Yield analysis maize part 1

Kathi

Contents

Introduction

In this project I analyse yield data from a variety of crop species from 1981 - 2016.

Iizumi, Toshichika (2019): Global dataset of historical yields v1.2 and v1.3 aligned version. PANGAEA, https://doi.org/10.1594/PANGAEA.909132, Supplement to: Iizumi, Toshichika; Sakai, T (2020): The global dataset of historical yields for major crops 1981–2016. Scientific Data, 7(1), https://doi.org/10.1038/s41597-020-0433-7 (published under CC-BY-4.0 License)

The data is stored in NetCDF-4 (Network Common Data Form, versin 4) format (.nc4). To read more about that file format follow this link: https://docs.unidata.ucar.edu/netcdf-c/current/

Helpful websites:
https://www.r-bloggers.com/2016/08/a-netcdf-4-in-r-cheatsheet/

https://stackoverflow.com/questions/21708488/get-country-and-continent-from-longitude-and-latitude-point-in-r https://towardsdatascience.com/the-correct-way-to-average-the-globe-92ceecd172b7

http://www.idlcoyote.com/map_tips/lonconvert.html

Setup

Installing and loading required packages

```
# Store package names, required for the analysis in a vector
packages <- c("tidyverse", "ncdf4",  "reshape2", "RColorBrewer", "raster", "sp", "rworldmap", "sf", "pra

# Install packages that are not yet installed
installed_packages <- packages %in% rownames(installed.packages())
if (any(installed_packages == FALSE)) {
  install.packages(packages[!installed_packages])
}

# Load packages
invisible(lapply(packages, library, character.only = TRUE))
```

Reading yield data

Load and investigate the first NetCDF file (single year example)

This section helps to understand the structure of the files. Understanding the structure makes it easier to automatize loading multiple files from this dataset. So in a first step the yield data from one year (= one .nc4 file) will be read into memory and the data from all years will be read in in the next section.

```
# Create a list of all .nc4 files in the working directory
# This will be used to open the first file for a first investigation of the file structure
# Later on this list is used to automatically open all .nc4 files and write the data into
# a dataframe
list_maize <- list.files(path = ".", pattern = ".nc4")

print(list_maize)
```

```
##  [1] "yield_1981.nc4" "yield_1982.nc4" "yield_1983.nc4" "yield_1984.nc4"
##  [5] "yield_1985.nc4" "yield_1986.nc4" "yield_1987.nc4" "yield_1988.nc4"
##  [9] "yield_1989.nc4" "yield_1990.nc4" "yield_1991.nc4" "yield_1992.nc4"
## [13] "yield_1993.nc4" "yield_1994.nc4" "yield_1995.nc4" "yield_1996.nc4"
## [17] "yield_1997.nc4" "yield_1998.nc4" "yield_1999.nc4" "yield_2000.nc4"
## [21] "yield_2001.nc4" "yield_2002.nc4" "yield_2003.nc4" "yield_2004.nc4"
## [25] "yield_2005.nc4" "yield_2006.nc4" "yield_2007.nc4" "yield_2008.nc4"
## [29] "yield_2009.nc4" "yield_2010.nc4" "yield_2011.nc4" "yield_2012.nc4"
## [33] "yield_2013.nc4" "yield_2014.nc4" "yield_2015.nc4" "yield_2016.nc4"
```

```
# open the first nc file
maize_1981 <- nc_open(paste(list_maize[1]))

# Get a description of what is in the nc file
print(maize_1981)
```

```
## File yield_1981.nc4 (NC_FORMAT_NETCDF4):
##
##      1 variables (excluding dimension variables):
```

```
##         float var[lon,lat]   (Chunking: [720,360])  (Compression: shuffle,level 1)
##             _FillValue: -9.99e+08
##
##      2 dimensions:
##         lon  Size:720
##             units: degrees_east
##             long_name: Longitude
##         lat  Size:360
##             units: degrees_north
##             long_name: Latitude
```

```
# Save the description of the nc file in .txt format
{
sink(paste0(str_sub(list_maize[1], 1, 10), ".txt"))
  # str_sub() extracts the indicated letters from the file name
  # (to remove the .nc4 file ending)
print(maize_1981)
sink()
}

# Get a list of the NetCDF's R attributes
print(attributes(maize_1981)$names)
```

```
##  [1] "filename"    "writable"    "id"          "error"       "safemode"
##  [6] "format"      "is_GMT"      "groups"      "fqgn2Rindex" "ndims"
## [11] "natts"       "dim"         "unlimdimid"  "nvars"       "var"
```

The file has one variable, representing yield (in t/ha), called "var". The file has 2 dimensions, "lon" (with 720 values) and "lat" (with 360 values). The nc4 file contains a data structure, which seems to phase out of usage and is more common in scientific context of climate data analysis.

```
# Read the dimensions and store them in memory
maize_lon <- ncvar_get(maize_1981, "lon")
maize_lat <- ncvar_get(maize_1981, "lat")

# Inspect the first/last entries of the longitude and latitude vector and its structure
head(maize_lon)
```

```
## [1] 0.25 0.75 1.25 1.75 2.25 2.75
```

```
tail(maize_lon)
```

```
## [1] 357.25 357.75 358.25 358.75 359.25 359.75
```

```
str(maize_lon)
```

```
##  num [1:720(1d)] 0.25 0.75 1.25 1.75 2.25 2.75 3.25 3.75 4.25 4.75 ...
```

```
head(maize_lat)
```

```
## [1] -89.75 -89.25 -88.75 -88.25 -87.75 -87.25
```

```
tail(maize_lat)
```

```
## [1] 87.25 87.75 88.25 88.75 89.25 89.75
```

```
str(maize_lat)
```

```
##  num [1:360(1d)] -89.8 -89.2 -88.8 -88.2 -87.8 ...
```

The longitude vector contains values from 0.25 - 359.75 (in 0.5 degree steps, 720 entries). This format (0 - 360) will be changed later to the more standard -180 - 180 format! The latitude vector contains values from -89.75 - 89.75 (in 0.5 degree steps, 360 entries).

```
# Read in the data from the variable, in this data set called "var".
# This variable represents the amount of yield in t/ha
# store the data in a 2-dimensional array
maize_yield_1981_array <- ncvar_get(maize_1981, "var")

# Show the dimensions of the array
dim(maize_yield_1981_array)
```

```
## [1] 720 360
```

The yield data consists of an array with 720 x 360 (=259200) elements. The longitude values (720) are in the rows and the latitude values (360) are in the columns. In this wide format, the data is not tidy and standard data transformation operations are cumbersome to apply. Hence, the structure will be altered for data analytical purposes.

```
# Show the fillvalue
# The fillvalue is a variable that contains a value, that is used for missing data
fillvalue <- ncatt_get(maize_1981, "var", "_FillValue")
print(fillvalue)
```

```
## $hasatt
## [1] TRUE
##
## $value
## [1] -9.99e+08
```

```
# show the first five rows and columns of the array
print(maize_yield_1981_array[1:5, 1:5])
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   NA   NA   NA   NA   NA
## [2,]   NA   NA   NA   NA   NA
## [3,]   NA   NA   NA   NA   NA
## [4,]   NA   NA   NA   NA   NA
## [5,]   NA   NA   NA   NA   NA
```

The fillvalue used in this dataset is -9.99e+08. However, a closer inspection of the data in the array shows that this fill value is not used, instead the common NA is displayed for missing values.

Data tidying and cleaning

```
# Replace the FillValue (-9.99e+08) with the R-standard NA
  # But the Fillvalue seems not to have been used, so this step is not required
# maize_yield_1981_array[maize_yield_1981_array == fillvalue$value] <- NA
```

```
# Show a short peak on a random selection of the data
maize_yield_1981_array[1:3, 193:196]
```

```
##          [,1]     [,2]      [,3]      [,4]
## [1,]       NA 1.905901 1.4954479 2.0380375
## [2,] 1.018452 1.325993 1.3603835 1.4654421
## [3,]       NA 1.090900 0.9970124 0.9462057
```

```
# Change the dimension names of the data array to "lon" and "lat"
# and the row and column names to the latitude and longitude values
dimnames(maize_yield_1981_array) <- list(lon = maize_lon, lat = maize_lat)

# Check the same selection again
maize_yield_1981_array[1:3, 193:196]
```

```
##       lat
## lon        6.25     6.75      7.25      7.75
##   0.25       NA 1.905901 1.4954479 2.0380375
##   0.75 1.018452 1.325993 1.3603835 1.4654421
##   1.25       NA 1.090900 0.9970124 0.9462057
```

```
# Close the netCDF file, as all relevant data is read into R
nc_close(maize_1981)
```

```
# Change the data array from wide to long format
maize_yield_1981_long <- melt(maize_yield_1981_array, value.name = "yield" )

# show the head of the dataframe, ignoring NA values
head(na.omit(maize_yield_1981_long))
```

```
##          lon    lat    yield
## 79057 288.25 -35.25 4.127699
## 79488 143.75 -34.75 2.459394
## 79489 144.25 -34.75 3.058407
## 80207 143.25 -34.25 1.232776
## 80212 145.75 -34.25 3.058407
## 80924 141.75 -33.75 4.376572
```

```
# show the structure of the dataframe
str(maize_yield_1981_long)
```

```
## 'data.frame':    259200 obs. of  3 variables:
##  $ lon  : num  0.25 0.75 1.25 1.75 2.25 2.75 3.25 3.75 4.25 4.75 ...
##  $ lat  : num  -89.8 -89.8 -89.8 -89.8 -89.8 ...
##  $ yield: num  NA NA NA NA NA NA NA NA NA NA ...
```

As expected, the change from wide to long format results in a dataframe with 259200 observations.

Load multiple NetCDF files for analysis (data from year 1981 - 2016)

With the above-gathered information about the structure of the .nc4 files it is now possible to define a function that automatically loads the available files and stores the data in long format for further analysis.

```
# Define the function to load multiple .nc4 files
# This function uses as input a list of .nc4 file names that are supposed to be written
# into one dataframe with the variables lon, lat, yield and year.
# In this case each file contains worldwide maize yield data of one year,
# from 1981 to 2016.

process_maize_yield <- function(list_maize){
  # Create an empty dataframe called "maize_yield"
  maize_yield = data.frame()

  # iterate through the nc files
  for (i in 1:length(list_maize)) {
        # open a connection to the nc file
        maize_yield_tmp <- nc_open(list_maize[i])

        # store values from variables and attributes
        maize_yield_mtx <- ncvar_get(maize_yield_tmp,
                                     attributes(maize_yield_tmp$var)$names[1])
        maize_lon <- ncvar_get(maize_yield_tmp,
                               attributes(maize_yield_tmp$dim)$names[1])
        maize_lat <- ncvar_get(maize_yield_tmp,
                               attributes(maize_yield_tmp$dim)$names[2])

        # Create a variable, based on the year of the data collected,
        # extracted from the file name
        maize_year <- str_sub(list_maize[i], 7, 10)

        # close the connection since we're finished
        nc_close(maize_yield_tmp)

        # set the dimension names and values of your matrix to the appropriate latitude
        # and longitude values
        dimnames(maize_yield_mtx) <- list(lon = maize_lon, lat = maize_lat)


        tmp_maize_yield_df <- maize_yield_mtx %>%
          # Store the data in long format
          melt(value.name = "yield") %>%
          # Add a variable, based on the year of the data collected
          mutate(year = maize_year)

        # for debugging
        #print(names(maize_yield))
        #print(names(tmp_maize_yield_df))


        # set the name of my new variable and bind the new data to it
       # if (exists("maize_yield")) {
          maize_yield <- bind_rows(maize_yield, tmp_maize_yield_df)
        #}
       # else {
        #  maize_yield <- data.frame(tmp_maize_yield_df)
       # }
    }

    return(maize_yield)

    # return(tmp_maize_yield_df)
}

# Load the data from multiple .nc4 files into one dataframe
data <- process_maize_yield(list_maize)

# Show the structure of dataframe
str(data)
```

```
## 'data.frame':    9331200 obs. of  4 variables:
##  $ lon  : num  0.25 0.75 1.25 1.75 2.25 2.75 3.25 3.75 4.25 4.75 ...
##  $ lat  : num  -89.8 -89.8 -89.8 -89.8 -89.8 ...
##  $ yield: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ year : chr  "1981" "1981" "1981" "1981" ...
```

As exprected from 36 files with 259200 observations the resulting dataframe contains 9331200 observations.

Data transformations

Convert the longitude format from 0 - 360 to -180 - 180

The longitude information in this dataset is given in 0 - 360 format. To use the data in some of the below-described transformations it is required to change the format to the more standard -180 - 180 format.

```r
# Define the function to change the longitude format from 0 - 360 to -180 - 180
lon_to_180 <- function(lon){

  lon_180 <- ((lon + 180) %% 360) - 180

  # Output:
  lon_180
}

# Add a column (lon_180) that contains the longitude in -180 - 180 format to the dataframe
data <- data %>%
  mutate(lon_180 = lon_to_180(lon))

# Show the last entries of the lon_180 column
tail(data$lon_180)
```

```
## [1] -2.75 -2.25 -1.75 -1.25 -0.75 -0.25
```

Convert the longitude latitude information into country/continent names and add it to the dataframe

Based on a coordinate reference system (CRS) the observations of the dataset, defined by a longitude and latitude value can be mapped on to their position on the globe. With this information it is possible to assign country/continent names according to where each point is located.

The two functions coords2country and coords2continent could also be combined into one. For better control I will keep them separate.

```r
# The single argument to this function, points, is a data.frame in which:
#   - column 1 contains the longitude in degrees
#   - column 2 contains the latitude in degrees
coords2country = function(points){

  # Access a map stored in the rworldmap package
  countriesSP <- getMap(resolution='low')

  # set CRS (coordinate reference system) directly to that from rworldmap
  pointsSP = SpatialPoints(points, proj4string=CRS(proj4string(countriesSP)))

  # use 'over' to get indices of the Polygons object containing each point
  indices = over(pointsSP, countriesSP)
  # return the ADMIN names of each country
    # ADMIn are the country names filed in the rworldmap package
  indices$ADMIN
  #indices$ISO3 # returns the ISO3 code
  #indices$continent   # returns the continent (6 continent model)
  #indices$REGION   # returns the continent (7 continent model)
}

# Add a column called "country" to the data frame "data"
# as the function coords2country() uses a dataframe with 2 columns (lon and lat) as input use only the
data$country <- coords2country(dplyr::select(data, c("lon_180", "lat")))

# Show the first entries of the country column
head(data$country)
```

```
## [1] Antarctica Antarctica Antarctica Antarctica Antarctica Antarctica
## 244 Levels: Afghanistan Aland Albania Algeria American Samoa Andorra ... Zimbabwe
```

```r
# The single argument to this function, points, is a data.frame in which:
#   - column 1 contains the longitude in degrees
#   - column 2 contains the latitude in degrees
coords2continent = function(points){

  countriesSP <- getMap(resolution='low')

  # setting CRS directly to that from rworldmap
  pointsSP = SpatialPoints(points, proj4string=CRS(proj4string(countriesSP)))

  # use 'over' to get indices of the Polygons object containing each point
  indices = over(pointsSP, countriesSP)

  # return the ADMIN names of each country
  indices$REGION
  #indices$ISO3 # returns the ISO3 code
  #indices$continent   # returns the continent (6 continent model)
  #indices$REGION   # returns the continent (7 continent model)
}

# Add a column called "continent" to the data frame "data"
data$continent <- coords2continent(dplyr::select(data, c("lon_180", "lat")))

# Show the first entries of the continent column
head(data$continent)
```

```
## [1] Antarctica Antarctica Antarctica Antarctica Antarctica Antarctica
## 7 Levels: Africa Antarctica Asia Australia Europe ... South America
```

Calculate the area "around" each point, defined by longitude and latitude, of the dataframe

I assume that each point in the dataframe (definded by longitude and latitude) represents an area (square) around it. As the earth is not flat, the area of each square changes with its longitude. In a first step we need to calculate the earth radius in meters at each latitude.

```r
# Define the function (calc_earth_radius) to calculate earth_radius in meters
# corresponding to latitude
# This function takes latitude (in degree) as input
calc_earth_radius = function(lat){

  # define oblate spheroid from WGS84
  a <- 6378137 # semi-major axis
  b <- 6356752.3142 # semi-minor axis
  e2 = 1 - (b^2/a^2)

  # convert the input lat information from geodetic to geocentric
  lat_rad <- deg2rad(lat)
  lat_gc <- atan((1 - e2) * tan(lat_rad))

  # calculate the radius
  r <- ( (a * (1-e2)^0.5)  / (1 - (e2 * cos(lat_gc)^2))^0.5)

  # Define the output
  r
}


# add the earth_radius per data point to the data frame
data$earth_radius <- calc_earth_radius(data$lat)

# Show the first entries of the earth_radius column
head(data$earth_radius)
```

```
## [1] 6356753 6356753 6356753 6356753 6356753 6356753
```

```r
# Create a square "around" each long/lat point to calculate its area
# Define the length of long and lat as the difference between one data point and the next
# one (within the group) (lon_diff & lat_diff)
data <- data %>%
  # arrange the data frame by lat value in ascending orderto assure correct work of the
  # lead function
  arrange(lat, lon_180) %>%
  group_by(year, lon_180) %>%
  mutate(lat_diff = lead(lat) - lat) %>%  # add the lat_diff variable (in degree)
    # lead function returns the next point to the analysed data point
    # as the last point within each group has no corresponding next point it returns NA
    # as this point is at lat = 89.75, corresponding to the south pole, this value will
    # be dropped in further analysis, as no yield value is expected at this location
  group_by(lat, year) %>%
  # add the lon_diff variable (in degree)
  mutate(lon_diff = coalesce(lead(lon_180) - lon_180, lon_180 - lag(lon_180)))
    # lead function returns the the next point to the analysed data point
    # coalesce function returns the first non-missing value, which means:
    # it returns [lead(_180) - lon_180] for every data point of that group,
    # as the last point point has no corresponding  "next" point it will return
    # [lon_180 - lag(lon_180)] = "the last point - the penultimate point"
```

```r
# calculate the long and lat length (in m) and add it to the data frame
# and calculate the "area of each point"
```

```
data <- data %>%
  mutate(lon_m = deg2rad(lon_diff * earth_radius * cos(deg2rad(lat))),
         lat_m = deg2rad(lat_diff * earth_radius),
         area = lon_m *lat_m, # in m²
         area_ha = area * 0.0001) # in ha, required as the yield data is given in t/ha

# Show the first entries of the dataframe
head(data)
```

```
## # A tibble: 6 x 14
## # Groups:   lat, year [6]
##     lon   lat yield year  lon_180 country  conti~1 earth~2 lat_d~3 lon_d~4 lon_m
##   <dbl> <dbl> <dbl> <chr>   <dbl> <fct>    <fct>     <dbl>   <dbl>   <dbl> <dbl>
## 1  180. -89.8    NA 1981    -180. Antarct~ Antarc~  6.36e6     0.5     0.5  242.
## 2  180. -89.8    NA 1982    -180. Antarct~ Antarc~  6.36e6     0.5     0.5  242.
## 3  180. -89.8    NA 1983    -180. Antarct~ Antarc~  6.36e6     0.5     0.5  242.
## 4  180. -89.8    NA 1984    -180. Antarct~ Antarc~  6.36e6     0.5     0.5  242.
## 5  180. -89.8    NA 1985    -180. Antarct~ Antarc~  6.36e6     0.5     0.5  242.
## 6  180. -89.8    NA 1986    -180. Antarct~ Antarc~  6.36e6     0.5     0.5  242.
## # ... with 3 more variables: lat_m <dbl>, area <dbl>, area_ha <dbl>, and
## #   abbreviated variable names 1: continent, 2: earth_radius, 3: lat_diff,
## #   4: lon_diff
```

This dataframe (or parts of it) will be loaded into an SQL database later.

Reading demographic data (Source: world bank)

For the analysis that I am planning to do I am interested in demographic data, that I can use from The World Bank (published under CC BY 4.0 license)

References

The World Bank: Population, total: (1) United Nations Population Division. World Population Prospects: 2019 Revision. (2) Census reports and other statistical publications from national statistical offices, (3) Eurostat: Demographic Statistics, (4) United Nations Statistical Division. Population and Vital Statistics Reprot (various years), (5) U.S. Census Bureau: International Database, and (6) Secretariat of the Pacific Community: Statistics and Demography Programme. https://datacatalog.worldbank.org/public-licenses#cc-by

The World Bank: GDP (current US$): World Bank national accounts data, and OECD National Accounts data files. https://datacatalog.worldbank.org/public-licenses#cc-by

The World Bank: Exports of goods, services and primary income (BoP, current US$): International Monetary Fund, Balance of Payments Statistics Yearbook and data files. https://datacatalog.worldbank.org/public-licenses#cc-by

The World Bank: Adjusted net national income (current US$): World Bank staff estimates based on sources and methods in World Bank's "The Changing Wealth of Nations: Measuring Sustainable Development in the New Millennium" (2011). https://datacatalog.worldbank.org/public-licenses#cc-by

The World Bank: Imports of goods, services and primary income (BoP, current US$): International Monetary Fund, Balance of Payments Statistics Yearbook and data files. https://datacatalog.worldbank.org/public-licenses#cc-by

Import the csv files downloaded from worldbank.org

```r
# Load csv files
Population_wide <- read.csv("WorldBank_DevelopmentIndicators_Population.csv")
GDP_wide <- read.csv("WorldBank_DevelopmentIndicators_GDP.csv")
Income_wide <- read.csv("WorldBank_DevelopmentIndicators_Income.csv")
Export_wide <- read.csv("WorldBank_DevelopmentIndicators_Export.csv")
Import_wide <- read.csv("WorldBank_DevelopmentIndicators_Import.csv")
```

The data is stored in wide format, but to be compatible with the yield data it should be in long format.

Data transformation

```r
# Transform the data from wide to long format
Population <- pivot_longer(Population_wide, cols = starts_with("X"), names_to = "Year")
GDP <- pivot_longer(GDP_wide, cols = starts_with("X"), names_to = "Year")
Income <- pivot_longer(Income_wide, cols = starts_with("X"), names_to = "Year")
Export <- pivot_longer(Export_wide, cols = starts_with("X"), names_to = "Year")
Import <- pivot_longer(Import_wide, cols = starts_with("X"), names_to = "Year")

# rename the "value" variable in each data frame
Population <- rename(Population, population = value)
GDP <- rename(GDP, gdp = value)
Income <- rename(Income, income = value)
Export <- rename(Export, export = value)
Import <- rename(Import, import = value)

# Delete unnecessary columns
Population_clean <- subset(Population, select = -c(Series.Name, Series.Code, Country.Code))
GDP_clean <- subset(GDP, select = -c(Series.Name, Series.Code, Country.Code))
Income_clean <- subset(Income, select = -c(Series.Name, Series.Code, Country.Code))
Export_clean <- subset(Export, select = -c(Series.Name, Series.Code, Country.Code))
Import_clean <- subset(Import, select = -c(Series.Name, Series.Code, Country.Code))

# Merge the dataframes into one
demographic_data <- Population_clean %>%
  inner_join(GDP_clean, by = c("Country.Name", "Year")) %>%
  inner_join(Income_clean, by = c("Country.Name", "Year")) %>%
  inner_join(Export_clean, by = c("Country.Name", "Year")) %>%
  inner_join(Import_clean, by = c("Country.Name", "Year"))

# redefine the "Year" variable to only contain string 2 - 5, naming it "year"
demographic_data$year <- substring(demographic_data$Year, 2,5)


# Replace missing values that are labelled ".." with NA
demographic_data_clean <- demographic_data %>%
  subset(select = -c(Year)) %>%
  replace_with_na(replace = list(population = "..",
                                 gdp = "..",
                                 income = "..",
                                 export = "..",
                                 import = ".."))
```

```r
# Rename the Country.Name column in demographic_data_clean and remove duplicates
demographic_data_clean <- demographic_data_clean %>%
  rename(., country = Country.Name) %>%
  distinct() # remove duplicates!
```

Data cleaning

Check if Country Names are written similarly in both dataframes that will be uploaded to SQL

```r
# Extract levels attribute of country variable from both dataframes
data_country_names <- levels(data$country)
demographic_country_names <- levels(as.factor(demographic_data_clean$country))

# Extract overlapping and unique country names for both dataframes
overlap_country_names <-  intersect(data_country_names, demographic_country_names)
unique_data_country_names <- setdiff(data_country_names, demographic_country_names)
unique_demographic_country_names <- setdiff(demographic_country_names, data_country_names)
```

179 country names are found in both datasets, which represents 73 % of the country names in the yield dataset (244).

```r
# Print unique country names from both datasets
print(unique_data_country_names)
```

```
##  [1] "Aland"
##  [2] "Anguilla"
##  [3] "Antarctica"
##  [4] "Ashmore and Cartier Islands"
##  [5] "British Indian Ocean Territory"
##  [6] "Brunei"
##  [7] "Cape Verde"
##  [8] "Cook Islands"
##  [9] "Democratic Republic of the Congo"
## [10] "East Timor"
## [11] "Egypt"
## [12] "Falkland Islands"
## [13] "Federated States of Micronesia"
## [14] "French Guiana"
## [15] "French Southern and Antarctic Lands"
## [16] "Gambia"
## [17] "Gaza"
## [18] "Guernsey"
## [19] "Guinea Bissau"
## [20] "Heard Island and McDonald Islands"
## [21] "Hong Kong S.A.R."
## [22] "Indian Ocean Territories"
## [23] "Iran"
## [24] "Ivory Coast"
## [25] "Jersey"
## [26] "Kyrgyzstan"
## [27] "Laos"
## [28] "Macau S.A.R"
## [29] "Macedonia"
## [30] "Montserrat"
## [31] "Niue"
## [32] "Norfolk Island"
## [33] "North Korea"
## [34] "Northern Cyprus"
## [35] "Pitcairn Islands"
## [36] "Republic of Serbia"
## [37] "Republic of the Congo"
## [38] "Russia"
## [39] "Saint Barthelemy"
## [40] "Saint Helena"
## [41] "Saint Kitts and Nevis"
## [42] "Saint Lucia"
## [43] "Saint Martin"
## [44] "Saint Pierre and Miquelon"
## [45] "Saint Vincent and the Grenadines"
## [46] "Siachen Glacier"
## [47] "Sint Maarten"
## [48] "Slovakia"
## [49] "Somaliland"
## [50] "South Georgia and South Sandwich Islands"
## [51] "South Korea"
## [52] "Swaziland"
## [53] "Syria"
## [54] "Taiwan"
## [55] "The Bahamas"
## [56] "Turkey"
## [57] "United Republic of Tanzania"
## [58] "United States of America"
## [59] "United States Virgin Islands"
## [60] "Vatican"
## [61] "Venezuela"
## [62] "Wallis and Futuna"
## [63] "West Bank"
## [64] "Western Sahara"
## [65] "Yemen"
```

```
print(unique_demographic_country_names)
```

```
##  [1] ""
##  [2] "Africa Eastern and Southern"
##  [3] "Africa Western and Central"
##  [4] "Arab World"
##  [5] "Bahamas, The"
##  [6] "Brunei Darussalam"
##  [7] "Cabo Verde"
##  [8] "Caribbean small states"
##  [9] "Central Europe and the Baltics"
## [10] "Channel Islands"
## [11] "Congo, Dem. Rep."
## [12] "Congo, Rep."
## [13] "Cote d'Ivoire"
## [14] "Data from database: World Development Indicators"
## [15] "Early-demographic dividend"
## [16] "East Asia & Pacific"
## [17] "East Asia & Pacific (excluding high income)"
## [18] "East Asia & Pacific (IDA & IBRD countries)"
## [19] "Egypt, Arab Rep."
## [20] "Eswatini"
## [21] "Euro area"
## [22] "Europe & Central Asia"
## [23] "Europe & Central Asia (excluding high income)"
## [24] "Europe & Central Asia (IDA & IBRD countries)"
## [25] "European Union"
## [26] "Fragile and conflict affected situations"
## [27] "Gambia, The"
## [28] "Gibraltar"
## [29] "Guinea-Bissau"
## [30] "Heavily indebted poor countries (HIPC)"
## [31] "High income"
## [32] "Hong Kong SAR, China"
## [33] "IBRD only"
## [34] "IDA & IBRD total"
## [35] "IDA blend"
## [36] "IDA only"
## [37] "IDA total"
## [38] "Iran, Islamic Rep."
## [39] "Korea, Dem. People's Rep."
## [40] "Korea, Rep."
## [41] "Kyrgyz Republic"
## [42] "Lao PDR"
## [43] "Last Updated: 07/20/2022"
## [44] "Late-demographic dividend"
## [45] "Latin America & Caribbean"
## [46] "Latin America & Caribbean (excluding high income)"
## [47] "Latin America & the Caribbean (IDA & IBRD countries)"
## [48] "Least developed countries: UN classification"
## [49] "Low & middle income"
## [50] "Low income"
## [51] "Lower middle income"
## [52] "Macao SAR, China"
## [53] "Micronesia, Fed. Sts."
## [54] "Middle East & North Africa"
## [55] "Middle East & North Africa (excluding high income)"
## [56] "Middle East & North Africa (IDA & IBRD countries)"
## [57] "Middle income"
## [58] "North America"
## [59] "North Macedonia"
## [60] "Not classified"
## [61] "OECD members"
## [62] "Other small states"
## [63] "Pacific island small states"
## [64] "Post-demographic dividend"
## [65] "Pre-demographic dividend"
## [66] "Russian Federation"
## [67] "Serbia"
## [68] "Sint Maarten (Dutch part)"
## [69] "Slovak Republic"
## [70] "Small states"
## [71] "South Asia"
## [72] "South Asia (IDA & IBRD)"
## [73] "St. Kitts and Nevis"
## [74] "St. Lucia"
## [75] "St. Martin (French part)"
## [76] "St. Vincent and the Grenadines"
## [77] "Sub-Saharan Africa"
## [78] "Sub-Saharan Africa (excluding high income)"
## [79] "Sub-Saharan Africa (IDA & IBRD countries)"
## [80] "Syrian Arab Republic"
## [81] "Tanzania"
## [82] "Timor-Leste"
## [83] "Turkiye"
## [84] "United States"
## [85] "Upper middle income"
## [86] "Venezuela, RB"
## [87] "Virgin Islands (U.S.)"
## [88] "West Bank and Gaza"
## [89] "World"
## [90] "Yemen, Rep."
```

A comparison by hand easily identifies simple differences in the naming of the same country that will be changed in the "demographic_data_clean" dataframe to overlap with the names from "data".

```
# Relabel country names in the demographic dataframe
demographic_data_clean[demographic_data_clean == "Bahamas, The"] <- "The Bahamas"
demographic_data_clean[demographic_data_clean == "Brunei Darussalam"] <- "Brunei"
demographic_data_clean[demographic_data_clean == "Cabo Verde"] <- "Cape Verde"
demographic_data_clean[demographic_data_clean == "Congo, Dem. Rep."] <- "Democratic Republic of the Cong
demographic_data_clean[demographic_data_clean == "Congo, Rep."] <- "Republic of the Congo"
demographic_data_clean[demographic_data_clean == "Cote d'Ivoire"] <- "Ivory Coast"
demographic_data_clean[demographic_data_clean == "Egypt, Arab Rep."] <- "Egypt"
demographic_data_clean[demographic_data_clean == "Eswatini"] <- "Swaziland"
demographic_data_clean[demographic_data_clean == "Gambia, The"] <- "Gambia"
demographic_data_clean[demographic_data_clean == "Guinea-Bissau"] <- "Guinea Bissau"
demographic_data_clean[demographic_data_clean == "Hong Kong SAR, China"] <- "Hong Kong S.A.R."
demographic_data_clean[demographic_data_clean == "Iran, Islamic Rep."] <- "Iran"
demographic_data_clean[demographic_data_clean == "Korea, Dem. People's Rep."] <- "North Korea"
demographic_data_clean[demographic_data_clean == "Korea, Rep."] <- "South Korea"
demographic_data_clean[demographic_data_clean == "Kyrgyz Republic"] <- "Kyrgyzstan"
demographic_data_clean[demographic_data_clean == "Lao PDR"] <- "Laos"
demographic_data_clean[demographic_data_clean == "Macao SAR, China"] <- "Macau S.A.R"
demographic_data_clean[demographic_data_clean == "Micronesia, Fed. Sts."] <- "Federated States of Micro
demographic_data_clean[demographic_data_clean == "North Macedonia"] <- "Macedonia"
demographic_data_clean[demographic_data_clean == "Russian Federation"] <- "Russia"
demographic_data_clean[demographic_data_clean == "Serbia"] <- "Republic of Serbia"
demographic_data_clean[demographic_data_clean == "Sint Maarten (Dutch part)"] <- "Sint Maarten"
demographic_data_clean[demographic_data_clean == "Slovak Republic"] <- "Slovakia"
demographic_data_clean[demographic_data_clean == "St. Kitts and Nevis"] <- "Saint Kitts and Nevis"
demographic_data_clean[demographic_data_clean == "St. Lucia"] <- "Saint Lucia"
demographic_data_clean[demographic_data_clean == "St. Martin (French part)"] <- "Saint Martin"
demographic_data_clean[demographic_data_clean == "St. Vincent and the Grenadines"] <- "Saint Vincent an
demographic_data_clean[demographic_data_clean == "Syrian Arab Republic"] <- "Syria"
demographic_data_clean[demographic_data_clean == "Tanzania"] <- "United Republic of Tanzania"
demographic_data_clean[demographic_data_clean == "Timor-Leste"] <- "East Timor"
demographic_data_clean[demographic_data_clean == "Turkiye"] <- "Turkey"
```

```r
demographic_data_clean[demographic_data_clean == "United States"] <- "United States of America"
demographic_data_clean[demographic_data_clean == "Venezuela, RB"] <- "Venezuela"
demographic_data_clean[demographic_data_clean == "Virgin Islands (U.S.)"] <- "United States Virgin Islan
demographic_data_clean[demographic_data_clean == "Yemen, Rep."] <- "Yemen"
```

```r
# Recheck which countries are still unique
data_country_names <- levels(data$country)
demographic_country_names <- levels(as.factor(demographic_data_clean$country))

overlap_country_names <-  intersect(data_country_names, demographic_country_names)
unique_data_country_names <- setdiff(data_country_names, demographic_country_names)
unique_demographic_country_names <- setdiff(demographic_country_names, data_country_names)

print(unique_data_country_names)
```

```
##  [1] "Aland"
##  [2] "Anguilla"
##  [3] "Antarctica"
##  [4] "Ashmore and Cartier Islands"
##  [5] "British Indian Ocean Territory"
##  [6] "Cook Islands"
##  [7] "Falkland Islands"
##  [8] "French Guiana"
##  [9] "French Southern and Antarctic Lands"
## [10] "Gaza"
## [11] "Guernsey"
## [12] "Heard Island and McDonald Islands"
## [13] "Indian Ocean Territories"
## [14] "Jersey"
## [15] "Montserrat"
## [16] "Niue"
## [17] "Norfolk Island"
## [18] "Northern Cyprus"
## [19] "Pitcairn Islands"
## [20] "Saint Barthelemy"
## [21] "Saint Helena"
## [22] "Saint Pierre and Miquelon"
## [23] "Siachen Glacier"
## [24] "Somaliland"
## [25] "South Georgia and South Sandwich Islands"
## [26] "Taiwan"
## [27] "Vatican"
## [28] "Wallis and Futuna"
## [29] "West Bank"
## [30] "Western Sahara"
```

```r
print(unique_demographic_country_names)
```

```
##  [1] ""
##  [2] "Africa Eastern and Southern"
##  [3] "Africa Western and Central"
##  [4] "Arab World"
##  [5] "Caribbean small states"
##  [6] "Central Europe and the Baltics"
##  [7] "Channel Islands"
##  [8] "Data from database: World Development Indicators"
##  [9] "Early-demographic dividend"
## [10] "East Asia & Pacific"
## [11] "East Asia & Pacific (excluding high income)"
## [12] "East Asia & Pacific (IDA & IBRD countries)"
## [13] "Euro area"
## [14] "Europe & Central Asia"
## [15] "Europe & Central Asia (excluding high income)"
## [16] "Europe & Central Asia (IDA & IBRD countries)"
## [17] "European Union"
## [18] "Fragile and conflict affected situations"
## [19] "Gibraltar"
## [20] "Heavily indebted poor countries (HIPC)"
## [21] "High income"
## [22] "IBRD only"
## [23] "IDA & IBRD total"
## [24] "IDA blend"
## [25] "IDA only"
## [26] "IDA total"
## [27] "Last Updated: 07/20/2022"
## [28] "Late-demographic dividend"
## [29] "Latin America & Caribbean"
## [30] "Latin America & Caribbean (excluding high income)"
## [31] "Latin America & the Caribbean (IDA & IBRD countries)"
## [32] "Least developed countries: UN classification"
## [33] "Low & middle income"
## [34] "Low income"
## [35] "Lower middle income"
## [36] "Middle East & North Africa"
## [37] "Middle East & North Africa (excluding high income)"
## [38] "Middle East & North Africa (IDA & IBRD countries)"
## [39] "Middle income"
## [40] "North America"
## [41] "Not classified"
## [42] "OECD members"
## [43] "Other small states"
## [44] "Pacific island small states"
## [45] "Post-demographic dividend"
## [46] "Pre-demographic dividend"
## [47] "Small states"
## [48] "South Asia"
## [49] "South Asia (IDA & IBRD)"
## [50] "Sub-Saharan Africa"
## [51] "Sub-Saharan Africa (excluding high income)"
## [52] "Sub-Saharan Africa (IDA & IBRD countries)"
## [53] "Upper middle income"
## [54] "West Bank and Gaza"
## [55] "World"
```

Upload data into a SQL database

The data is now in a format that could be used for further analysis. Therefore it will be uploaded into an SQL database from which it can be retrieved in a second R script (the analysis script).

Create a connection to the DB

```
# Create a "connection object"
con = dbConnect(
  drv = RSQLite::SQLite(),
  dbname = "yield.db"
)
```

Create a new table in a DB

```
# Check the structure of the data that should be uploaded
str(data)
```

```
## grouped_df [9,331,200 x 14] (S3: grouped_df/tbl_df/tbl/data.frame)
##  $ lon         : num [1:9331200] 180 180 180 180 180 ...
##  $ lat         : num [1:9331200] -89.8 -89.8 -89.8 -89.8 -89.8 ...
##  $ yield       : num [1:9331200] NA NA NA NA NA NA NA NA NA NA ...
##  $ year        : chr [1:9331200] "1981" "1982" "1983" "1984" ...
##  $ lon_180     : num [1:9331200] -180 -180 -180 -180 -180 ...
##  $ country     : Factor w/ 244 levels "Afghanistan",..: 9 9 9 9 9 9 9 9 9 9 ...
##  $ continent   : Factor w/ 7 levels "Africa","Antarctica",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ earth_radius: num [1:9331200] 6356753 6356753 6356753 6356753 6356753 ...
##  $ lat_diff    : num [1:9331200] 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
##  $ lon_diff    : num [1:9331200] 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 ...
##  $ lon_m       : num [1:9331200] 242 242 242 242 242 ...
##  $ lat_m       : num [1:9331200] 55473 55473 55473 55473 55473 ...
##  $ area        : num [1:9331200] 13427074 13427074 13427074 13427074 13427074 ...
##  $ area_ha     : num [1:9331200] 1343 1343 1343 1343 1343 ...
##  - attr(*, "groups")= tibble [12,960 x 3] (S3: tbl_df/tbl/data.frame)
##   ..$ lat  : num [1:12960] -89.8 -89.8 -89.8 -89.8 -89.8 ...
##   ..$ year : chr [1:12960] "1981" "1982" "1983" "1984" ...
##   ..$ .rows: list<int> [1:12960]
##   .. ..$ : int [1:720] 1 37 73 109 145 181 217 253 289 325 ...
##   .. ..$ : int [1:720] 2 38 74 110 146 182 218 254 290 326 ...
##   .. ..$ : int [1:720] 3 39 75 111 147 183 219 255 291 327 ...
##   .. ..$ : int [1:720] 4 40 76 112 148 184 220 256 292 328 ...
##   .. ..$ : int [1:720] 5 41 77 113 149 185 221 257 293 329 ...
##   .. ..$ : int [1:720] 6 42 78 114 150 186 222 258 294 330 ...
##   .. ..$ : int [1:720] 7 43 79 115 151 187 223 259 295 331 ...
##   .. ..$ : int [1:720] 8 44 80 116 152 188 224 260 296 332 ...
##   .. ..$ : int [1:720] 9 45 81 117 153 189 225 261 297 333 ...
##   .. ..$ : int [1:720] 10 46 82 118 154 190 226 262 298 334 ...
##   .. ..$ : int [1:720] 11 47 83 119 155 191 227 263 299 335 ...
##   .. ..$ : int [1:720] 12 48 84 120 156 192 228 264 300 336 ...
##   .. ..$ : int [1:720] 13 49 85 121 157 193 229 265 301 337 ...
##   .. ..$ : int [1:720] 14 50 86 122 158 194 230 266 302 338 ...
##   .. ..$ : int [1:720] 15 51 87 123 159 195 231 267 303 339 ...
##   .. ..$ : int [1:720] 16 52 88 124 160 196 232 268 304 340 ...
##   .. ..$ : int [1:720] 17 53 89 125 161 197 233 269 305 341 ...
##   .. ..$ : int [1:720] 18 54 90 126 162 198 234 270 306 342 ...
##   .. ..$ : int [1:720] 19 55 91 127 163 199 235 271 307 343 ...
##   .. ..$ : int [1:720] 20 56 92 128 164 200 236 272 308 344 ...
##   .. ..$ : int [1:720] 21 57 93 129 165 201 237 273 309 345 ...
##   .. ..$ : int [1:720] 22 58 94 130 166 202 238 274 310 346 ...
##   .. ..$ : int [1:720] 23 59 95 131 167 203 239 275 311 347 ...
##   .. ..$ : int [1:720] 24 60 96 132 168 204 240 276 312 348 ...
##   .. ..$ : int [1:720] 25 61 97 133 169 205 241 277 313 349 ...
##   .. ..$ : int [1:720] 26 62 98 134 170 206 242 278 314 350 ...
##   .. ..$ : int [1:720] 27 63 99 135 171 207 243 279 315 351 ...
##   .. ..$ : int [1:720] 28 64 100 136 172 208 244 280 316 352 ...
##   .. ..$ : int [1:720] 29 65 101 137 173 209 245 281 317 353 ...
##   .. ..$ : int [1:720] 30 66 102 138 174 210 246 282 318 354 ...
##   .. ..$ : int [1:720] 31 67 103 139 175 211 247 283 319 355 ...
##   .. ..$ : int [1:720] 32 68 104 140 176 212 248 284 320 356 ...
##   .. ..$ : int [1:720] 33 69 105 141 177 213 249 285 321 357 ...
##   .. ..$ : int [1:720] 34 70 106 142 178 214 250 286 322 358 ...
##   .. ..$ : int [1:720] 35 71 107 143 179 215 251 287 323 359 ...
##   .. ..$ : int [1:720] 36 72 108 144 180 216 252 288 324 360 ...
##   .. ..$ : int [1:720] 25921 25957 25993 26029 26065 26101 26137 26173 26209 26245 ...
##   .. ..$ : int [1:720] 25922 25958 25994 26030 26066 26102 26138 26174 26210 26246 ...
##   .. ..$ : int [1:720] 25923 25959 25995 26031 26067 26103 26139 26175 26211 26247 ...
##   .. ..$ : int [1:720] 25924 25960 25996 26032 26068 26104 26140 26176 26212 26248 ...
##   .. ..$ : int [1:720] 25925 25961 25997 26033 26069 26105 26141 26177 26213 26249 ...
##   .. ..$ : int [1:720] 25926 25962 25998 26034 26070 26106 26142 26178 26214 26250 ...
##   .. ..$ : int [1:720] 25927 25963 25999 26035 26071 26107 26143 26179 26215 26251 ...
##   .. ..$ : int [1:720] 25928 25964 26000 26036 26072 26108 26144 26180 26216 26252 ...
##   .. ..$ : int [1:720] 25929 25965 26001 26037 26073 26109 26145 26181 26217 26253 ...
##   .. ..$ : int [1:720] 25930 25966 26002 26038 26074 26110 26146 26182 26218 26254 ...
##   .. ..$ : int [1:720] 25931 25967 26003 26039 26075 26111 26147 26183 26219 26255 ...
##   .. ..$ : int [1:720] 25932 25968 26004 26040 26076 26112 26148 26184 26220 26256 ...
##   .. ..$ : int [1:720] 25933 25969 26005 26041 26077 26113 26149 26185 26221 26257 ...
##   .. ..$ : int [1:720] 25934 25970 26006 26042 26078 26114 26150 26186 26222 26258 ...
##   .. ..$ : int [1:720] 25935 25971 26007 26043 26079 26115 26151 26187 26223 26259 ...
##   .. ..$ : int [1:720] 25936 25972 26008 26044 26080 26116 26152 26188 26224 26260 ...
##   .. ..$ : int [1:720] 25937 25973 26009 26045 26081 26117 26153 26189 26225 26261 ...
##   .. ..$ : int [1:720] 25938 25974 26010 26046 26082 26118 26154 26190 26226 26262 ...
##   .. ..$ : int [1:720] 25939 25975 26011 26047 26083 26119 26155 26191 26227 26263 ...
##   .. ..$ : int [1:720] 25940 25976 26012 26048 26084 26120 26156 26192 26228 26264 ...
##   .. ..$ : int [1:720] 25941 25977 26013 26049 26085 26121 26157 26193 26229 26265 ...
##   .. ..$ : int [1:720] 25942 25978 26014 26050 26086 26122 26158 26194 26230 26266 ...
##   .. ..$ : int [1:720] 25943 25979 26015 26051 26087 26123 26159 26195 26231 26267 ...
##   .. ..$ : int [1:720] 25944 25980 26016 26052 26088 26124 26160 26196 26232 26268 ...
##   .. ..$ : int [1:720] 25945 25981 26017 26053 26089 26125 26161 26197 26233 26269 ...
##   .. ..$ : int [1:720] 25946 25982 26018 26054 26090 26126 26162 26198 26234 26270 ...
##   .. ..$ : int [1:720] 25947 25983 26019 26055 26091 26127 26163 26199 26235 26271 ...
##   .. ..$ : int [1:720] 25948 25984 26020 26056 26092 26128 26164 26200 26236 26272 ...
##   .. ..$ : int [1:720] 25949 25985 26021 26057 26093 26129 26165 26201 26237 26273 ...
##   .. ..$ : int [1:720] 25950 25986 26022 26058 26094 26130 26166 26202 26238 26274 ...
##   .. ..$ : int [1:720] 25951 25987 26023 26059 26095 26131 26167 26203 26239 26275 ...
##   .. ..$ : int [1:720] 25952 25988 26024 26060 26096 26132 26168 26204 26240 26276 ...
##   .. ..$ : int [1:720] 25953 25989 26025 26061 26097 26133 26169 26205 26241 26277 ...
##   .. ..$ : int [1:720] 25954 25990 26026 26062 26098 26134 26170 26206 26242 26278 ...
##   .. ..$ : int [1:720] 25955 25991 26027 26063 26099 26135 26171 26207 26243 26279 ...
##   .. ..$ : int [1:720] 25956 25992 26028 26064 26100 26136 26172 26208 26244 26280 ...
##   .. ..$ : int [1:720] 51841 51877 51913 51949 51985 52021 52057 52093 52129 52165 ...
##   .. ..$ : int [1:720] 51842 51878 51914 51950 51986 52022 52058 52094 52130 52166 ...
##   .. ..$ : int [1:720] 51843 51879 51915 51951 51987 52023 52059 52095 52131 52167 ...
##   .. ..$ : int [1:720] 51844 51880 51916 51952 51988 52024 52060 52096 52132 52168 ...
##   .. ..$ : int [1:720] 51845 51881 51917 51953 51989 52025 52061 52097 52133 52169 ...
##   .. ..$ : int [1:720] 51846 51882 51918 51954 51990 52026 52062 52098 52134 52170 ...
##   .. ..$ : int [1:720] 51847 51883 51919 51955 51991 52027 52063 52099 52135 52171 ...
##   .. ..$ : int [1:720] 51848 51884 51920 51956 51992 52028 52064 52100 52136 52172 ...
##   .. ..$ : int [1:720] 51849 51885 51921 51957 51993 52029 52065 52101 52137 52173 ...
##   .. ..$ : int [1:720] 51850 51886 51922 51958 51994 52030 52066 52102 52138 52174 ...
##   .. ..$ : int [1:720] 51851 51887 51923 51959 51995 52031 52067 52103 52139 52175 ...
##   .. ..$ : int [1:720] 51852 51888 51924 51960 51996 52032 52068 52104 52140 52176 ...
##   .. ..$ : int [1:720] 51853 51889 51925 51961 51997 52033 52069 52105 52141 52177 ...
##   .. ..$ : int [1:720] 51854 51890 51926 51962 51998 52034 52070 52106 52142 52178 ...
##   .. ..$ : int [1:720] 51855 51891 51927 51963 51999 52035 52071 52107 52143 52179 ...
##   .. ..$ : int [1:720] 51856 51892 51928 51964 52000 52036 52072 52108 52144 52180 ...
##   .. ..$ : int [1:720] 51857 51893 51929 51965 52001 52037 52073 52109 52145 52181 ...
##   .. ..$ : int [1:720] 51858 51894 51930 51966 52002 52038 52074 52110 52146 52182 ...
##   .. ..$ : int [1:720] 51859 51895 51931 51967 52003 52039 52075 52111 52147 52183 ...
##   .. ..$ : int [1:720] 51860 51896 51932 51968 52004 52040 52076 52112 52148 52184 ...
##   .. ..$ : int [1:720] 51861 51897 51933 51969 52005 52041 52077 52113 52149 52185 ...
##   .. ..$ : int [1:720] 51862 51898 51934 51970 52006 52042 52078 52114 52150 52186 ...
##   .. ..$ : int [1:720] 51863 51899 51935 51971 52007 52043 52079 52115 52151 52187 ...
##   .. ..$ : int [1:720] 51864 51900 51936 51972 52008 52044 52080 52116 52152 52188 ...
##   .. ..$ : int [1:720] 51865 51901 51937 51973 52009 52045 52081 52117 52153 52189 ...
##   .. ..$ : int [1:720] 51866 51902 51938 51974 52010 52046 52082 52118 52154 52190 ...
##   .. ..$ : int [1:720] 51867 51903 51939 51975 52011 52047 52083 52119 52155 52191 ...
##   .. .. [list output truncated]
##   .. ..@ ptype: int(0)
##   ..- attr(*, ".drop")= logi TRUE
```

```r
# Drop a table if it already exists
DDL_query = "
    DROP TABLE IF EXISTS maize_yield;"

rs <- dbSendQuery(con, DDL_query)

# Create table in the SQL database
DDL_query = "
    CREATE TABLE  maize_yield (
        lon_180 NUMERIC,
      lat NUMERIC,
      yield NUMERIC,
      year Text,
      country Text,
      continent Text,
      area_ha NUMERIC,
      PRIMARY KEY (lon_180, lat, year)
);
  "
rs <- dbSendQuery(con, DDL_query)
```

```
## Warning: Closing open result set, pending rows
```

```r
print(rs)
```

```
## <SQLiteResult>
##   SQL
##     CREATE TABLE  maize_yield (
##      lon_180 NUMERIC,
##        lat NUMERIC,
##        yield NUMERIC,
##        year Text,
##        country Text,
##        continent Text,
##        area_ha NUMERIC,
##    PRIMARY KEY (lon_180, lat, year)
## );
##
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
# Check if the table exists
query = "SELECT name FROM sqlite_master WHERE type='table' AND name='maize_yield';"

rs <- dbFetch(dbSendQuery(con, query))
```

```
## Warning: Closing open result set, pending rows
```

```
print(rs)
```

```
##          name
## 1 maize_yield
```

Add data to the table

```
# Select the columns of the dataframes that are supposed to be uploaded
data %>%
  dplyr::select("lon_180", "lat", "yield", "year", "country", "continent", "area_ha") %>%
#  print(.)

# Add the data
dbAppendTable(
  conn = con,
  name = "maize_yield", # name of the table you want to add data to
  value = .
  )
```

```
## Warning: Closing open result set, pending rows
```

```
## Warning: Factors converted to character
```

```
## [1] 9331200
```

check the rows in the table

```
# Select all rows from the table
query = "SELECT
          *

         FROM maize_yield
         LIMIT 5;"

rs <- dbFetch(dbSendQuery(con, query))

print(rs)
```

```
##   lon_180    lat yield year    country  continent   area_ha
## 1 -179.75 -89.75    NA 1981 Antarctica Antarctica 1342.707
## 2 -179.75 -89.75    NA 1982 Antarctica Antarctica 1342.707
## 3 -179.75 -89.75    NA 1983 Antarctica Antarctica 1342.707
## 4 -179.75 -89.75    NA 1984 Antarctica Antarctica 1342.707
## 5 -179.75 -89.75    NA 1985 Antarctica Antarctica 1342.707
```

Create a second table

```
# Drop the table if it already exists
DDL_query = "
    DROP TABLE IF EXISTS demographic_data;"

rs <- dbSendQuery(con, DDL_query)
```

```
## Warning: Closing open result set, pending rows
```

```
# Create the table
DDL_query = "
    CREATE TABLE  demographic_data (
      country Text,
      population NUMERIC,
      gdp NUMERIC,
      income NUMERIC,
      export NUMERIC,
      import NUMERIC,
      year Text,
      PRIMARY KEY (country, year)
);
  "

rs <- dbSendQuery(con, DDL_query)
```

```
## Warning: Closing open result set, pending rows
```

```
print(rs)
```

```
## <SQLiteResult>
##   SQL
##     CREATE TABLE  demographic_data (
##       country Text,
##        population NUMERIC,
##        gdp NUMERIC,
##        income NUMERIC,
##        export NUMERIC,
##        import NUMERIC,
##        year Text,
##    PRIMARY KEY (country, year)
## );
##
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

Add data to the table

```
dbAppendTable(
  conn = con,
  name = "demographic_data", # name of the table you want to add data to
  value = demographic_data_clean
  )
```

```
## Warning: Closing open result set, pending rows

## [1] 9684
```

check the rows in the table

```
query = "SELECT
          *

         FROM demographic_data
         LIMIT 5;"

rs <- dbFetch(dbSendQuery(con, query))

print(rs)
```

```
##       country population        gdp     income    export     import year
## 1 Afghanistan   13171679 3478787909 3241000000 766300000 1080500000 1981
## 2 Afghanistan   12882518         NA       <NA> 798000000  989600000 1982
## 3 Afghanistan   12537732         NA       <NA> 806200000 1062900000 1983
## 4 Afghanistan   12204306         NA       <NA> 841500000 1419400000 1984
## 5 Afghanistan   11938204         NA       <NA> 697400000 1084300000 1985
```

disconnect

```
dbDisconnect(con)
```

```
## Warning in connection_release(conn@ptr): There are 1 result in use. The
## connection will be released when they are closed
```