of a geometric object.

```
sf::sf_use_s2(FALSE)
sf::st_centroid(x) |>
  sf::st_transform(.,crs)
```

Disables geographic projection

Reproject the resulting object

### INTERSECTION & JOIN

'Intersect' tests whether geometric objects x and y intersect each other.

'Intersection' performs the intersection and returns an object of the same type as x.

'Join' spatially matches x and y objects.

```
sf::st_intersection(x,y)
sf::st_join(x,y, left=T)
```

T for a left join
F for an inner join

### AREA

Computes the area of features x

```
sf::st_area(x) |>
  units::set_units(., km^2)
```

Specifies area unit

Rbanism
Reproducibility.
Automation.
Scalability.