Chapter 4

Data Import and Wrangling

This section shows how to import data, and perform some "data wrangling." Data wrangling refers to steps taken to make data more useful to downstream applications. Here we show approaches to clean up the raw data, i.e., remove missing values, filter out unnecessary information, and merge data sets. Data visualization and transformation of non-spatial data to spatial simple feature objects is introduced in the next section.

It is assumed that you already downloaded and unpacked the data into the `data` folder in your `R project` folder. If not, please do so before continuing (see Section 3.1).

4.1 Import Data into `R`

Food Access Research Atlas Data (2015 Data Set)

As mentioned in Section 3.2, the data is compiled into an Excel workbook. Excel workbooks (`.xlsx` format) can be imported directly into `R` by several packages, including `xlsx`, `XLconnect`, `read_xlsx`, and `openxlsx`. In this section we are using the package `openxlsx`. In contrast to the `xlsx` and `XLConnect` packages, `openxlsx` does not depend on `Java`. For detailed information on the `openxlsx` package please consult the documentation available:

- https://www.rdocumentation.org/packages/openxlsx/versions/4.2.3
- https://cran.r-project.org/web/packages/openxlsx/openxlsx.pdf

The `openxlsx` package provides functions to work with Excel worksheets, including functions to extract the of names of worksheets as well as to import selected worksheets.

Explore the structure of the workbook

The function `openxlsx::getSheetNames()` extracts the names of Excel worksheets.

```r
openxlsx::getSheetNames("data/FoodAccess2015.xlsx")
```

```
## [1] "Read Me"                    "Variable Lookup"
## [3] "Food Access Research Atlas"
```

The output shows that the workbook has 3 worksheets:

1. Read Me
2. Variable Lookup
3. Food Access Research Atlas

Import relevant sheets

The worksheet **`Food Access Research Atlas`** contains the data we are interested in. The following code imports this worksheet as a data frame and assigns the data frame to the object **`FoodAccess2015`**. Note, this is a large file. Reading it into **`R`** will take a while. Be patient.

```
FoodAccess2015 <- openxlsx::read.xlsx("data/FoodAccess2015.xlsx",
                                      sheet = "Food Access Research Atlas")
```

Alternatively, the index number of the worksheet can be used:

```
FoodAccess2015 <- openxlsx::read.xlsx("data/FoodAccess2015.xlsx",
                                      sheet = 3)
```

Next, the structure of the data frame is explored with **`str()`**. The output of the following code lists the first 20 variables (set by the argument **`list.len = 20`**). If all variables should be displayed, the argument would be **`list.len = ncol()`**.

```
str(FoodAccess2015, list.len = 20, strict.width = "cut")
```

```
## 'data.frame':    72864 obs. of  147 variables:
##  $ CensusTract         : chr  "01001020100" "01001020200" "01001020300" "0100"..
##  $ State               : chr  "Alabama" "Alabama" "Alabama" "Alabama" ...
##  $ County              : chr  "Autauga" "Autauga" "Autauga" "Autauga" ...
##  $ Urban               : num  1 1 1 1 1 1 1 0 0 0 ...
##  $ POP2010             : num  1912 2170 3373 4386 10766 ...
##  $ OHU2010             : num  693 743 1256 1722 4082 ...
##  $ GroupQuartersFlag   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ NUMGQTRS            : num  0 181 0 0 181 0 36 0 0 14 ...
##  $ PCTGQTRS            : num  0 0.0834 0 0 0.0168 ...
##  $ LILATracts_1And10   : num  0 0 0 0 0 0 1 0 0 0 ...
##  $ LILATracts_halfAnd10: num  0 0 0 0 0 0 1 0 0 0 ...
##  $ LILATracts_1And20   : num  0 0 0 0 0 0 1 0 0 0 ...
##  $ LILATracts_Vehicle  : num  0 0 0 0 0 0 1 0 0 0 ...
##  $ HUNVFlag            : num  0 0 0 0 1 0 1 1 0 0 ...
##  $ LowIncomeTracts     : num  0 0 0 0 0 0 1 0 0 0 ...
##  $ PovertyRate         : num  10 18.2 19.1 3.3 8.5 ...
##  $ MedianFamilyIncome  : num  74750 51875 52905 68079 77819 ...
##  $ LA1and10            : num  1 0 1 1 1 1 1 0 0 1 ...
##  $ LAhalfand10         : num  1 1 1 1 1 1 1 0 0 1 ...
##  $ LA1and20            : num  1 0 1 1 1 1 1 0 0 0 ...
##   [list output truncated]
```

The data frame has 72,864 observations (rows) and 147 variables (columns). Note that the variable **`CensusTract`** is a character string (**`chr`**). Furthermore, variables such as **`Urban`** or **`POP2010`** were imported as real numbers (double precision numbers, **`num`**).

Census Tracts Polygons

The census tracts polygons are stored in a so called "shapefile collection" with the filename prefix **`gz_2010_24_140_00_500k`**. A shapefile collection consists of a number of different file types with the same filename prefix. These files contain geometric location and features as well attributes to these features. The files need to be stored in the same directory. This filing format was developed and is maintained by the Environmental Systems Research Institute (ESRI) (ESRI, 2020).

Four of these file types are required when performing spatial analysis:

- **`.shp`**, the shapefile itself, contains the features' geometry,
- **`.shx`**, contains the index of the feature geometry,
- **`.dbf`**, a table in **`dBASE`** that contains the attributes of the features, and
- **`.prj`**, a text file that contains the coordinate reference system information (**`CRS`**) of the features.

The function **`st_read()`** from the package **`sf`** imports shapefiles into **`R`**. While only the shapefile proper (**`.shp`**) is called, the function needs the other 3 files to properly import the geometric features and their attributes.

```
MD_CensusTracts_2010 <- sf::st_read("data/gz_2010_24_140_00_500k.shp")

MD_CensusTracts_2010
```

```
## Simple feature collection with 1403 features and 7 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -79.48765 ymin: 37.91172 xmax: -75.04894 ymax: 39.72304
## Geodetic CRS:  NAD83
## First 10 features:
##                  GEO_ID STATE COUNTY  TRACT    NAME  LSAD CENSUSAREA
## 1  1400000US24015031400    24    015 031400     314 Tract     16.039
## 2  1400000US24017850101    24    017 850101 8501.01 Tract      9.718
## 3  1400000US24017850706    24    017 850706 8507.06 Tract      2.698
## 4  1400000US24017850710    24    017 850710 8507.10 Tract      1.401
## 5  1400000US24017850901    24    017 850901 8509.01 Tract      2.082
## 6  1400000US24017851500    24    017 851500    8515 Tract      5.010
## 7  1400000US24017990000    24    017 990000    9900 Tract      0.000
## 8  1400000US24019970702    24    019 970702 9707.02 Tract     43.137
## 9  1400000US24019970804    24    019 970804 9708.04 Tract     87.808
## 10 1400000US24019970900    24    019 970900    9709 Tract    211.650
##                          geometry
## 1  MULTIPOLYGON (((-76.15435 3...
## 2  MULTIPOLYGON (((-77.09875 3...
## 3  MULTIPOLYGON (((-76.94249 3...
## 4  MULTIPOLYGON (((-76.96267 3...
## 5  MULTIPOLYGON (((-76.90672 3...
## 6  MULTIPOLYGON (((-76.96267 3...
## 7  MULTIPOLYGON (((-77.08633 3...
## 8  MULTIPOLYGON (((-76.25027 3...
## 9  MULTIPOLYGON (((-76.16439 3...
## 10 MULTIPOLYGON (((-76.04837 3...
```

The first line of the code reads in the data. You may get a warning similar to the following:

```
## Warning: replacing previous import 'vctrs::data_frame' by 'tibble::data_frame'
## when loading 'dplyr'
```

The warning can be ignored. It basically tells you that if the package **`dplyr`** would be loaded, "traditional" data frames may be replaced by **`tibble`** data frames. This will not affect our analysis.

The second line of code (**`MD_CensusTracts_2010`**) prints out the first 10 entries of the read in shapefile. It tells us that **`MD_CensusTracts_2010`** is a "simple feature" object (**`sf`**) with 1,403 and 7 fields (plus a geometry column). The geometric features are multi polygons. The coordinate reference system (**`CRS`**) is **`NAD83`**. **`NAD83`** stands for the North American Datum of 1983. The **`NAD83(2011)`** is the current geodetic system that is used for the continental U.S. (Section 2.3) (National Geodetic Survey, 2018).

Maryland Counties (Physical) Boundaries

The **`Maryland_Physical_Boundaries_-County_Boundaries(Detailed)`** shapefile collection contains polygons of the Maryland counties (physical) boundaries that take into consideration waterways (such as the tributaries of Chesapeake Bay).

Import the file with **`st_read()`**.

```
MD_counties_map <-
sf::st_read("data/Maryland_Physical_Boundaries_-_County_Boundaries_(Detailed).shp")

MD_counties_map
```

```
## Simple feature collection with 24 features and 7 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -8848486 ymin: 4564403 xmax: -8354439 ymax: 4825752
## Projected CRS: WGS 84 / Pseudo-Mercator
## First 10 features:
##    OBJECTID          COUNTY DISTRICT COUNTY_FIP COUNTYNUM CREATION_D LAST_UPDAT
## 1         1        Allegany        6          1         1 2010-01-28 2010-01-28
## 2         2   Anne Arundel        5          3         2 2006-04-18 2006-04-18
## 3         3       Baltimore        4          5         3 2006-10-09 2006-10-09
## 4         4 Baltimore City        0        510        24 2006-04-18 2009-11-16
## 5         5         Calvert        5          9         4 2010-01-28 2010-01-28
## 6         6        Caroline        2         11         5 2007-05-21 2008-07-30
## 7         7         Carroll        7         13         6 2008-06-16 2012-01-17
## 8         8           Cecil        2         15         7 2006-04-18 2008-08-20
## 9         9         Charles        5         17         8 2009-06-08 2009-06-08
## 10       10      Dorchester        1         19         9 2007-02-08 2007-02-22
##                          geometry
## 1  MULTIPOLYGON (((-8721085 48...
## 2  MULTIPOLYGON (((-8527741 47...
## 3  MULTIPOLYGON (((-8523507 48...
## 4  MULTIPOLYGON (((-8519244 47...
## 5  MULTIPOLYGON (((-8531762 46...
## 6  MULTIPOLYGON (((-8432189 47...
## 7  MULTIPOLYGON (((-8556981 47...
## 8  MULTIPOLYGON (((-8441053 48...
## 9  MULTIPOLYGON (((-8580309 46...
## 10 MULTIPOLYGON (((-8439760 46...
```

This dataset uses a projected **CRS** that is based on **WGS 84** using a pseudo-mercator projection. Pseudo-mercator projections are used by many web based mapping apps.

List of Vaccination Sites

MD_Covid19_VacSites_2021-04-04.csv is a comma separated file that lists COVID-19 vaccination sites in Maryland. The table is read into **R** with **read.table()** and assigned to the object **VaccineSites**. Its structure is explored with **str()**.

```
VaccineSites <- read.table(file = "data/MD_Covid19_VacSites_2021-04-04.csv",
                           sep = ",",
                           na.strings = c(""," ", "NA"),
                           header = TRUE,
                           quote = "\"",
                           fill = TRUE)

str(VaccineSites, list.len = 20, strict.width = "cut")
```

```
## 'data.frame':    556 obs. of  48 variables:
##  $ OBJECTID                : int  84 87 92 93 95 96 98 99 100 102 ...
##  $ facilityid              : chr  "Anna_Lumi_83" "Lanh_Lumi_86" "Balt_Grac_91"..
##  $ name                    : chr  "Luminis Health Anne Arundel Medical Center"..
##  $ fulladdr                : chr  "2001 Medical Parkway, Annapolis, MD 21401""..
##  $ Location                : chr  "(38.990512076102, -76.5341664410021)" "(38"..
##  $ X                       : num  -76.5 -76.9 -76.6 -77 -76.9 ...
##  $ Y                       : num  39 39 39.3 39.6 39.2 ...
##  $ municipality            : chr  "Annapolis" "Lanham" "Baltimore" "Westminst"..
##  $ CreationDate            : chr  "1970/01/01 00:00:00+00" "1970/01/01 00:00:"..
##  $ Creator                 : logi  NA NA NA NA NA NA ...
##  $ EditDate                : chr  "1970/01/01 00:00:00+00" "1970/01/01 00:00:"..
##  $ Editor                  : chr  NA NA NA NA ...
##  $ ActiveYesNo             : chr  "Yes" "Yes" "Yes" "Yes" ...
##  $ site_type               : chr  "Hospital" "Hospital" "Hospital" "Hospital" ..
##  $ appt_required           : logi  NA NA NA NA NA NA ...
##  $ operationalhours        : chr  "Mon - Fri 7 am - 7 pm" "Mon - Fri 7am - 7 "..
##  $ docorder_required       : logi  NA NA NA NA NA NA ...
##  $ costfree                : logi  NA NA NA NA NA NA ...
##  $ cost_outpocket          : logi  NA NA NA NA NA NA ...
##  $ drivethru               : logi  NA NA NA NA NA NA ...
##   [list output truncated]
```

This data frame contains 556 observations and 48 variables. The data frame as a variable called **`Location`** that has GPS coordinates presented as **`(longitude, latitude)`**. The data frame also has *xy* coordinates (variables **`X`** and **`Y`**). However for the purpose of practice, we will not use them.

4.2 Data Manipulation (Wrangling)

Once the data are imported, in most cases the data needs to be manipulated to continue with the analysis.

Our aim is to map “vaccination deserts” in Maryland and in Baltimore City. We have 3 datasets, namely 2 (non spatial) data frames, **`FoodAccess2015`** and **`VaccineSites`**, and a spatial object, **`MD_CensusTracts`**. The two data frames contain geographic and demographic information (location of vaccination sites in Maryland, demographic profiles of the census tracts within the U.S.), and the spatial (simple feature) object the geographic boundaries of census tracts in Maryland.

We need to:

1. Extract the data for Maryland and Baltimore City, where possible,
2. Combine demographic profiles and geographic boundaries, and
3. Convert the data frame with location information on vaccination sites into a spatial (simple feature) object.

This section describes how to extract data from a data frame (spatial and non-spatial) as well as merging data frames. Conversion of data frames into spatial objects, manipulation of spatial objects, and mapping is discussed in the next section.

Subsetting (Filtering) Data Frames

We are interested in data within Maryland and Baltimore City. To extract the data we use the function **`subset()`** (part of the core **`R`** installation).

Food Access Data

The data frame contains data for the entire US. List the first five variables with **`str()`**.

```r
str(FoodAccess2015, list.len = 5, strict.width = "cut")
```

```
## 'data.frame':    72864 obs. of  147 variables:
##  $ CensusTract         : chr  "01001020100" "01001020200" "01001020300" "0100"..
##  $ State               : chr  "Alabama" "Alabama" "Alabama" "Alabama" ...
##  $ County              : chr  "Autauga" "Autauga" "Autauga" "Autauga" ...
##  $ Urban               : num  1 1 1 1 1 1 1 0 0 0 ...
##  $ POP2010             : num  1912 2170 3373 4386 10766 ...
##   [list output truncated]
```

The output shows that the data frame contains the variable **`State`** and **`County`**. The following code extracts only the rows for the state of Maryland, assigns the new data frame to the object **`FoodAccess2015_MD`**, and shows the structure of the first 20 variables of the new object.

```r
FoodAccess2015_MD <- subset(FoodAccess2015, State == "Maryland")

str(FoodAccess2015_MD, list.len = 20, strict.width = "cut")
```

```
## 'data.frame':    1399 obs. of  147 variables:
##  $ CensusTract          : chr  "24001000100" "24001000200" "24001000300" "2400"..
##  $ State                : chr  "Maryland" "Maryland" "Maryland" "Maryland" ...
##  $ County               : chr  "Allegany" "Allegany" "Allegany" "Allegany" ...
##  $ Urban                : num  0 0 0 1 1 1 1 1 1 1 ...
##  $ POP2010              : num  3718 4564 2780 3022 2734 ...
##  $ OHU2010              : num  1523 1284 1133 1350 1044 ...
##  $ GroupQuartersFlag    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ NUMGQTRS             : num  39 1517 155 14 345 ...
##  $ PCTGQTRS             : num  0.01049 0.33238 0.05576 0.00463 0.12619 ...
##  $ LILATracts_1And10    : num  1 0 0 1 1 0 0 0 0 0 ...
##  $ LILATracts_halfAnd10: num  1 0 0 1 1 1 1 1 0 0 ...
##  $ LILATracts_1And20    : num  0 0 0 1 1 0 0 0 0 0 ...
##  $ LILATracts_Vehicle   : num  0 0 0 0 0 0 1 1 0 0 ...
##  $ HUNVFlag             : num  0 0 0 0 0 0 1 1 0 0 ...
##  $ LowIncomeTracts      : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ PovertyRate          : num  6.6 13.7 20.2 12.8 56 ...
##  $ MedianFamilyIncome   : num  56875 60943 42727 48831 34519 ...
##  $ LA1and10             : num  1 0 0 1 1 0 0 0 0 0 ...
##  $ LAhalfand10          : num  1 0 0 1 1 1 1 1 0 0 ...
##  $ LA1and20             : num  0 0 0 1 1 0 0 0 0 0 ...
##   [list output truncated]
```

To verify that the new object only contains Maryland, call **`unique()`**:

```
unique(FoodAccess2015_MD$State)
```

```
## [1] "Maryland"
```

While it is possible to continue to work with the entire **`FoodAccess2015`** data frame, removing unwanted variables (columns) reduces the size of the data frame.

Based on the documentation `https://www.ers.usda.gov/data-products/food-access-research-atlas/documentation/`, we are interested in:

- Urban/Rural designation,
- County,
- Total population,
- Low Income, and
- Vehicles access.

To identify variable names in the **`FoodAccess2015`** data frame we import the **`Variable Lookup`** sheet from the **`FoodAccess2015.xlsx`** Excel workbook (sheet 2, see above), and show the first six entries (rows) with the **`head()`** function.

```
FoodAccessVar <- openxlsx::read.xlsx("data/FoodAccess2015.xlsx",
                                      sheet = "Variable Lookup")

head(FoodAccessVar, 6)
```

```
##         Field                LongName
## 1 CensusTract            Census tract
## 2       State                   State
## 3      County                  County
## 4       Urban             Urban tract
## 5     POP2010 Population, tract total
## 6     OHU2010    Housing units, total
##                                Description
## 1                      Census tract number
## 2                               State name
## 3                              County name
## 4                     Flag for urban tract
## 5           Population count from 2010 census
## 6 Occupied housing unit count from 2010 census
```

To identify the variable name for low income, we can query the **`FoodAccessVar`** data frame using a regular expression (**`regex`**). Regular expressions are strings of text that help to find text pattern. The regular expression **`"[Ll]ow[[:space:]][Ii]ncome"`** for example will match the character strings **`Low income`**, **`low income`**, **`Low Income`**, and **`low Income`**.

The function **`grepl()`** is a function that will check if the content of an element of a vector (such the cell of a table) matches a regular expression. It will return a logical vector (**`TRUE/FALSE`**). Therefore we can use **`grepl()`** to extract rows from a data frame that contain the regular expression.

```
FoodAccessVar_low_income <- subset(FoodAccessVar,
                                   grepl("[Ll]ow[[:space:]][Ii]ncome",
                                   FoodAccessVar$LongName) == TRUE)

FoodAccessVar_low_income[,1:2]
```

```
##                   Field
## 10    LILATracts_1And10
## 11 LILATracts_halfAnd10
## 12    LILATracts_1And20
## 13   LILATracts_Vehicle
## 15      LowIncomeTracts
##                                             LongName
## 10       Low income and low access tract measured at 1 mile for urban areas and 10 miles
           for rural areas
## 11      Low income and low access tract measured at 1/2 mile for urban areas and 10 miles
```

```
           for rural areas
## 12        Low income and low access tract measured at 1 mile for urban areas and 20 miles
             for rural areas
## 13      Low income and low access tract using vehicle access or low income and low access
           tract measured at 20 miles
## 15                                          Low income tract
```

The first line of the code extracts the rows of the **FoodAccessVar** data frame that have regular expression in the variable **LongName**. The filtered data frame is assigned to the object **FoodAccessVar_low_income**.

The second line displays the first two columns of the new data frame/object.

We repeat the code above for vehicle access, and extract rows that contain the expression **[Vv]ehicle [Aa]ccess]**. The filtered data frame is assigned to the object **FoodAccessVar_vehicle_access**.

```
FoodAccessVar_vehicle_access <- subset(FoodAccessVar,
                                       grepl("[Vv]ehicle[[:space:]][Aa]ccess",
                                       FoodAccessVar$LongName) == TRUE)

FoodAccessVar_vehicle_access[,1:2]
```

```
##                   Field
## 13  LILATracts_Vehicle
## 14             HUNVFlag
## 25  LATractsVehicle_20
## 54           lahunvhalf
## 55      lahunvhalfshare
## 80              lahunv1
## 81         lahunv1share
## 106            lahunv10
## 107       lahunv10share
## 132            lahunv20
## 133       lahunv20share
##                               LongName
## 13      Low income and low access tract using vehicle access or low income and low access
        tract measured at 20 miles
## 14                                       Vehicle access, tract with low vehicle access
## 25                  Low access tract using vehicle access and at 20 miles  in rural areas
## 54               Vehicle access, housing units without and low access at 1/2 mile, number
## 55                Vehicle access, housing units without and low access at 1/2 mile, share
## 80                 Vehicle access, housing units without and low access at 1 mile, number
## 81                  Vehicle access, housing units without and low access at 1 mile, share
## 106              Vehicle access, housing units without and low access at 10 miles, number
## 107               Vehicle access, housing units without and low access at 10 miles, share
## 132              Vehicle access, housing units without and low access at 20 miles, number
## 133               Vehicle access, housing units without and low access at 20 miles, share
```

Based on the above outputs, it looks like that the variables **`HUNVFlag`** and **`LowIncomeTracts`** are of interest. The first identifies tracts with low vehicle access, the second low-income tracts.

We therefore subset the **`FoodAccess2015_MD`** data frame to only keep the **`CensusTract`**, **`County`**, **`Urban`**, **`POP2010`**, **`LowIncomeTracts`**, and **`HUNVFlag`** variables (columns).

```
FoodAccess2015_MD <- subset(FoodAccess2015_MD, select = c(CensusTract, County, Urban,
                                                          POP2010, LowIncomeTracts,
                                                          HUNVFlag))

str(FoodAccess2015_MD, strict.width = "cut")
```

```
## 'data.frame':    1399 obs. of  6 variables:
##  $ CensusTract    : chr  "24001000100" "24001000200" "24001000300" "240010004"..
##  $ County         : chr  "Allegany" "Allegany" "Allegany" "Allegany" ...
##  $ Urban          : num  0 0 0 1 1 1 1 1 1 1 ...
##  $ POP2010        : num  3718 4564 2780 3022 2734 ...
##  $ LowIncomeTracts: num  1 1 1 1 1 1 1 1 1 1 ...
##  $ HUNVFlag       : num  0 0 0 0 0 0 1 1 0 0 ...
```

Let's see if the data set contains suspicious data. In particular, we should check whether the data for the census tracts are complete and make sense.

First, let's verify that there are no missing values with the **`colSums(is.na())`** nested function.

```
colSums(is.na(FoodAccess2015_MD))
```

```
##     CensusTract          County           Urban         POP2010 LowIncomeTracts
##               0               0               0               0               0
##        HUNVFlag
##               0
```

The output shows that there are no missing values.

The variable **`POP2010`** reports the number of residents in a census tract. If reported correctly, all entries should be greater than 0. A census tract with no residents would just not make any sense. The population should range between 1,800 and 8,000 (U.S. Census Bureau, 2019). To check whether there are entries without residents, we first search for entries without residents and then count them with **`nrow()`**. **`nrow()`** is a function that shows you how many rows are in a data frame.

```
bad_census_tracts <- subset(FoodAccess2015_MD, POP2010 == 0)

nrow(bad_census_tracts)
```

```
## [1] 9
```

It looks like there are nine entries without residents. Let's see who they are by calling **`bad_census_tracts`**. To shorten the output, we only display the census tract, the county, (columns/-variables 1 and 2) and **`POP2010`** (variables 4).

```
bad_census_tracts[,c(1:2, 4)]
```

```
##       CensusTract       County POP2010
## 30065 24005980000    Baltimore       0
## 30066 24005980100    Baltimore       0
## 30067 24005980200    Baltimore       0
## 30191 24019990000   Dorchester       0
## 30821 24035990100 Queen Anne's       0
## 30839 24037990000   St. Mary's       0
## 30847 24039990100     Somerset       0
## 30924 24047980000    Worcester       0
## 30925 24047990000    Worcester       0
```

We remove these tracts by keeping all the entries that have a **`POP2010`** value that is not 0 (hence the **`!=`** operator).

```
FoodAccess2015_MD_good <- subset(FoodAccess2015_MD, POP2010 != 0)
```

Calling **`nrow(subset(FoodAccess2015_MD_good, POP2020 == 0))`** should return **`[1] 0`**

```
nrow(subset(FoodAccess2015_MD_good, POP2010 == 0))
```

```
## [1] 0
```

And it does.

We are also interested in identifying possible vaccination deserts in Baltimore City, and could subset the Maryland Food Access data set for Baltimore City. However, at this point the Maryland data set does not contain any geographic information. The subsection “Merging Data Frames” shows how to add geographic information to the data set. Once that is done we will create a subset for Baltimore City.

Census Tracts

The census tract data is for the entire state of Maryland and contains the census tract boundaries (as polygons). For our purposes, we will combine this data set with the Food Access data set.

Let’s check if all entries do have geographic information (i.e., a geometry entry).

```
colSums(is.na(MD_CensusTracts_2010))
```

```
##     GEO_ID      STATE     COUNTY      TRACT       NAME       LSAD CENSUSAREA 
##          0          0          0          0          0          0          0 
##   geometry 
##          0
```

There are no missing values.

Vaccination Sites

The **`VaccineSites`** data frame lists COVID-19 vaccination sites in Maryland. To list all the variables (columns) we can use the function **`colnames()`**.

```
colnames(VaccineSites)
```

```
##  [1] "OBJECTID"                "facilityid"             
##  [3] "name"                    "fulladdr"               
##  [5] "Location"                "X"                      
##  [7] "Y"                       "municipality"           
##  [9] "CreationDate"            "Creator"                
## [11] "EditDate"                "Editor"                 
## [13] "ActiveYesNo"             "site_type"              
## [15] "appt_required"           "operationalhours"       
## [17] "docorder_required"       "costfree"               
## [19] "cost_outpocket"          "drivethru"              
## [21] "pedaccess"               "transitAccess"          
## [23] "test_antigen"            "test_antibody"          
## [25] "other_notes"             "insurance_accepted"     
## [27] "medicaid"                "cost_other"             
## [29] "cost_notes"              "test_rapid"             
## [31] "schedule_url"            "online_scheduling"      
## [33] "scheduling_contact"      "scheduling_contact_phone"
## [35] "scheduling_contact_email" "test_pcr"
## [37] "website_url"              "X_coord"
## [39] "Y_coord"                  "test_pediatric"
## [41] "multi_language"           "test_pediatric_notes"
## [43] "created_user"             "created_date"
## [45] "last_edited_user"         "last_edited_date"
## [47] "County"                   "PreRegistrationURL"
```

The data frame has 48 variables including a variable with GPS information (**`Location`**). Let's check if all entries have GPS information.

```
sum(is.na(VaccineSites$Location))
```

```
## [1] 0
```

All entries have GPS coordinates.

Remove unwanted variables

The **`VaccineSites`** data frame contains information that is not relevant for our analysis. We are interested in the GPS coordinates (**`Location`**), and the name of the facility (**`name`**). Furthermore, information on whether a physician referral is required (**`docorder_required`**), whether it is cost-free (**`costfree`**), and whether the facility is accessible by foot (**`pedaccess`**) would be interesting in the context of affordable service.

The following code subsets the **`VaccineSites`** data frame for the above variables, and checks whether data for all variables is available.

```
VaccineSites_mod <- subset(VaccineSites, select = c("name", "Location", "pedaccess",
                                                    "docorder_required", "costfree"))

str(VaccineSites_mod, strict.width = "cut")
```

```
## 'data.frame':    556 obs. of  5 variables:
##  $ name             : chr  "Luminis Health Anne Arundel Medical Center" "Lumi"..
##  $ Location         : chr  "(38.990512076102, -76.5341664410021)" "(38.982597"..
##  $ pedaccess        : logi  NA NA NA NA NA NA ...
##  $ docorder_required: logi  NA NA NA NA NA NA ...
##  $ costfree         : logi  NA NA NA NA NA NA ...
```

```
colSums(is.na(VaccineSites_mod))
```

```
##              name          Location         pedaccess docorder_required
##                 0                 0               556               556
##          costfree
##               556
```

It turns out that we only have information on the name and coordinates for all facilities. Therefore, we drop the other variables.

```
VaccineSites_mod <- subset(VaccineSites, select = c("name", "Location"))
```

```
str(VaccineSites_mod, strict.width = "cut")
```

```
## 'data.frame':    556 obs. of  2 variables:
##  $ name    : chr  "Luminis Health Anne Arundel Medical Center" "Luminis Healt"..
##  $ Location: chr  "(38.990512076102, -76.5341664410021)" "(38.9825972406456, "..
```

```
colSums(is.na(VaccineSites_mod))
```

```
##     name Location
##        0        0
```

As so often with **R**, there are alternative ways to accomplish the same task. **R** has functions that remove entries if they have missing values. For example the function **na.omit()** removes all entries that have missing values. Another function, **complete.cases()** removes either all variables or all rows that have some missing values. We only want to drop variables that only have missing values. We can do so with the following code. It will remove all variables that only contain missing values from the subsetted data frame **VaccineSites_mod2**.

```
VaccineSites_mod2 <- subset(VaccineSites, select = c("name",
                                                     "Location",
                                                     "pedaccess",
                                                     "docorder_required",
                                                     "costfree"))

VaccineSites_mod2 <- VaccineSites_mod2[, colSums(is.na(VaccineSites_mod2))
                                       != nrow(VaccineSites_mod2)]

colSums(is.na(VaccineSites_mod2))
```

```
##     name Location
##        0        0
```

```
str(VaccineSites_mod2, strict.width = "cut")
```

```
## 'data.frame':    556 obs. of  2 variables:
##  $ name    : chr  "Luminis Health Anne Arundel Medical Center" "Luminis Healt"..
##  $ Location: chr  "(38.990512076102, -76.5341664410021)" "(38.9825972406456, "..
```

Merging Data Frames

MD_CensusTracts_2010 has the geographic boundaries of the census tracts in Maryland. To add the attributes/properties to the census tracts that will allow us to map census tracts that have limited access to vaccination sites, and/or are defined as low income, and/or have limited access to a vehicle, etc. the **FoodAccess2015_MD** data frame is merged with **MD_CensusTracts_2010**.

The function **merge()** allows to combine data frames based on a common variable. Here the common variable is the census tract. Naturally, the requirement for this function is that both data frames have one variable in common.

Prepare data frames for merging

Both data frames/objects (**FoodAccess2015_MD_good**, **MD_CensusTracts_2010**) have census tract information. However, the data frames encode the census tracts differently, and have different variable names. Furthermore, **FoodAccess2015_MB_good** has 13 fewer observations (in part because we removed a few observations).

```
# Structure of Food Access
str(FoodAccess2015_MD_good, list.len = 20, strict.width = "cut")
```

```
## 'data.frame':    1390 obs. of  6 variables:
##  $ CensusTract    : chr  "24001000100" "24001000200" "24001000300" "240010004"..
##  $ County         : chr  "Allegany" "Allegany" "Allegany" "Allegany" ...
##  $ Urban          : num  0 0 0 1 1 1 1 1 1 1 ...
##  $ POP2010        : num  3718 4564 2780 3022 2734 ...
##  $ LowIncomeTracts: num  1 1 1 1 1 1 1 1 1 1 ...
##  $ HUNVFlag       : num  0 0 0 0 0 0 1 1 0 0 ...
```

```
# Struture of MD_CensusTracts
str(MD_CensusTracts_2010, list.len = 5, strict.width = "cut")
```

```
## Classes 'sf' and 'data.frame':   1403 obs. of  8 variables:
##  $ GEO_ID    : chr  "1400000US24015031400" "1400000US24017850101" "1400000US2"..
##  $ STATE     : chr  "24" "24" "24" "24" ...
##  $ COUNTY    : chr  "015" "017" "017" "017" ...
##  $ TRACT     : chr  "031400" "850101" "850706" "850710" ...
##  $ NAME      : chr  "314" "8501.01" "8507.06" "8507.10" ...
##   [list output truncated]
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA N..
##   ..- attr(*, "names")= chr [1:7] "GEO_ID" "STATE" "COUNTY" "TRACT" ...
```

The tract ID of the **FoodAccess2015_MD** data frame contains the state ID (**24**), a three-digit county ID (f.e. **001** for Allegany county), and a six-digit identifier of the census tract which unique within each county. In contrast, **MD_CensusTracts_2010** shows only the six-digit identifier of the census tract in each county. The two-digit state ID and the three-digit county ID are stored in a separate variable.

FoodAccess2015_MD	24001000100 24001000200 ...
MD_CensusTracts_2010	000100 000200 ...

Furthermore, the variable name is different (**CensusTract** vs. **TRACT**).

The function **paste()** allows to combine variables (columns). The following code merges the three variables (columns) **STATE**, **COUNTY** and **TRACT** of the **MD_CensusTracts_2010** data frame and puts the new ID into a new variable **(CensusTract**)

```
# merge columns
MD_CensusTracts_2010$CensusTract <- paste(MD_CensusTracts_2010$STATE,
                                          MD_CensusTracts_2010$COUNTY,
                                          MD_CensusTracts_2010$TRACT,
                                          sep = "")
```

The code below checks the class of the modified data frame, and confirms that the **MD_CensusTracts_2010** is still a (spatial) simple feature (**sf**) object.

```
class(MD_CensusTracts_2010)
```

```
## [1] "sf"         "data.frame"
```

Merge data frames

`MD_CensusTracts_2010` is a **`sf`** object and contains spatial data. When merging with non spatial data frames, the **`sf`** object that contains spatial data has to come first.

```
MD_CensusTracts_Map <- merge(MD_CensusTracts_2010, FoodAccess2015_MD_good,
                             by = "CensusTract", all.x = TRUE)

# dimension
dim(MD_CensusTracts_Map)
```

```
## [1] 1403   14
```

```
class(MD_CensusTracts_Map)
```

```
## [1] "sf"         "data.frame"
```

```
# structure (columns 1:13), 14 = geometry
str(MD_CensusTracts_Map[,c(1:13)], list.len = 13, strict.width = "cut")
```

```
## Classes 'sf' and 'data.frame':   1403 obs. of  14 variables:
##  $ CensusTract    : chr  "24001000100" "24001000200" "24001000300" "240010004"..
##  $ GEO_ID         : chr  "1400000US24001000100" "1400000US24001000200" "14000"..
##  $ STATE          : chr  "24" "24" "24" "24" ...
##  $ COUNTY         : chr  "001" "001" "001" "001" ...
##  $ TRACT          : chr  "000100" "000200" "000300" "000400" ...
##  $ NAME           : chr  "1" "2" "3" "4" ...
##  $ LSAD           : chr  "Tract" "Tract" "Tract" "Tract" ...
##  $ CENSUSAREA     : num  187.94 48.07 8.66 3.72 4.42 ...
##  $ County         : chr  "Allegany" "Allegany" "Allegany" "Allegany" ...
##  $ Urban          : num  0 0 0 1 1 1 1 1 1 1 ...
##  $ POP2010        : num  3718 4564 2780 3022 2734 ...
##  $ LowIncomeTracts: num  1 1 1 1 1 1 1 1 1 1 ...
##  $ HUNVFlag       : num  0 0 0 0 0 0 1 1 0 0 ...
##   [list output truncated]
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA N..
##   ..- attr(*, "names")= chr [1:13] "CensusTract" "GEO_ID" "STATE" "COUNTY" ...
```

The **`by = "CensusTract"`** argument of the **`merge()`** function merges the data frames based on the variable **`CensusTract`**. The argument **`all.x = TRUE`** will keep all entries of the first data frame and will add missing values if there is no match of the common variable (**`CensusTract`**) in the 2nd data frame. Entries that do have a matching **`CensusTract`** in the first data frame are excluded.

We should have at least 13 entries with missing data.

```
colSums(is.na(MD_CensusTracts_Map))
```

```
##      CensusTract           GEO_ID            STATE           COUNTY            TRACT
##                0                0                0                0                0
##             NAME             LSAD       CENSUSAREA           County            Urban
##                0                0                0               13               13
##          POP2010 LowIncomeTracts         HUNVFlag         geometry
##               13               13               13                0
```

Indeed, we have 13 entries with missing values. These census tracts will be mapped as having "no data."

Subset for Baltimore City

Last, we extract the data for Baltimore City from the **`MD_CensusTracts_Map`** data set. We can subset using the last three digits of the FIPS county code (stored in the variable **`COUNTY`**), which for Baltimore City is **`510`**.

```
BC_CensusTracts_Map <- subset(MD_CensusTracts_Map, COUNTY == "510")

str(BC_CensusTracts_Map, list.len = 13, strict.width = "cut")
```

```
## Classes 'sf' and 'data.frame':   200 obs. of  14 variables:
##  $ CensusTract    : chr  "24510010100" "24510010200" "24510010300" "245100104"..
##  $ GEO_ID         : chr  "1400000US24510010100" "1400000US24510010200" "14000"..
##  $ STATE          : chr  "24" "24" "24" "24" ...
##  $ COUNTY         : chr  "510" "510" "510" "510" ...
##  $ TRACT          : chr  "010100" "010200" "010300" "010400" ...
##  $ NAME           : chr  "101" "102" "103" "104" ...
##  $ LSAD           : chr  "Tract" "Tract" "Tract" "Tract" ...
##  $ CENSUSAREA     : num  0.152 0.137 0.26 0.144 0.06 0.067 0.076 0.256 0.165 0..
##  $ County         : chr  "Baltimore City" "Baltimore City" "Baltimore City" ""..
##  $ Urban          : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ POP2010        : num  3022 3009 2208 2870 1724 ...
##  $ LowIncomeTracts: num  0 0 0 0 0 0 0 0 1 1 ...
##  $ HUNVFlag       : num  0 0 0 0 0 0 0 0 1 0 ...
##   [list output truncated]
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA N..
##   ..- attr(*, "names")= chr [1:13] "CensusTract" "GEO_ID" "STATE" "COUNTY" ...
```

```
colSums(is.na(BC_CensusTracts_Map))
```

```
##     CensusTract          GEO_ID           STATE          COUNTY           TRACT
##               0               0               0               0               0
##            NAME            LSAD      CENSUSAREA          County           Urban
##               0               0               0               0               0
##         POP2010 LowIncomeTracts        HUNVFlag        geometry
##               0               0               0               0
```

Baltimore City has 200 entries, and there are no missing values.