EPIDEMIA Malaria Forecasting System: Detailed Walk-through Guide

Dawn Nekorchuk, Michael Wimberly, and EPIDEMIA Team Members
Department of Geography and Environmental Sustainability, University of Oklahoma
dawn.nekorchuk@ou.edu; mcwimberly@ou.edu

Updated May 15, 2019

Contents

1 Introduction **1**

2 R scripts **3**
2.1 `run_epidemiar_demo.R`: Part I . . . 3
2.1.1 Loading packages & functions . . . 3
2.1.2 Reading in the data . . . 4
2.2 `model_parameters_amhara.R` . . . 8
2.2.1 Set up forecast controls . . . 8
2.2.2 Set up early detection controls . . . 10
2.3 `run_epidemiar_demo.R`: Part II . . . 11
2.3.1 Run epidemiar & create report data . . . 12
2.3.2 Report data output . . . 13
2.3.3 Merge species data, save, and create pdf report . . . 16
2.3.4 Alternative: Create pdf report . . . 18
2.3.5 Alternative: Rnw compile pdf . . . 18
2.4 `create_epidemiar_demo.R` . . . 18

3 Looping Version **18**

4 Woreda names **19**

1 Introduction

This detailed walk-through will explain each section of the `run_epidemiar_demo.R` script in the `epidemiar-demo` R project.

All surveillance data in this demo is *simulated* and for *demo use only*. The epidemiolgoical data are artificial and should not be used for research of public health purposes. The environmental data, from Google Earth Engine, is real. This walkthrough is adapted from documentation given to our colleagues in Ethiopia.

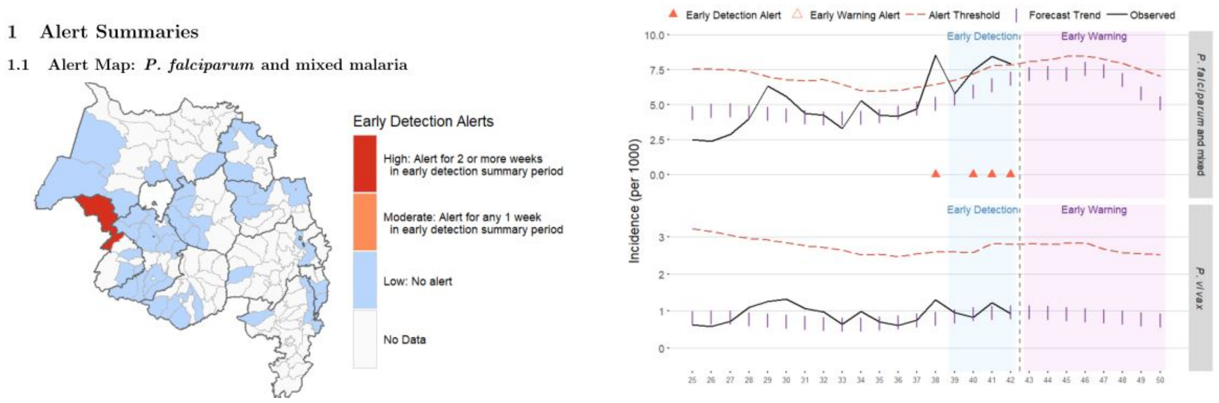
For more details on the `epidemiar` package, see the vignettes:

- Overview: `vignette("overview-epidemiar", package = "epidemiar")`, and
- Input data and modeling parameters: `vignette("data-modeling", package = "epidemiar")`.

This demo is an example of how data and settings can be organized to feed into `epidemiar::run_epidemia()`.

Goal & end product: The final report presents a malaria forecasting report for 47 woredas in the Amhara region for the past 18 weeks through 8 weeks forecasted into the future (26 total weeks). Malaria is broken out by species: *Plasmodium falciparum* and mixed species, and also *P. vivax*. The report includes environmental and epidemiological surveillance data.

Example sections of the final report:



1 Alert Summaries

1.1 Alert Map: *P. falciparum* and mixed malaria

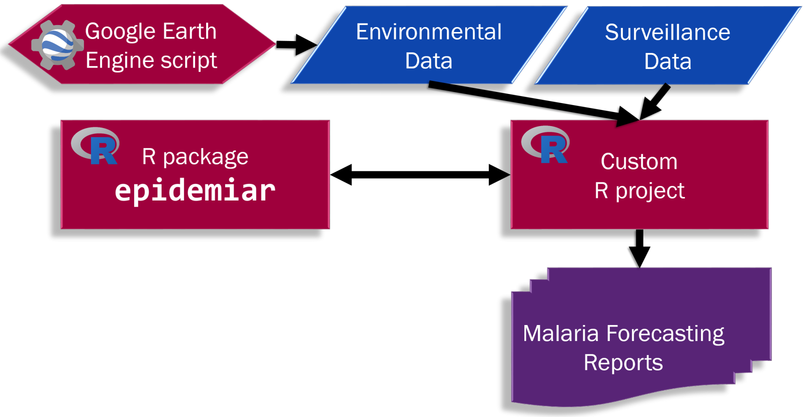
We are going to use an R script to:

- bring in malaria surveillance and remotely-sensed environmental data
- set some model and event detection parameters
- call the epidemiar functions to run the model, forecast, event detection, produce & save the report data ouput
- send the results to a formatting script (`report/epidemia_report_demo.Rnw`, a Sweave file) to create the final pdf report.

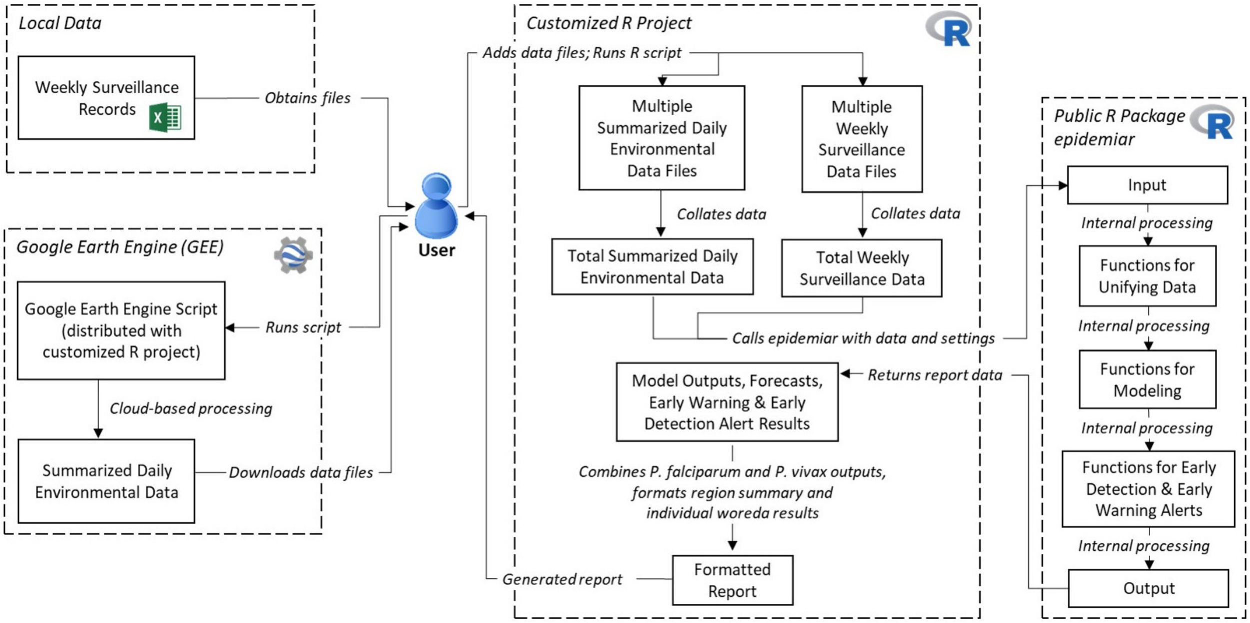
If you have not installed `epidemiar`, or MikTeX ("Install missing packages on-the-fly" = Yes; and restart the computer) yet, please see the Install & Update Guide (`"documentation/install_update.pdf"`) for details on those steps.

Video tutorials were made for our colleagues using a previous version of the software. Some of the details have since changed, but you can view them here: https://www.youtube.com/channel/UC-NKR1cer4wkg8hHHP7K9Vw.

Overview diagram of how `epidemiar-demo` (custom R project) fits into the EPIDEMIA Forecasting System:



A more detailed look at the system:


Local Data
Weekly Surveillance Records
Obtains files
Google Earth Engine (GEE)
Google Earth Engine Script (distributed with customized R project)
Runs script
Cloud-based processing
Summarized Daily Environmental Data
Downloads data files
User
Customized R Project
Adds data files; Runs R script
Multiple Summarized Daily Environmental Data Files
Multiple Weekly Surveillance Data Files
Collates data
Collates data
Total Summarized Daily Environmental Data
Total Weekly Surveillance Data
Calls epidemiar with data and settings
Model Outputs, Forecasts, Early Warning & Early Detection Alert Results
Returns report data
Combines P. falciparum and P. vivax outputs, formats region summary and individual woreda results
Formatted Report
Generated report
Public R Package epidemiar
Input
Internal processing
Functions for Unifying Data
Internal processing
Functions for Modeling
Internal processing
Functions for Early Detection & Early Warning Alerts
Internal processing
Output

2 R scripts

Open the `epidemiar_demo.Rproj`. From here, there are three scripts of major importance.

1. The script `run_epidemiar_demo.R` will bring in data, run and create the model and forecasts, and generate a pdf report.
2. The script `create_model_demo.R` generates only the forecasting model, which can be used in subsequent runs of `run_epidemiar_demo.R`, saving processing time.
3. The parameter file `data/model_parameters_amhara.R` contains the common parameters for both scripts.

2.1 `run_epidemiar_demo.R`: Part I

This script is divided into sections:

1. Loading packages & functions
2. Reading in the data
3. Run epidemia & create report data
4. Merge species data, save, and create pdf report

Part I covers the first two sections.

2.1.1 Loading packages & functions

First, we need to load the R packages and functions we will use in this R script.

```
## Load packages necessary for script

#make sure pacman is installed
if (!require("pacman")) install.packages("pacman")
```

```
## Loading required package: pacman
```

```
#load packages
  #note: data corrals use dplyr, tidyr, lubridate, readr, readxl, epidemiar
  #note: pdf creation requires knitr, tinytex
```

```r
pacman::p_load(dplyr,
               knitr,
               lubridate,
               parallel,
               readr,
               readxl,
               tidyr,
               tinytex,
               tools)
#load specialized package (https://github.com/EcoGRAPH/epidemiar)
library(epidemiar)
```

Using pacman is a convenient way of loading libraries (`library(package-name)`). If a package is missing, it will install it before continuing and prevent the script from stopping with an error.

We use `library()` for the epidemiar package, because this is our project-specific package and does not exist on the R package repository (CRAN), so pacman would fail trying to install epidemiar. See the Install & Update Guide (`"documentation/install_update.pdf"`) for how to install epidemiar, if you have not done so already.

There are also some local R functions, which are local user-defined functions but not part of any package. The functions are in separate R script files that we `source()` so we can use the function. The date_functions offer a function for created a full date (year, month, day) from ISO year and ISO week. The data_corrals functions are how the epidemiological and environmental data are each read in and merged together to create datasets to be given to the epidemiar function for modeling. The report_save_create_helpers offer two functions, one to merge the results from *P. falciparum* & mixed species, and *P. vivax* model data, and another function to send the result data to the formatting script to produce the pdf.

```r
## Locally-defined Functions
source("R/data_corrals.R")
source("R/report_save_create_helpers.R")
```

2.1.2 Reading in the data

The epidemiar modeling and code requires 3 main sets of data:

1) epidemiological data,
2) daily environmental data, and
3) historical environmental reference/climatology data.

A few look-up tables (metadata or informational reference sets) are needed as well.

2.1.2.1 Woreda metadata

First, we read in some information about the woredas in the Amhara region. We are going to create a subset of woredas that are going to be included in the report. A column named `report` in the Excel metadata file `"data/woredas.xlsx"` indicates if the woreda should be in the report. Currently, these are the 47 pilot woredas. To be included in the report, the woreda would need sufficient epidemiolgical data, environmental data, and model cluster ID. (Note: See documentation inside the `epidemiar` package regarding cluster information. Options are available for clustering, or not, of various geographic areas.) The list of report woredas is going to be used to filter the epidemiological and environmental data into a smaller dataset to reduce processing time.

```r
# read in woreda metadata
report_woredas <- readxl::read_xlsx("data/woredas.xlsx", na = "NA") %>%
  dplyr::filter(report == 1)
```

2.1.2.2 Epidemiological data

For the epidemiology data, we will need weekly case counts per woreda of confirmed *P. falciparum* & mixed, and *P. vivax* malaria. To merge data from multiple files, we will use a "data corral" subfolder `data_epidemiological` in the `epidemia-demo` folder.

The data in this demo project have been **simulated**. These data should not be used beyond this demonstration of what a disease report could look like, and should not be taken as indictative of actual epidemiological data.

The `corral_epidemiological()` function will merge all xlsx files in this directory, so you can have multiple files, e.g. one for each year, month, or even by week. No special file names are expected, so you are free to use your standard naming convention. The script is expecting the file to be in your standard format. Specifically, it is looking for the fields: `Woreda/Hospital`, `Budget Year`, `Epi- Week`, `Blood film P. falciparum`, `RDT P. falciparum`, `Blood film P. vivax`, `RDT P. vivax`.

There should be a line for each week and woreda, even for missing data. Any missing (NA) values in the data will be filled in by linear interpolation inside of the epidemiar modeling functions. Gaps in the data, like missing weeks for a woreda, will trigger an error and stop the script. A log file of missing dates will be written, `log_missing_report_epidemiology.csv`, which can be opened with Excel.

No other files should be in this folder. An Excel file that is not epidemiological data will cause the script to fail.

The `corral_epidemiological()` function will loop through all of your xlsx files and combine them. Next, the function will remove any duplicates, choosing data from the file that was most recently modified. The `corral_epidemiological()` function is flexible enough to handle overlapping or partial year files, as long as there are no gaps in dates.

In regards to dates: Converion from Ethiopian date to Gregorian date was done by adding 7 (ISO week >= 28) or 8 (ISO week < 28) years, e.g. budget year 2011 is in year 2018 for weeks 28 - 52, and 2019 for weeks 1 - 27. Currently, no R function exists for converting full Ethiopian dates to full Gregorian calendar dates. The `obs_date` field is the last day of ISO-8601 week.

The epidemiar package needs total malaria case counts per species. To create this variable, we will add the count of positive blood tests and the count of positive rapid diagnositic tests, for each species. `test_pf_tot` = `'Blood film P. falciparum'` + `'RDT P. falciparum'` and `test_pv_only` = `'Blood film P. vivax'` + `'RDT P. vivax'`.

The `corral_epidemiological()` function also needs three additional metadata files:

- Spelling crosswalk: `data/woreda_spellings.xlsx`: This is a crosswalk between the spellings found in the shapefile and the ones found in the epidemiological xlsx file. Additional alternative spellings can be added into this metafile as needed: `woreda_name` is the shapefile spelling, and `to_replace` is the alternative spelling, and a woreda may have multiple entries. (Note: This is not needed in the demo dataset, but kept for reference.)
- Split woreda information: `"data/woredas_split.xlsx"`: This is a list of woredas that have split since the model was fitted and produced. Seperate environmental data does not yet exist for the split woredas. A new shapefile will need to be added to GEE first and historical information gathered. Model fitting with cluster identification was done on the pre-split woreda and until the model has been updated, the report can only be generated on the combined woreda. The split woredas will be added back together, to match the original woreda before the split. More splits can be added: `woreda_name` is the original woreda, and `split_1` and `split_2` are the two woredas that it was split into. (Note: This is not needed in the demo dataset, but kept for reference.)
- Population data: `"data/population_weekly_2012_2030.csv"`: This population data is from the EPIDEMIA project: population living in malarious areas, called population at risk, and total population numbers. The population numbers were estimated out into the past and future, using an estimated population growth factor of 1.018.

The script will use the most recent week of data as the malaria report date, i.e. it will produce a report with forecasting starting the following future week.

```r
# read & process case data needed for report
epi_data <- corral_epidemiological(report_woreda_names = report_woredas$woreda_name)
```

```
## Reading epidemiological data...
```

```
## Processing epidemiological data...
```

```
## Epidemiological data date range is 2012-07-15 to 2018-12-30 (YYYY-MM-DD).
```

2.1.2.3 Environmental data

For the environmental data, daily data is expected for each environmental variable for each woreda.

We are using the previous 181 days of environmental data in modeling the effects of the environmental variables to the case numbers, and so we must have environmental data at least that number of days *before* the first epidemiology data date. The included EPIDEMIA project environmental dataset has environmental data starting 2012 W1. We will set the lag length to 181 days (`lag_length`) in the Forecasting section below.

Updated environmental data will be obtained from Google Earth Engine (GEE), which is a repository for many different types of satellite imagery and will process and summarize this data for us, so that we only have to download a small text-based file.

The script can run with more recent epidemiological data than environmental data. Missing environmental data, up to the date of the last epidemiological data date, will be filled in using a persistence scheme. This scheme uses the last known value of the environmental variable and copies it forward in time for each missing week. If there are many values being estimated, the accuracy of the results may be reduced. We highly recommend getting the latest environmental data available before running the report.

The `corral_environment()` function works similarly. All environmental data files are stored in subfolder `data_environmental/`. The `corral_environment()` function will loop through all csv (GEE) files and combine them. Next, the function will remove any duplicates, choosing data from the file that was most recently modified. Finally, it will check for any gaps in data. Missing days for any environmental variable will cause an error and stop the script. A log file of the missing days will be written, `log_missing_environmental.csv` which can be opened in Excel. To fix the error, run the GEE script for the missing dates and copy the files into the `data_environmental` subfolder.

Only GEE downloaded data in csv format should be added to the `data_environmental/` folder. An csv file that is not GEE-formated environmental data will cause the script to fail.

The `corral_environment()` function takes a single argument, a tibble of woredas that will be included in the report. This list of woredas is used to filter the environmental data into a smaller subset of data. The smaller dataset will take less time to process and the script will run faster.

```r
# read & process environmental data for woredas in report
env_data <- corral_environment(report_woredas = report_woredas)
```

```
## Reading environmental data...
```

```
## Processing environmental data...
```

```
## Environmental data date range (YYYY-MM-DD):
```

```
## # A tibble: 9 x 3
##   environ_var_code start_dt   end_dt    
##   <chr>            <date>     <date>    
## 1 evi              2001-01-01 2018-12-30
## 2 lst_day          2002-01-01 2018-12-30
## 3 lst_mean         2002-01-01 2018-12-30
## 4 lst_night        2002-01-01 2018-12-30
## 5 ndvi             2001-01-01 2018-12-30
## 6 ndwi5            2001-01-01 2018-12-30
## 7 ndwi6            2001-01-01 2018-12-30
## 8 savi             2001-01-01 2018-12-30
## 9 totprec          2001-01-01 2018-12-30
```

2.1.2.4 Optional: Date Filtering

By default, the report will be generated for the last week of available epidemiological data in the `data_epidemiological/` corral folder. If you wish to run a report for a particular previous week, you can quickly filter the data after the corrals. Environmental data does not need to be filtered as it will run with whatever environmental data is or is not available.

```
# ## Optional: Date Filtering for running certain week's report
# # week is always end of the week, 7th day
# req_date <- epidemiar::make_date_yw(year = 2018, week = 52, weekday = 7)
# epi_data <- epi_data %>%
#   filter(obs_date <= req_date)
# env_data <- env_data %>%
#   filter(obs_date <- req_date)
```

2.1.2.5 Environmental reference/climatology data

The environmental reference / climate data file `"data/env_ref_data.csv"` contains a average value for each the environmental variable for each woreda for each week of the year. These data are used with the most recently known daily environmental values to estimate future values of the environmental variables for forecasting. These data are also used to calculate anomalies. Anomalies are the difference between the current value and the historical average value. The anomalies are used in the modelling. The current reference file is a 17 year average (2001 - 2018).

```
# read in climatology / environmental reference data
env_ref_data <- read_csv("data/env_GEE_ref_data.csv", col_types = cols())
```

2.1.2.6 Environmental variable information

The environmental variable information file `"data/environ_info.xlsx"` lists the environmental variables and important information about the variable.

```
# read in environmental info file
env_info <- read_xlsx("data/environ_info.xlsx", na = "NA")
```

- `environ_var_code`: Short name for the environmental variable, using the GEE variable names
- `reference_method`: 'sum' or 'mean' for how to aggregate daily values into weekly values. For example, rainfall would be the 'sum' of the daily values, while LST would be the 'mean' value during that week
- `report_label`: The axis label to be used in creating the formatted report graphs

2.1.2.7 Model object

If previous models have been built, they can be read in here. This will skip the model building steps inside of `run_epidemia()` and save on processing time. Models will need to be rebuilt periodically, and how often that needs to be done is heavily dependent on the specific dataset and reporting needs. The script will look for all of the *P. falciparum* and mixed models (RDS files that start with "pfm") and separately for *P. vivax*. It will automatically choose the model with the latest (most recent) file creation time.

```
# read in latest model to use - select model per species with latest file created time
# pfm
all_pfm_models <- file.info(list.files("data/models/",
                                       full.names = TRUE, pattern="^pfm.*\\.RDS$"))
if (nrow(all_pfm_models) > 0){
  latest_pfm_model <- rownames(all_pfm_models)[which.max(all_pfm_models$ctime)]
  pfm_model_obj <- readRDS(latest_pfm_model)$model_obj
} else { latest_pfm_model <- ""; pfm_model_obj <- NULL }

#pv
all_pv_models <- file.info(list.files("data/models/",
                                      full.names = TRUE, pattern="^pv.*\\.RDS$"))
if (nrow(all_pv_models) > 0){
  latest_pv_model <- rownames(all_pv_models)[which.max(all_pv_models$ctime)]
  pv_model_obj <- readRDS(latest_pv_model)$model_obj
} else { latest_pv_model <- ""; pv_model_obj <- NULL}
```

Alternatively, specific models files can be selected instead:

```
#or select specific file
latest_pfm_model <- "data/pfm_model_xxxxxxx.RDS"
pfm_model_obj <- readRDS(latest_pfm_model)$model_obj

#or select specific model
latest_pv_model <- "data/pv_model_xxxxxxxx.RDS"
pv_model_obj <- readRDS(latest_pv_model)$model_obj
```

The pfm or pv model_obj is what will be given as the `model_obj` argument to `run_epidemia()`. The file path and name in `latest_pfm_model` and `latest_pv_model` will be added to the output report data metadata (`params_meta`) for record keeping.

2.1.2.8 Parameter file

```
# read in forecast and event detection parameters
source("data/model_parameters_amhara.R")
```

This parameter file contains the settings for forecasting and early detection. If a pre-built model is used, this must be the same parameters as when the model was generated. See next section for a full description of this file.

2.2 `model_parameters_amhara.R`

The parameter file contains all the settings for the model, forecasting, and early detection.

2.2.1 Set up forecast controls

In this section, we first set the number of weeks in the report and set up for modeling and forecasting.

The malaria forecasting report will show a total of 26 weeks. We want to forecast eight (8) weeks out into the future from the last known epidemiological data date. The total 26 weeks is therefore 18 weeks of known data and 8 weeks of future forecasts. (Note: These week lengths were not arbitrary, but a consensus agreement from discussions with our Ethiopian colleagues.)

```
#total number of weeks in report (including forecast period)
report_period <- 26
```

```
#forecast 8 weeks into the future
forecast_future <- 8
```

For the Amhara incidence, we report in rates of malaria cases per 1000 people at risk.

```
#report incidence rates per 1000 people
inc_per <- 1000
```

Next, we are going to read in two files that contain which environmental variables to use, and the woreda clustering information. The model is based on a general additive model (GAM) regression of multiple factors, including the woreda cluster, lagged environmental drivers (anomalies of the environmental variables), long terms trends, and seasonality.

There is a switch to instead of using the anomalies of the environmental variables, to use the raw values instead (set `anom_env_var` to FALSE).

```
#read in model environmental variables to use
pfm_model_env <- read_csv("data/falciparum_model_envvars.csv", col_types = cols())
pv_model_env <- read_csv("data/vivax_model_envvars.csv", col_types = cols())

#environmental data should be transformed to anomalies
anom_env_var <- TRUE
```

- These envvars files just have a list of which `environ_var_code` variables to use in the modeling.

For the *P. falciparum* and mixed malaria model, the environmental variables are rainfall, daytime Land Surface Temperature (LST), and Normalized Difference Water Index (NDWI6; a satellite-derived index for vegetation water content). For the *P. vivax* model, the environmental variables are rainfall, the mean of daytime and nighttime LST, and NDWI6.

Woredas were clustered by the pattern of how the malaria incidence responds to the environmental variables. Woredas where these *interactions* between malaria incidence and environment variables were similar were placed in the same cluster. This gives us greater power for the forecast modeling because we have more datapoints in that cluster than in a single woreda alone.

```
#read in model cluster information
pfm_clusters <- read_csv("data/falciparum_model_clusters.csv", col_types = cols())
pv_clusters <- read_csv("data/vivax_model_clusters.csv", col_types = cols())
```

- The field `cluster_id` gives the cluster value for each woreda, by `woreda_name`.

(Note: Clustering is not necessary, you may also run with a global or individual model. See the vignette in `epidemiar` on modeling data inputs `vignette("data-modeling", package = "epidemiar")` for more details.)

We want to use the previous 181 days, the "lag length", of environmental data in modeling the effects of the environmental variables on malaria case numbers.

Each woreda and week is associated with environmental data on the day the week began, up to 180 days in the past, so that each woreda-week has a 181-day history of weather data. A distributed lag basis is created with the natural cubic splines function, including intercept, with knots at 25%, 50%, and 75% of the lag length. The 5 basis functions that result are multiplied by each woreda's history, so that there are just 5 summary statistics, instead of 181, for every combination of woreda, week, and environmental covariate.

```
#set maximum environmental lag length (in days)
lag_length <- 181
```

Next are a few technical settings for how the modeling itself is run. There is an option to fit the model week by week, but this is very slow, and is not noticeably different from fitting the model once in our data. We are also taking advantage of parallel processing. We are splitting the modeling into pieces that run *at the same* *time* on different processors on the computer. (Note: BAM can only efficiently use a number of cores. If you are working on a computer with a large number, e.g. 24, you may wish to experiement with a set value around six or so.)

```
#model fit frequency: fit once ("once), or fit every week ("week")
fit_freq <- "once"

#set number of cores to use for parallel processing (nthreads in bam discretization)
  #default value is the number of physical cores minus 1, minimum 1 core.
cores <- max(detectCores(logical=FALSE) - 1, 1)
```

Finally, we just put the forecasting controls into a list, so that we have one list object per species with all the controls for the forecasting.

```
#make control lists
pfm_fc_control <- list(env_vars = pfm_model_env,
                       anom_env = anom_env_var,
                       clusters = pfm_clusters,
                       lag_length = lag_length,
                       fit_freq = fit_freq,
                       ncores = cores)

pv_fc_control <- list(env_vars = pv_model_env,
                      anom_env = anom_env_var,
                      clusters = pv_clusters,
                      lag_length = lag_length,
                      fit_freq = fit_freq,
                      ncores = cores)
```

2.2.2 Set up early detection controls

In this section, we set up the parameters for the early detection algorithm.

The malaria forecasting report shows a total of 26 weeks. This will be 18 weeks of known data, and 8 weeks of forecasted values. Within the 18 weeks of known data, the most recent four (4) weeks are designated as the 'early detection period'. (Note: These lengths were not arbitrary, but a consensus agreement from discussions with our Ethiopian colleagues.)

```
#number of weeks in early detection period
# (last n weeks of known epidemiological data to summarize alerts)
ed_summary_period <- 4
```

Next, we are going to set up the parameters used in the Farrington improved event detection algorithm. We will be using the `farringtonFlexible()` function as implemented in the `surveillance` package. We specify that we are using this algorithm with `ed_method = "Farrington"`. Currently, the only other built option is "None" for no event detection.

The central idea of event detection is to identify when the numer of cases exceeds a baseline threshold, and this detection happens as close to real-time as possible. Event detection is done in a prospective manner, where the algorithm knows past data up to the present. This is different from retrospective methods, where events are identified from historical data. There are many different prospective event detection algorithms. Each algorithm calculates baseline thresholds differently and have different assumptions about the pattern of disease transmission, speed of outbreak development, seasonality, and trends.

The Farrington Original method was developed in 1996 and an improved version was released in 2013. The original method is used at Statens Serum Institut in Denmark, Centre for Infections of the Health Protection Agency (HPA) UK, National Institute for Public Health and the Environment (RIVM) the Netherlands (as of 2010). The Farrington methods are based on quasi-Poisson regression, and allows for long-term trend adjustments, seasonality, and re-weighting of past event case numbers.

The parameters below are the Farrington improved versions we tested that had the highest percent of events caught and the lowest rate of false alarms. The Farrington method performed better than the other methods we compared: Center for Disease Control and Prevention (CDC) Early Aberation Reporting System (EARS) and the first EPIDEMIA system using dynamic linear modeling. We tested each species separately, so there are different settings for *P. falciparum* (and mixed) and *P. vivax*. The parameters are all added to a list so that we have one object that has all the controls for Farrington event detection.

```r
#event detection algorithm
ed_method <- "Farrington"

#settings for Farrington event detection algorithm
pfm_ed_control <- list(
  w = 3, reweight = TRUE, weightsThreshold = 2.58,
  trend = TRUE, pThresholdTrend = 0,
  populationOffset = TRUE,
  noPeriods = 12, pastWeeksNotIncluded = 4,
  thresholdMethod = "nbPlugin")

pv_ed_control <- list(
  w = 4, reweight = TRUE, weightsThreshold = 2.58,
  trend = TRUE, pThresholdTrend = 0,
  populationOffset = TRUE,
  noPeriods = 10, pastWeeksNotIncluded = 4,
  thresholdMethod = "nbPlugin")
```

Farrington flexible settings:

- w: the number of timepoints in the window
- reweight: if identified past events are reweighted lower (so past events don't raise the new thresholds too high)
- weightsThreshold: the default value 2.58 in the Farrington revised algorithm that was found to be best performing by the authors who updated the algorithm
- trend: Use trend weighting over the past years
- pThresholdTrend: 0 value means always use trend adjustment
- populationOffset: Use population to adjust case numbers
- noPeriods: break up the year into 10 periods for modeling
- pastWeeksNotIncluded: Discount the first 4 weeks when testing for events to avoid incorrect results when an event is occuring right at the beginning of the period
- thresholdMethod: Use the recommended statistical method for calculating thresholds.

For more details, please run `?surveillance::farringtonFlexible` in the RStudio console to get the help file for this function.

2.3 `run_epidemiar_demo.R`: Part II

This script is divided into sections:

1. Loading packages & functions
2. Reading in the data
3. Run epidemia & create report data
4. Merge species data, save, and create pdf report

Part II covers the last two sections.

2.3.1 Run epidemiar & create report data

Now to actually run the model and generate the report data! Basically, we gather up all the data and settings, and feed it to the `run_epidemia()` function in our `epidemiar` package. We are also setting that we want incidence reported as per 1000 persons.

Each malaria species is run on its own, with their respective settings. The main epidemiological dataset contains both species in different columns, and therefore the `casefield` changes between "test_pf_tot" for *P. falciparum* and mixed species, and "test_pv_only" for *P. vivax*. Similarly, the controls for event detection (`ed_control`) and forecasting (`am_pfm_fc_control`) also change.

After the report data objects have been generated, we add to the metadata, `$params_meta`, a field `$model_used`, to give the file location and name of the model used. If no model was passed in, i.e. it created a model on the fly with the current data, then this field will be an empty string.

Note: Since this is a long script, and early error messages may have been missed, we've added a simple check to make sure the epidemiological and environmental datasets have been generated, and if not, it'll produce an informative message towards this end of the script.

```r
#Run modeling to get report data
# with check on current epidemiology and environmental data sets

if (exists("epi_data") & exists("env_data")){

  # P. falciparum & mixed
  message("Running P. falciparum & mixed")
  pfm_reportdata <- run_epidemia(epi_data = epi_data,
                                 casefield = test_pf_tot,
                                 populationfield = pop_at_risk,
                                 inc_per = inc_per,
                                 groupfield = woreda_name,
                                 week_type = "ISO",
                                 report_period = report_period,
                                 ed_summary_period = ed_summary_period,
                                 ed_method = ed_method,
                                 ed_control = pfm_ed_control,
                                 env_data = env_data,
                                 obsfield = environ_var_code,
                                 valuefield = obs_value,
                                 forecast_future = forecast_future,
                                 fc_control = pfm_fc_control,
                                 env_ref_data = env_ref_data,
                                 env_info = env_info,
                                 model_obj = pfm_model_obj)

  # P. vivax
  message("Running P. vivax")
  pv_reportdata <- run_epidemia(epi_data = epi_data,
                                casefield = test_pv_only,
                                populationfield = pop_at_risk,
                                inc_per = inc_per,
                                groupfield = woreda_name,
                                week_type = "ISO",
                                report_period = report_period,
                                ed_summary_period = ed_summary_period,
                                ed_method = ed_method,
                                ed_control = pv_ed_control,
                                env_data = env_data,
                                obsfield = environ_var_code,
                                valuefield = obs_value,
                                forecast_future = forecast_future,
                                fc_control = pv_fc_control,
                                env_ref_data = env_ref_data,
                                env_info = env_info,
                                model_obj = pv_model_obj)

  #append model information to report data metadata
  pfm_reportdata$params_meta$model_used <- latest_pfm_model
  pv_reportdata$params_meta$model_used <- latest_pv_model

} else {
  message("Error: Epidemiological and/or environmental datasets are missing.
          Check Section 2 for data error messages.")
}
```

```
## Running P. falciparum & mixed

## Preparing for forecasting

## Generating forecasts

## Running early detection

## Running P. vivax

## Preparing for forecasting

## Generating forecasts

## Running early detection
```

The processing time depends on how powerful your computer is and the settings used for modeling. Passing in a model reduces the time, since it does not have to calculate the model first. Expect this to take about 3 to 5 minutes per species on an average computer.

The `run_epidemia()` function returns one object. For *P. falciparum* we called this object `pfm_reportdata`, and `pv_repordata` for *P. vivax*. The reportdata object is a list of tibbles of the outputs from the modeling, early detection and early warning algorithms, and other results. (Note: In the Amhara implementation, we have two species that we then combine the results before sending it to the formatting script. You could have multiple diseases you are working with, or you could have only one. Your situation will then drive how you code your report formatting script/Rnw file.)

2.3.2 Report data output

The `run_epidemia()` function returns one object. For *P. falciparum* we called this object `pfm_reportdata`. This object is a list of tibbles, or dataframes. These tibbles are the outputs from the modeling, early detection and early warning algorithms, and other results.

1. `summary_data`
2. `epi_summary`
3. `modeling_results_data`
4. `environ_timeseries`
5. `environ_anomalies`
6. `params_meta`
7. `regression_object`

(Note: For a more generic description of the reportdata object, including per column definitions, see the output data vignette in the `epidemiar` package: `vignette("output-report=data", package = "epidemiar")`. The following description is specific to the malaria demo data.)

2.3.2.1 summary_data

```
pfm_reportdata$summary_data
```

```
## # A tibble: 47 x 5
##    woreda_name     ed_alert_count ed_sum_level ew_alert_count ew_level
##    <chr>                    <dbl> <ord>                 <dbl> <ord>
##  1 Abargelie                    0 Low                       0 Low
##  2 Alefa                        0 Low                       0 Low
##  3 Andabiet                     0 Low                       0 Low
##  4 Ankesha                      0 Low                       0 Low
##  5 Antsokiya Gemza              0 Low                       0 Low
##  6 Artuma Fursi                 0 Low                       0 Low
##  7 Awabel                       0 Low                       0 Low
##  8 Bahir Dar Zuria              0 Low                       0 Low
##  9 Baso Liben                   0 Low                       0 Low
## 10 Borena                       0 Low                       0 Low
## # ... with 37 more rows
```

This tibble contains the early detection and early warning alert levels for each woreda.

Early detection alerts (`ed_alert_count`) are alerts that are triggered during the early detection period, which is defined as the 4 most recent weeks of known epidemiology data. Similarly, early warning alerts (`ew_alert_count`) are alerts in the future forecast estimates. “High” level indicates two or more weeks in this period had incidences greater than the alert threshold, “Medium” means that one week was in alert status, and “Low” means no weeks had alerts (`ed_sum_level` and `ew_level`, respectively).

2.3.2.2 epi_summary

```
pfm_reportdata$epi_summary
```

```
## # A tibble: 47 x 2
##    woreda_name     mean_inc
##    <chr>              <dbl>
##  1 Abargelie         1.77
##  2 Alefa             0.533
##  3 Andabiet          0.115
##  4 Ankesha           0.258
##  5 Antsokiya Gemza   0.0151
##  6 Artuma Fursi      0.0220
##  7 Awabel            0.0463
##  8 Bahir Dar Zuria   0.135
##  9 Baso Liben        0.684
## 10 Borena            0.0606
## # ... with 37 more rows
```

This tibble holds the mean incidence of malaria in the early detection period per woreda, and is used to generate maps in the third section of the pdf report.

2.3.2.3 modeling_results_data

```
pfm_reportdata$modeling_results_data
```

```
## # A tibble: 4,512 x 9
##    woreda_name obs_date   series value lab   upper lower week_epidemiar
##    <chr>       <date>     <chr>  <dbl> <chr> <dbl> <dbl>          <dbl>
##  1 Abargelie   2018-09-02 obs    0.900 Obse~    NA    NA             35
##  2 Abargelie   2018-09-09 obs    0.919 Obse~    NA    NA             36
##  3 Abargelie   2018-09-16 obs    0.937 Obse~    NA    NA             37
##  4 Abargelie   2018-09-23 obs    1.07  Obse~    NA    NA             38
##  5 Abargelie   2018-09-30 obs    1.14  Obse~    NA    NA             39
##  6 Abargelie   2018-10-07 obs    1.41  Obse~    NA    NA             40
##  7 Abargelie   2018-10-14 obs    1.56  Obse~    NA    NA             41
##  8 Abargelie   2018-10-21 obs    1.80  Obse~    NA    NA             42
##  9 Abargelie   2018-10-28 obs    2.10  Obse~    NA    NA             43
## 10 Abargelie   2018-11-04 obs    2.40  Obse~    NA    NA             44
## # ... with 4,502 more rows, and 1 more variable: year_epidemiar <dbl>
```

This tibble dataset contains multiple timeseries values for observed, forecast, and alert thresholds of malaria incidence, for each woreda. These data are used in creating the individual woreda control charts in the pdf report.

- `series`: “obs” = observed disease incidence, “fc” = modeled/forecast incidence values, “thresh” = event detection threshold values, “ed” = early detection alert (binary), “ew” = early warning alert (binary)
- `value`: Value of the `series` for that woreda for that week
- `lab`: Labels for the series (“Observed”, “Forecast Trend”, “Alert Threshold”, “Early Detection Alert”, “Early Warning Alert”)

2.3.2.4 environ_timeseries

```
pfm_reportdata$environ_timeseries
```

```
## # A tibble: 3,666 x 16
##    woreda_name environ_var_code year_epidemiar week_epidemiar obs_date
##    <chr>       <chr>                     <dbl>          <dbl> <date>
##  1 Abargelie   lst_day                    2018             35 2018-09-02
##  2 Abargelie   ndwi6                      2018             35 2018-09-02
##  3 Abargelie   totprec                    2018             35 2018-09-02
##  4 Abargelie   lst_day                    2018             36 2018-09-09
##  5 Abargelie   ndwi6                      2018             36 2018-09-09
##  6 Abargelie   totprec                    2018             36 2018-09-09
##  7 Abargelie   lst_day                    2018             37 2018-09-16
##  8 Abargelie   ndwi6                      2018             37 2018-09-16
##  9 Abargelie   totprec                    2018             37 2018-09-16
## 10 Abargelie   lst_day                    2018             38 2018-09-23
## # ... with 3,656 more rows, and 11 more variables: val_epidemiar <dbl>,
## #   reference_method <chr>, data_source <chr>, ref_value <dbl>,
## #   ref_sd <dbl>, ref_yrcount <dbl>, ref_max <dbl>, ref_uq <dbl>,
## #   ref_median <dbl>, ref_lq <dbl>, ref_min <dbl>
```

This tibble dataset contains multiple timeseries for the environmental variables for each woreda, and are used to generate the environmental timeseries graphs on the individual woreda report pages.

- `val_epidemiar`: Value of the environmental variable for that geographic group for that week. Values are a combination of observed, or interpolated (for missing) or extended (future estimated) values.

- `data_source`: “Observed”, “Interpolated”, or “Extended”. Missing environmental data is handled in three different ways, depending on time period. For missing values in the middle of series, the value is a linear approximation of surrounding values (“Interpolated”). For missing values at the end of the series, up to the future forecast portion, values are carried forward in a persistence approach (also marked

"Interpolated" at the moment). For the forecast future portion, values are a blending of the last known values and the climatic historical mean, with a gradual weighting scheme shifting from more weight from last known to historical mean ("Extended").

2.3.2.5 environ_anomalies

```
pfm_reportdata$environ_anomalies
```

```
## # A tibble: 141 x 3
##    woreda_name environ_var_code anom_ed_mean
##    <chr>       <chr>                   <dbl>
##  1 Abargelie   lst_day              -0.228
##  2 Abargelie   ndwi6                -0.000560
##  3 Abargelie   totprec              -1.38
##  4 Alefa       lst_day              -0.338
##  5 Alefa       ndwi6                 0.0146
##  6 Alefa       totprec              -0.0222
##  7 Andabiet    lst_day              -1.01
##  8 Andabiet    ndwi6                 0.00729
##  9 Andabiet    totprec              -1.17
## 10 Ankesha     lst_day              -0.442
## # ... with 131 more rows
```

This tibble dataset contains the differences of the environmental variable values from the climatology/reference average during the early detection period. These data are used to make anomaly maps in the third section of the pdf report.

- `anom_ed_mean`: The mean of the anomalies per environmental variable per geographic group summarized during the early detection period. These anomalies are calculated as the difference from the observed value to the historical mean for that week of the year. (Not to be confused with the daily anomalies calculated for modeling and forecasting.)

2.3.2.6 params_meta

This lists all the dates, settings, and parameters that were used in the `run_epidemiar()` function. This keeps a record of all the settings so you can view them later.

2.3.2.7 regression_object

This is the regression object from the general additive model (GAM, parallelized with BAM) regression. This is only for statistical investigation of the model, and is usually not saved because it very large object. To save it, set the `save_reg` argument of `merge_save_report()` to "TRUE" (default is FALSE).

2.3.3 Merge species data, save, and create pdf report

The Amhara malaria demo data has two different reportdata objects (*P. falciparum* and mixed species, and *P. vivax*), which need to be merged together into one object to save and to create the single pdf report.

(Note: Since this is a long script, and early error messages may have been missed, we've added a simple check to make sure the report data for each species have been generated, and if not, it'll produce an informative message towards this end of the script.)

```
if (exists("pfm_reportdata") & exists("pv_reportdata")){

  #merging pfm & pv data, save out, and create pdf
  merge_save_report(rpt_data_main = pfm_reportdata,
                    rpt_data_secd = pv_reportdata,
```

```r
                    #mark sections as P. falciparum and mixed (pfm) or P. vivax (pv)
                    # used in the epidemia_report_am.Rnw file for the formatted report
                    var_labs = c("pfm","pv"),
                    #save out the report data in the file that the formatting file reads
                    save_file = "report/report_data.RData",
                    #save out a second copy of the report data
                    #   with year and week numbers in the name file
                    second_save = TRUE,
                    #create the pdf
                    create_report = TRUE,
                    #which Rnw file to use to create pdf
                    formatting_file = "epidemia_report_demo.Rnw",
                    #show the pdf immediately after creating
                    show_report = TRUE)

} else {
  message("Error: Report data for P. falciparum and/or
          P. vivax have not been generated and are missing.")
}
```

```
## Saved results to /report/report_data.RData
```

```
## Saved results to /report/report_data_2018W52.RData
```

```
## Saving report to /report/epidemia_report_demo_2018W52.pdf
```

This function will save out the merged report_data object in two places:

- `save_file = "report/report_data.RData"`: this file is the *input* file of the Rnw/formatting script (`epidemia_report_demo.Rnw`). The input file of the rnw cannot be particularly changed except by editing the rnw file directly. Therefore this is a 'generic' named file, which is overwritten each time to be used as input to the rnw file.
- `second_save = TRUE`: this saves out a more permanent version of the merged report_data object with an autogenerated year and week numbers in the file name (e.g. "report_data_2018W52.RData" for the report generated with last known epidemiological data of week 2018 Week 52: 2019-12-24 through 2018-12-30.)

(Note: This function also takes in the location/name of the rnw formatting script as well, so there is the possibility of having multiple different report formatting that uses the same input data. The save_file and input file of the formatting_file will need to match up correctly.)

This function calls a subfunction named `create_pdf()` which calls the Sweave formatting script, `report/epidemia_report_demo.Rnw`, to read in the report data (`report/report_data.RData`, as specified in the rnw file) we just created, and to create the formatted pdf report. This `Rnw` Sweave file has been written specifically for the malaria forecasting report in Amhara. The `Rnw` file also uses some additional data: shapefiles (processed into R data files as `data/am.rda` and `data/am_simpl.rda`), and the woreda reference list (`data/woredas.xlsx`).

This subfunction will save two copies of the pdf formatted report:

- `epidemia_report_demo.pdf`: default output of an rnw file - name of the rnw file as a pdf
- `epidemia_report_demo{YYYY"W"WW}.pdf`: a renamed version of the file with autogenerated year and week numbers in the file, same scheme as the second_save of the report_data object, above.

(Note: While only minimally tested, the save function was written to also handle single output dataset, i.e. ones that are not split by species. Give the output to the `rpt_data_main` argument, and leave `rpt_data_secd` NULL.)

2.3.4 Alternative: Create pdf report

You can also call the `create_pdf()` function directly is you want to generate a formatted pdf report from a previously save report_data object. The `new_data` file will overwrite `report_data_file`, which needs to be the input file that is coded inside the `formatting_file` (rnw script).

```
# If you want to later recreate a pdf from a saved report_data file:
# Change the input report_data_file to the previously saved version
# And add a file/name for the saved output.

create_pdf(new_data = "report/report_data_2018W52.RData",
          #file that is loaded in the formatting file
          report_data_file = "report/report_data.RData",
          formatting_file = "epidemia_report_demo.Rnw",
          #specific output file name
          report_save_file = "report/report_data_2018W52.pdf",
          show = TRUE)
```

2.3.5 Alternative: Rnw compile pdf

If you have not set MiKTeX to install missing packages on the fly without asking, the pdf creation will fail. You can compile the pdf once from the Rnw itself, and answer yes to the missing packages installation prompts. Then you can return to using the functions above. In RStudio, open `epidemia_report_demo.Rnw`, and click on `Compile PDF`.

2.4 `create_epidemiar_demo.R`

The `create_epidemiar_demo.R` is set up like `run_epidemiar_demo.R`, but uses the argument `run_epidemia(..., model_run = TRUE)`. This only creates, and returns, a model (regression object) and metadata on the model. The last section saves the model in the `data/models` folder with two dates in the file: the first is the year and week of the epidemiological data available when the model was generated, and the second is the date the model was created.

```
# Save models for later use ---------------------------------------------

# add last epidemiological known data date, and today's date to file name
save_filetail <- paste0("_", isoyear(max(epi_data$obs_date)),
                        "W", isoweek(max(epi_data$obs_date)),
                        "_", format(Sys.Date(), "%Y%m%d"))
pfm_name <- paste0("pfm_model", save_filetail, ".RDS")
pv_name <- paste0("pv_model", save_filetail, ".RDS")

#save to /data
saveRDS(pfm_model, file.path("data/models", pfm_name))
saveRDS(pv_model, file.path("data/models", pv_name))
```

When the `run_epidemiar_demo.R` script runs, it will automatically pull the model with the latest file creation time to use (if the user does not set a specific model file), and feeds it into `run_epidemiar(..., model_obj = {regression object})`.

3 Looping Version

As a prospective report, this script would be run about every week or so as new data comes in. There are times, however, when you want to run a number of historical reports. We've added a version of the script, `run_epidemiar_demo_loopingversion.R` that can be modified (near the top) to run for any number of past weeks.

The example below would be for isoweeks 15, 16, 17, an 18 in isoyear 2016, plus those same weeks in 2017. The `weekday` variable is “7” here, indicating the date at the end of the isoweek, which is what is used in the simulated malaria dataset.

```
# This version of the script can be used to loop through multiple weeks
#   to generate reports for each.
#
# Set the loop variable to TRUE, and change the isoyear and isoweeks wanted.
#

loop <- TRUE
wk_list <- c(epidemiar::make_date_yw(year = 2016, week = c(15:18), weekday = 7),
             epidemiar::make_date_yw(year = 2017, week = c(15:18), weekday = 7))
wk_list
```

```
## [1] "2016-04-17" "2016-04-24" "2016-05-01" "2016-05-08" "2017-04-16"
## [6] "2017-04-23" "2017-04-30" "2017-05-07"
```

Messages will be printed to the console on which week is currently running. It will save the RData and pdf files with year-week tags in the name, and not immediately pop up pdfs when they are created (`show = FALSE` in `merge_save_report()`).

4 Woreda names

For reference, here is a list of report woreda names:

```
readxl::read_xlsx("data/woredas.xlsx", na = "NA") %>%
  filter(report == 1) %>% pull(woreda_name)
```

```
##  [1] "Abargelie"        "Alefa"            "Andabiet"
##  [4] "Ankesha"          "Antsokiya Gemza"  "Artuma Fursi"
##  [7] "Awabel"           "Bahir Dar Zuria"  "Baso Liben"
## [10] "Borena"           "Bugna"            "Burie Zuria"
## [13] "Debre Elias"      "Dehena"           "Dembecha"
## [16] "Denbia"           "Dera"             "Efratana Gidim"
## [19] "Estea"            "Fagita Lekoma"    "Fogera"
## [22] "Gondar Zuria"     "Gonji Kolela"     "Gozamin"
## [25] "Guagusa Shekudad" "Jabi Tehnan"      "Jawi"
## [28] "Jilie Timuga"     "Kalu"             "Kewet"
## [31] "Kobo Town"        "Lasta"            "Libokemkem"
## [34] "Mecha"            "Mekit"            "Merehabete"
## [37] "Metema"           "Misrak Belesa"    "North Achefer"
## [40] "Quara"            "Raya Kobo"        "Sehela"
## [43] "Shewa Robit"      "South Achefer"    "Tehulederie"
## [46] "Womberma"         "Yilmana Densa"
```