Outbreak Monitoring using ASMODEE: an example of automated data infrastructure

Thibaut Jombart

2021-07-15

Contents

Preface

The analysis pipeline described in this handbook is the result of the work of many colleagues at the WHO and beyond. This includes, in alphabetic order:

- *Code contributors in the COVID-19 analytics team*: Neale Batra, Finlay Campbell, Yuka Jinnai, Henry Laurenson-Schaffer
- *other code contributions*: Tim Taylor
- *feedback and inputs*: Brett Archer, Raquel Medialdea Carrera, Laura Guzman, Esther Hamblion, Orlagh Ingeborg, Zyleen Kassamali, Olivier le Polain, Katelijn Vandemaele
- *supervision*: Olivier le Polain

Chapter 1

Introduction

Welcome to this short handbook describing an outbreak dynamics surveillance system for COVID-19. This infrastructure was developed to provide weekly assessments of risk levels for different countries in relation to the countries' overall levels of infection, epidemic growth, and indications of trend acceleration.

This infrastructure is written in **R**, and relies on the following key components:

- *Rmarkdown* (`.Rmd`) reports implementing the data gathering, preparation, analysis, and producing synthesis reports
- the *reportfactory* providing the backbone of the infrastructure
- a public *github* repository used for hosting files, version control, and automation through *github actions*

1.1 What does this infrastructure do?

This infrastructure provides an automated data pipeline that gathers publicly available data on COVID-19 cases, deaths, tests, as well as on Variants of Interest/Concerns (VoI / VoC) and vaccination by country, and provides a weekly assessment of COVID-19 dynamics and levels of risks. The pipeline is implemented as a reportfactory which simplifies the handling of dependencies, data and scripts, and enables the compilation of analysis reports stored locally in dedicated, time-stamped folders.

A series of `Rmd` reports performs data gathering, various analysis, and produce the following information products:

- png figures (*pinplots* and *tadpoles*) summarising the dynamics per WHO region, stored on github and visible on the README.md page (i.e. landing page of the repository)

- `rds` files containing the results for each WHO region, also stored on github, which can then be used for further investigation or for interactive dashboards (e.g. shiny apps); some of these analyses are computer-intensive and would not be readily available on dashboards otherwise
- automated emails to a specific list of recipients providing a detailed report of countries classified into epidemiological risk levels based on incidence, growth and trend changes; this latter part is not stored on github

1.2 Content of this handbook

This handbook is organised into the following chapters:

- **Chapter 2 - Getting started**: instructions to run the infrastructure locally
- **Chapter 3 - Reports overview**: outline of the `Rmd` reports and how they interact
- **Chapter 4 - Implementing ASMODEE**: methodological and practical considerations for implementing ASMODEE, a new method for trend change detection
- **Chapter 5 - Automation**: details on how analyses are automated using github actions

1.3 Disclaimer

This repository contains work in progress. It implements an automated data pipeline for country-level surveillance of COVID-19 dynamics using publicly available data. Its content should not be used for publications without explicit agreement from the authors. The accuracy of the estimates provided in these analyses is contingent on data quality and availability. Results presented here do not represent the official view of the WHO, its staff or consultants. Caution must be taken when interpreting all data presented, and differences between information products published by WHO, national public health authorities, and other sources using different inclusion criteria and different data cut-off times are to be expected. While steps are taken to ensure accuracy and reliability, all data are subject to continuous verification and change. All counts are subject to variations in case detection, definitions, laboratory testing, vaccination strategy, and reporting strategies. See https://covid19.who.int/info/ for further background and other important considerations surrounding the source data. The designations employed and the presentation of these materials do not imply the expression of any opinion whatsoever on the part of WHO concerning the legal status of any country, territory or area or of its authorities, or concerning the delimitation of its frontiers or boundaries. We seek to provide results for countries in all WHO regions (see inclusion criteria below). For technical

reasons (missing data, low incidence, model not successfully fitting to the data), results may not be available for some countries. Where this happens, the list of missing countries is indicated in the relevant sections.

1.4 Licensing

Chapter 2

Getting started

In this chapter, we provide an overview of the infrastructure and of how the data pipelines work. After outlining the main folders and files and their respective roles, we provide installation guidelines and tips for using it locally (i.e. on your computer).

2.1 Overview of the infrastructure

The infrastructure implements data pipelines that download national COVID-19 data from online sources, performs some analyses, classify countries by **epidemiological level of risk** (**ELR**) and produce synthesis reports which can be emailed to a pre-defined list of recipients. It is summarised in figure 2.1. The infrastructure itself is a *reportfactory*, which provides a structure for storing data, auxiliary scripts, reports sources and their outputs. Its key components include:

- a set of **Rmarkdown reports** detailed in **chapter 3**
- a set of **github actions** implementing automated tasks on github servers, and detailed in **chapter 5**

2.2 Main folders and files

The main components are:

- **data/**: a folder where raw and clean data are stored; the content of this folder is not stored on github to minimize the size of the git archive
- **report_sources/**: a folder containing the sources of the `Rmd` reports doing the data gathering and cleaning, and all analyses; see the reports chapter for more information

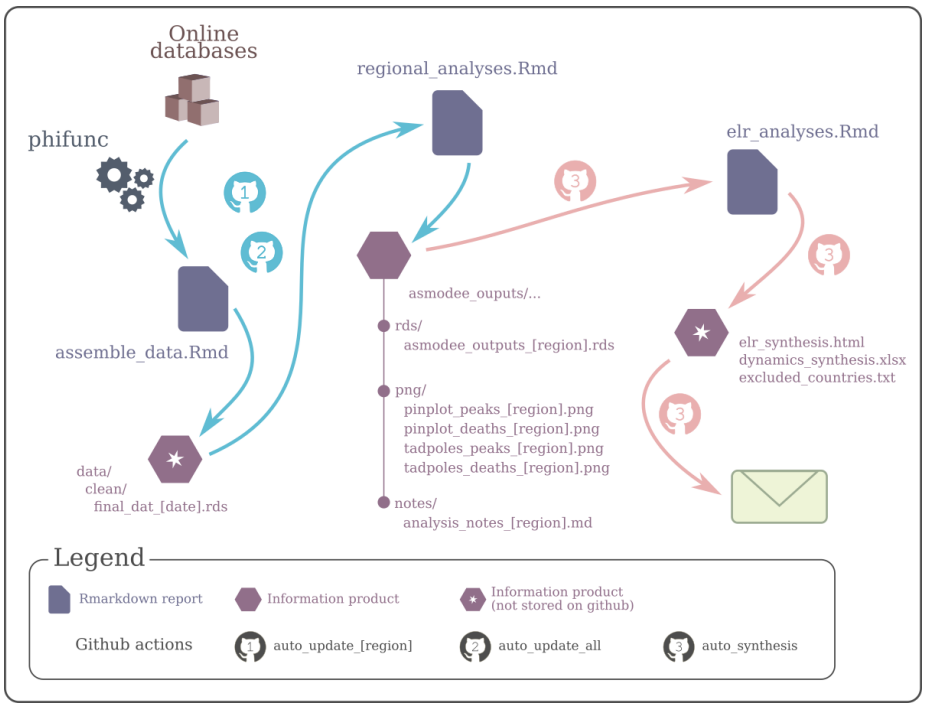


Figure 2.1: Overview of the infrastructure. This diagram represents the flow of information processed by the data pipelines. Data are gathered from the net by the WHO-private package *phifunc*, and processed by a series of *Rmd* reports. Some of the information products generated, indicated by a star, are not stored on github in order to reduce the size of the repository. Upon compilation, all reports outputs are stored in an /outputs/ folder (not shown on the diagram). The automated email (last step) can only be sent by github actions, as it requires SMTP credentials only stored as github secrets.

- **scripts/**: a folder containing `R` scripts used in the various reports; this includes small helper functions to load the latest versions of the data, help with bits of analysis, and code for installing packages not on CRAN, which cannot be handled automatically by `reportfactory::install_deps()`
- **outputs/**: a folder containing all compiled reports and associated outputs, in timestamped folders; this content is automatically generated when using `reportfactory::compile_reports()`, and is only stored locally (i.e. on your machine), and not on github (to keep the git archive to a minimal size)
- **css/**: a folder containing styling for the reports
- **asmodee_outputs/**: a folder containing outputs of the main analysis, shared on github; this includes images (pinplots and tadpoles), **R** objects (`rds` files) containing re-usable analysis results (e.g. for inclusion in websites or dashboards), and analysis notes (indicating excluded countries, and reasons for their exclusions)
- **docs/**: a folder containing the sources and compiled versions of this handbook

Other useful files and folders to know of include:

- **README.Rmd**: the source of the `README.md` file, which is displayed on the landing package of the github project, and shows the latest tadpoles figures by WHO region and also provides links to the corresponding `rds` files
- **run_factory.R**: the main R script for updating data, and running analysis reports for every WHO regions; also used by github actions
- **run_synthesis.R**: an R script used in github actions to generate a synthesis report for all countries, including a classification into levels of risk based on incidence, growth and trend acceleration, and for sending an email to a list of recipients with these documents attached
- **.github/workflows/**: a (hidden) folder storing the files used to define the github actions; see the automation chapter for more information
- **factory_config**: a simple text file containing some information about the name of key directories of the factory; normally will not need editing, unless you rename the folder containing the factory (see 2.3.1)

2.3 Installing the infrastructure

The infrastructure can be installed locally by downloading the repository from github and installing dependencies. To run the full pipeline, the user will need an authentication token used to install the private (i.e. not publicly available) WHO package `phifunc`, which is used for collating data.

2.3.1 Downloading the repository

You can download the repository from the **code** tab on the github page as illustrated in the figure 2.2.

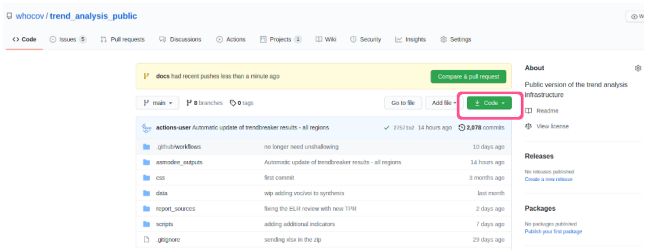


Figure 2.2: Downloading or cloning the repository. The repository containing the data infrastructure can be downloaded as a zip archive or cloned using github.

We recommend using *SSH* to clone the repository. For more information on setting up an SSH access to github, see this webpage.

Once you are set up to access github using SSH, you can clone the repository using GIT from the command line in your favourite terminal, by typing:

```
git clone git@github.com:whocov/trend_analysis_public.git
```

The advantage of cloning the repository rather than merely downloading a zip archive is that you will be able to update the infrastructure automatically using `git pull`.

By default, your local copy of the repository will be called *trend_analysis_public*. It is possible to change this name, but you will need to update the name of the factory in the **factory_config** file. From now on, we will refer to this folder as **root folder**.

2.3.2 Getting the phifunc authentication token

The tool we use to collate COVID-19 data is a package called `phifunc` developed at WHO. Because this package internally uses some non-public data API, it is currently not shared publicly, and an **authentication token** is needed to be able to install it in **R**. This token is a kind of *password*, stored in a file called `phifunc_token` in the root folder.

The easiest way to add this file is ask for it from someone in the COVID-19 analytics team, and add this file to the root of the project. Make sure you do not alter or rename it. Once `phifunc_token` is present in your infrastructure, the scripts installing dependencies will detect it automatically when run, and you will be able to run all analyses locally.

2.3.3 Installing dependencies

To install the dependencies, open the reportfactory by double-clicking on `open.Rproj`, or otherwise starting an R session in the **root** folder (*trend_analysis_public* by default), and copy-paste the following commands:

```
# install basic packages
pkg <- c("remotes", "reportfactory")
install.packages(pkg)

# install deps for the factory
reportfactory::install_deps()
source(here::here("scripts", "remote_packages.R"))
```

Note that the last operation will require the authentication token (file `phifunc_token`) for installing *phifunc*. We will be working separately on a similar infrastructure using data that are directly publicly available.

2.4 Running the infrastructure locally

The data infrastructure can run in two modes:

- **remotely**, through github actions; this happens at specific times but can also be triggered manually
- **locally**, on your computer

Here, we outline the *local* use. You will find more information about the github actions in the automation chapter.

2.4.1 Updating all analyses

The simplest way to update all analyses is run the script `run_factory.R` (using `source(run_factory.R)`, or step-by-step by opening the file in Rstudio). This will:

- download, assemble and pre-process the most recent data
- run a separate report for each WHO region, storing timestamped output in the `outputs/` folder, as well as figures, notes, and *rds* files in the `asmodee_outputs` folder
- update the *README* file with the new analyses

2.4.2 Updating the data

The report `assemble_data.Rmd` performs the data download, gathering and preparation. To update the data, run the following command within the factory:

```
library(reportfactory)
compile_reports(report = "assemble")
```

2.4.3 Running analyses for a specific region

The report `regional_analyses.Rmd` performs all analyses for a given WHO region, including calculations of growth rates and trend change detection using ASMODEE. The report automatically uses the latest version of the clean data (so needs to be run after `assemble_data.Rmd` if the data need updating). The parameter `who_region` determines which WHO region is analysed. Possible values are: `AFRO`, `EMRO`, `EURO` (default), `PAHO`, `SEARO`, `WPRO`. To run the report, type:

```
library(reportfactory)
reg <- "EURO" # replace as appropriate
cores <- 4
compile_reports(report = "regional_analyses",
                params = list(who_region = reg, n_cores = cores),
                subfolder = reg)
```

where:

- `reg` is the letter code for the WHO region to use (in upper case)
- `cores` is the number of cores to be used for the ASMODEE analyses, which support parallelisation
- `subfolder` indicates the name of the folder in `outputs` where the timestamped results will be stored

2.4.4 Generating the synthesis report

The report `elr_synthesis` collates the latest results for all WHO regions, classifies countries by level of risk, and produces a synthesis html report, alongside an .xlsx file `dynamics_summary.xlsx` summarising results and a list of excluded countries in the text file `excluded_countries.txt`.

To compile this report, use:

```
library(reportfactory)
compile_reports(report = "elr_synthesis")
```

Chapter 3

Overview of the reports

This chapter provides some additional informations about the different *Rmarkdown* reports. For details of data processing and analyses, we refer to the respective documents, as we attempted to document all code in each report. We recommend compiling the reports first and looking at the *html* outputs generated as a more user-friendly alternative to inspecting the source code directly.

Reports can be **compiled** by two ways:

1. using `reportfactory::compile_reports(...)` where `...` is a character string matching the name(s) of the reports to be processed (regular expression accepted); outputs will be stored in time-stamped folders inside *outputs/*
2. using the usual `rmarkdown::render(...)` where `...` is the path of the `Rmd` document to process; outputs will be stored inside *report_sources/*

Some reports accept **parameters**, i.e. variables set at compilation time which can impact the results generated. This is typically used, for instance, to indicate which WHO region the analyses are to be performed on, and is a more sustainable alternative to handling separate reports for every region. For both compilation methods the way to specify parameters to a report is the same; passing a list called `param` when compiling documents. For instance:

```
library(reportfactory)
compile_reports(report = "analyses",
                params = list(who_region = "AFRO")
```

will compile the analysis report *regional_analyses.Rmd* (only match for the word `"analyses"`) for the AFRO region.

Default values of parameters are set in the YAML headers of the respective reports.

3.1 Data preparation: *assemble_data.Rmd*

This report downloads the data and pre-processes it to enable further analyses.

3.1.1 Inputs

The package *phifunc* is used to download publicly-available, national COVID-19 data, and to do some of the pre-processing. Unfortunately, this package uses WHO internal APIs for some of these tasks and cannot be made publicly available without exposing these APIs, which would be a security breach. As a result, the package can only be installed with an **authentication token**, which needs to be stored in a file called *phifunc_token* at the root of the project folder.

3.1.2 Outputs

The main output of the report is a clean data file containing data for all countries named *final_dat_[date].rds*, a copy of which will be stored in the *data/clean* folder. These files are ignored by *git*, and will not be added to the *github* repository to reduce the size of the archive.

3.1.3 Parameters

The report accepts the following parameters:

- `import_data`: a `logical` indicating if the data should be downloaded (`TRUE`, default), or if local copies of the last raw data downloaded should be used (`FALSE`)

3.1.4 Example compilation

The following instructions will compile the report:

```
library(reportfactory)
compile_reports(report = "assemble")
```

3.2 Analyses of COVID-19 dynamics: *regional_analyses.Rmd*

This report uses the latest clean data to perform a range of analyses for a specific WHO region. Analyses include estimation of the growth rate, detection of trend changes using ASMODEE, and figures summarising the overall dynamics of COVID-19 by country (pinplots and tadpoles).

3.2.1 Inputs

The latest data are automatically detected and loaded by the auxiliary function `load_final_data()`, defined in *scripts/data_loaders.R*

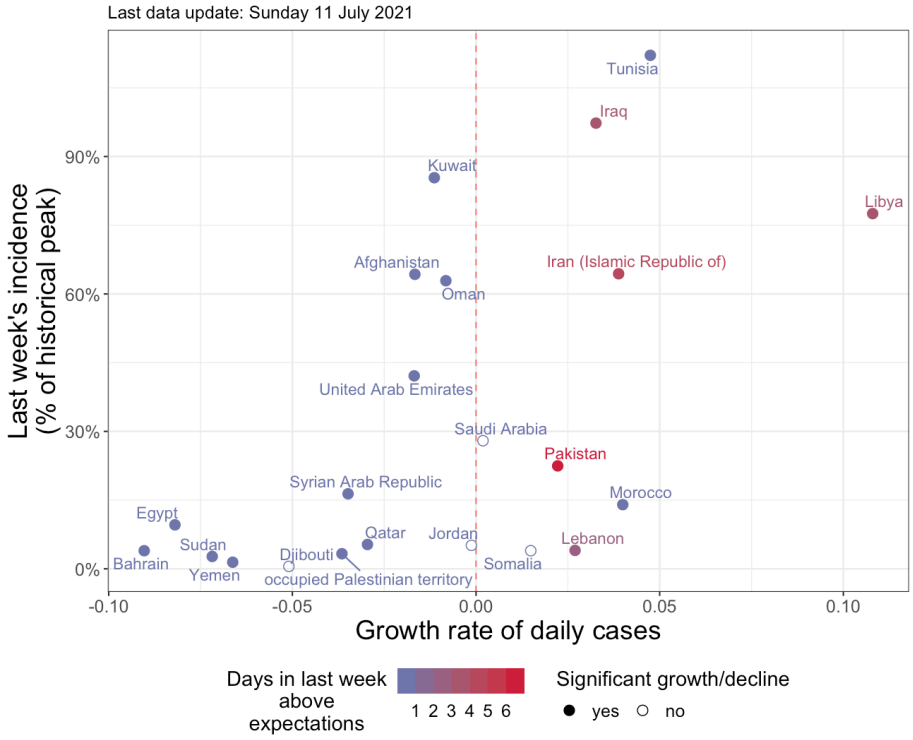


Figure 3.1: Example of pinplot. This figure summarises the dynamics of COVID-19 at a national level, using three complementary metrics: the daily growth rate *r* (*x* axis), the currentl weekly incidence (*y* axis), and the number of net increases showing trend acceleration detected by ASMODEE (colors). Another variant of this figure uses weekly incidence of deaths per capita on the *y* axis. Countries to the right show faster epidemic growth, and countries near the top experience high levels of incidence. Countries displayed in red show signs of acceleration, so that they may move further to the right in the coming days. This figure was generated for EMRO on 11th July 2021.

3.2.2 Outputs

The main output of the report is a `list` exported as a file named *asmodee_outputs*[WHO region]*[data].rds* stored in */asmodee_outputs/rds/*, and containing the following elements:

- `$summary`: summary of the ASMODEE results

- `$results`: outputs of ASMODEE
- `$plot_overall_deaths`: *ggplot2* object of the overall dynamics plot using death per capita on the y-axis
- `$plot_overall_peaks`: *ggplot2* object of the overall dynamics plot using incidence as percentage of historical peak on the y-axis
- `$df_dynamics`: a `data.frame` containing all the required info to recreate either global dynamics plots
- `$elr_extras`: a `data.frame` containing additional information for countries, including TPR and vaccination coverage
- `$timestamp`: the timestamp corresponding to the date of the dataset used in the analyses

Other outputs include:

- **pinplots** and **tadpoles** figures saved *asmodee_outputs/png/*; these figures summarise the COVID-19 dynamics according to the *growth rate*, the current weekly *incidence*, and the number of net accelerations identified by ASMODEE; pinplots show the current situation (see example in Figure 3.1, while tadpoles show the trajectories of countries over the last few days
- **notes** listing countries what were excluded from the analyses alongside the reason for their exclusion, stored in a markdown file named *analysis_notes*[WHO region]*[data].md* stored in */asmodee_outputs/notes/*

3.2.3 Parameters

The report accepts the following parameters:

- `who_region`: the code of the WHO region used in the analyses; possible values are (keep the upper case): `AFRO`, `EMRO`, `EURO` (default), `PAHO`, `SEARO`, `WPRO`
- `n_cores`: the number of cores/processors to be used for ASMODEE; as the method supports parallelisation, more cores usually mean faster analyses; defaults to 1 (no parallelisation)
- `tadpole_size`: the number of days to be considered when showing the trajectories of the countries on tadpoles plots; defaults to 7

3.2.4 Example compilation

The following instructions will compile the report for EMRO, using 12 cores for the calculations; also note the use of `subfolder` which will ensure that results are stored in a timestamped folder in *outputs/EMRO/* (rather than in *outputs/*):

```
library(reportfactory)
rmarkdown::render('regional_analyses',
```

```
                  params = list(n_cores = 12, who_region = "EMRO"),
                  subfolder = "EMRO")
```

The following loop will do the same but for every region:

```
library(reportfactory)
regions <- c("AFRO", "EMRO", "EURO", "PAHO", "SEARO", "WPRO")

for (reg in regions) {
rmarkdown::render('regional_analyses',
                  params = list(n_cores = 12, who_region = reg),
                  subfolder = reg)
}
```

3.3 Synthesis: *elr_synthesis.Rmd*

This report compiles results of all WHO regions, defines ELR for the countries, and stores the main results into an *xlsx* file.

3.3.1 Inputs

The report compiles the latest results obtained for the different WHO regions stored in `asmodee_outputs/rds/`.

3.3.2 Outputs

Based on the metrics used in the pinplots, countries are assigned an ELR using the algorithm summarised in figure 3.2. The report provides complementary informations including trend plots (ASMODEE results) for the countries with *Medium* risk or higher, and some additional information on Testing Positivity Rates (TPR) and vaccination coverage.

Besides the `html` version of the report, other outputs include:

- `dynamics_synthesis.xlsx`: and Excel spreadsheet providing the ELR for each country alongside the different metrics used in the algorithm as well as additional information on TPR and vaccination coverage
- `excluded_countries.txt`: a text file listing all countries that were excluded from the analysis alongside the reason for their exclusion

3.3.3 Parameters

The report does not accept parameters.

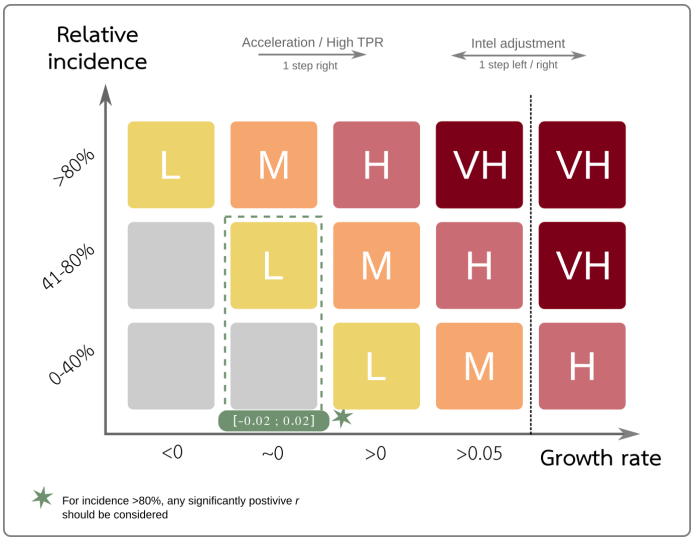


Figure 3.2: Algorithm to define Epi Level of Risk (**ELR**). This diagram summarises the algorithm used to define a risk level (*Low* / *Medium* / *High* / *Very High*) based on values of the daily growth rates, weekly incidence, and signs of acceleration in transmission. Grey squares indicate *Minimal* risk. *Intel adjustments* represent the possibility for other sources of information (e.g. status of the healthcare system) to shift ELR by one level (or more in extreme cases) right or left.

3.3.4 Example compilation

The following instructions will compile the report:

```
library(reportfactory)
compile_reports(report = "synthesis")
```

Chapter 4

Implementing ASMODEE

ASMODEE is a new method for detecting recent changes in temporal trends introduced by Jombart et al. (Jombart et al., 2021) and implemented in the R package *trendbreaker* (Schumacher and Jombart, 2021a) as the function `asmodee`. Rather than attempting to estimate significant changes in growth rates or reproduction numbers, which can usually only be done after changes have been taking place for a week or two, ASMODEE tries to answer the question: "*Are the last few days matching what we would expect given the previous trend in the data?*".

To answer this question, ASMODEE implements the following approach (Figure 4.1):

1. Split the data in two sets: a **testing** set formed by the most 'recent' data (typically the last week), and a **fitting** set one used for charactering trends on the past few weeks (typically 6 - 10 weeks) prior to the testing set.
2. Define a range of **candidate models** to characterise temporal trends in the **fitting set**.
3. Extrapolate past trends to derive a **95% prediction interval** (**PI**) for the last week of data.
4. Identify data outside the PI as outliers, suggesting either a slow-down (below the PI), or an acceleration (above the PI). In our algorithm for defining ELR for countries, we use a criteria of **3 net increases** as a sign of acceleration; *net increases* are defined as the number of outliers above the PI, minus the number of outliers below the PI, in the last 7 days.

Step 2 is the crucial one to obtain good results. Defining the right set of **candidate models** to capture past trends is non-trivial, and is also the non-standard part of implementing ASMODEE, as model-generation is currently not implemented in *trendbreaker*, and requires *ad-hoc* code. In this chapter, we pro-

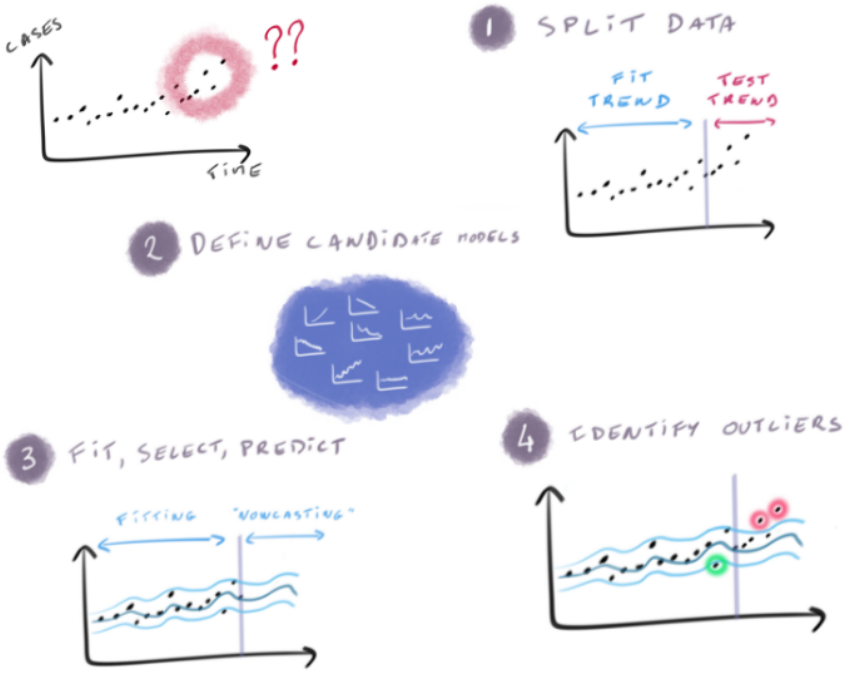


Figure 4.1: Rationale of ASMODEE. This figure illustrates the key steps of ASMODEE for detecting recent changes in temporal trends.

vide tips and explanations on how candidate models are generated, and can be adapted to other data streams.

4.1 A unified interface for linear models

In this section, we outline the general principle of model generation for ASMODEE. All models used in ASMODEE use *trending* (Schumacher and Jombart, 2021b), which provides a general interface for different types of statistical models, including:

- `lm_model`: linear regression, wrapper for `lm`
- `glm_model`: generalized linear models (GLM), wrapper for `glm`
- `glm_nb_model`: negative binomial GLM, wrapper for `MASS:glm.nb`

The advantage of this interface is consitency of behaviours for various operations, e.g. fitting, predictions, confidence intervals and prediction intervals.

The formula syntax of these models is the same as in regular models, so the user should not have new difficulties specifying models with *trending*. For more information on the package, see the dedicated website.

To use `asmodee`, the user needs to provide a `list` of *trending* models. An example of such a list is provided by `models` in the code below, which implements different models of cases over time:

- a constant model with Gaussian error
- a linear temporal trend with Gaussian error
- a log-linear temporal trend with Poisson distribution
- a log-linear temporal trend with Negative Binomial distribution

```
library(trending)

models <- list(
  cst = lm_model(cases ~ 1),
  linear = lm_model(cases ~ date),
  poisson = glm_model(cases ~ date, family = poisson),
  nb = glm_nb_model(cases ~ date)
)
```

This assumes the data we will fit these models to will have a `cases` and a `date` column containing, respectively, the daily case incidence and the corresponding date as a `Date` object. However, these models would capture only simple trends (constant, linear, or exponential), and data are typically more complicated. The following sections illustrate how more flexible models can be added to the list of candidate models.

4.2 Generating candidate models: general principle

The approach to generating many candidate models used in `regional_analysis.Rmd` is to generate `character` strings matching *trending* models definitions (see previous section) and then using the `eval(parse(text = my_text))` trick to turn these into actual models. The simplest approach is to:

1. generate text matching the right-hand side of the formula of the different models (we later refer to this as **model content**)
2. build `character` strings matching model calls using the terms in 1); `sprintf` is particularly handy for this part
3. transform these `character` strings to actual model calls within a `lapply`

Here is a toy example illustrating the approach:

```
library(trending)

# step 1
mod_content <- c("1", "test", "date", "date + tests")

# step 2
models_txt <- sprintf(
  "glm_model(cases ~ %s, family = poisson)",
  mod_content)

# step 3
models <- lapply(models_txt, function(e) eval(parse(text = e)))
class(models) # this is a list
## [1] "list"
length(models) # each component is a model
## [1] 4
lapply(models, class) # check classes of each model
## [[1]]
## [1] "trending_glm"   "trending_model"
##
## [[2]]
## [1] "trending_glm"   "trending_model"
##
## [[3]]
## [1] "trending_glm"   "trending_model"
##
## [[4]]
## [1] "trending_glm"   "trending_model"
```

As the main thing that changes across models is the **model content**, the main task boils down to generating combinations of predictors to capture different trends in the data. To this end, we will use `expand.grid`, which creates all possible combinations of a given set of variables. For instance, to generate all models which:

- include a `date` effect
- may include a `test` effect
- may include a `weekday` effect
- may include a previous day's incidence as predictor (`cases_lag_1`), i.e. autoregressive model

We can use:

```
# generate all combinations
mod_content_grid <- expand.grid(c("", "tests"),
                                "date",
                                c("", "weekday"),
                                c("", "cases_lag_1"))
mod_content_grid
##    Var1 Var2    Var3        Var4
## 1       date
## 2 tests date
## 3       date weekday
## 4 tests date weekday
## 5       date         cases_lag_1
## 6 tests date         cases_lag_1
## 7       date weekday cases_lag_1
## 8 tests date weekday cases_lag_1

# concatenate the columns
mod_content <- apply(mod_content_grid, 1, paste, collapse = " + ")
mod_content
## [1] " + date +  + "
## [2] "tests + date +  + "
## [3] " + date + weekday + "
## [4] "tests + date + weekday + "
## [5] " + date +  + cases_lag_1"
## [6] "tests + date +  + cases_lag_1"
## [7] " + date + weekday + cases_lag_1"
## [8] "tests + date + weekday + cases_lag_1"
```

We see that `mod_content` contains the relevant model content, with some issues of additional `+` signs which will need removing. This can be done using simple regular expressions:

```r
# load packages
pacman::p_load(tidyverse)

# comine effects to generate all possible models
# note the use of 'NA' to have models with/without effects
mod_content_grid <- expand.grid(
  c(NA, "tests"),
  "date",
  c(NA, "weekday"),
  c(NA, "cases_lag_1"))

mod_content_grid
##    Var1 Var2    Var3        Var4
## 1  <NA> date    <NA>        <NA>
## 2 tests date    <NA>        <NA>
## 3  <NA> date weekday        <NA>
## 4 tests date weekday        <NA>
## 5  <NA> date    <NA> cases_lag_1
## 6 tests date    <NA> cases_lag_1
## 7  <NA> date weekday cases_lag_1
## 8 tests date weekday cases_lag_1

# Use unite() to combine all columns, with sep = " + " and na.rm = TRUE
mod_content <- mod_content_grid %>%
     unite(
          col = "models",            # name of the new united column
          1:ncol(mod_content_grid),  # columns to unite
          sep = " + ",               # separator to use in united column
          remove = TRUE,             # if TRUE, removes input cols from the data frame
          na.rm = TRUE               # if TRUE, missing values are removed before unit
     ) %>%
  pull(models) # extract column into a character vector

# Check results
mod_content
## [1] "date"
## [2] "tests + date"
## [3] "date + weekday"
## [4] "tests + date + weekday"
## [5] "date + cases_lag_1"
## [6] "tests + date + cases_lag_1"
## [7] "date + weekday + cases_lag_1"
## [8] "tests + date + weekday + cases_lag_1"
```

We now have clean model content which can be turned into *trending* models

using the approach illustrated before. In the following sections, we highlight tricks for capturing specific trends in the data, but all ultimately rely on the principle illustrated here.

4.3 Capturing periodicity

Periodic changes are often observed due to either seasonality (over long time scales for seasonal diseases such as influenza), but are also frequent in the case of COVID-19 due to reporting artifacts (Figure 4.2. For instance, some countries report less cases over a weekend, followed by a spike on the following day (*backlog* effect).

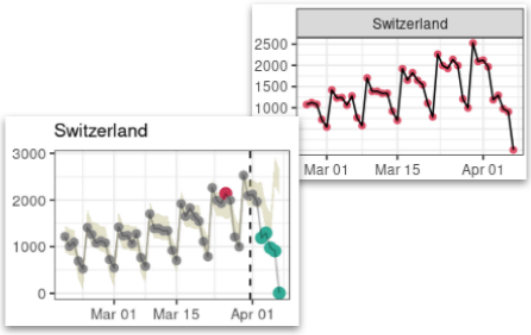


Figure 4.2: Example of periodicity in COVID-19 data. This figure illustrates a case of weekly periodicity in raw data (red dots and plain black line) captured by ASMODEE (grey dots and model envelope).

Such trends can be captured by different predictors:

1. a strict `weekend` effect, i.e. a categorical variable distinguishing weekends from weekdays
2. a `weekend` effect including a backlog effect, i.e. a categorical variable distinguishing weekends, Mondays, and other weekdays
3. a `weekday` effect, i.e. a categorical variable distinguishing each day in the week

At the time of writing, only 2 and 3 are used in the data pipelines. Option 2) is implemented by the function `day_of_week()` in the *scripts/* folder, also provided below. Option 3) is implemented in base R by `weekdays`. Both functions generate categorical variables from a `Date` input; for instance:

```
#' Convert dates to factors
#'
#' This will map a `Date' vector to weekdays, with the following
#' distinction:
```

```r
#' weekden, monday, of the rest of the week
#'
#' @author Thibaut
#'
#' @param date a vector of `Date`
#'
day_of_week <- function(date) {
  day_of_week <- weekdays(date)
  out <- vapply(
    day_of_week,
    function(x) {
      if (x %in% c("Saturday", "Sunday")) {
        "weekend"
      } else if (x == "Monday") {
        "monday"
      } else {
        "rest_of_week"
      }
    },
    character(1)
  )
  factor(out, levels = c("rest_of_week", "monday", "weekend"))
}

# generate some dates for the example
some_dates <- as.Date("2021-02-04") + 0:9
some_dates
##  [1] "2021-02-04" "2021-02-05" "2021-02-06" "2021-02-07" "2021-02-08"
##  [6] "2021-02-09" "2021-02-10" "2021-02-11" "2021-02-12" "2021-02-13"

# build new variables using dplyr
library(magrittr)
library(dplyr)
tibble(date = some_dates) %>%
  mutate(weekend = day_of_week(date), weekday = weekdays(date))
## # A tibble: 10 x 3
##    date       weekend      weekday
##    <date>     <fct>        <chr>
##  1 2021-02-04 rest_of_week Thursday
##  2 2021-02-05 rest_of_week Friday
##  3 2021-02-06 weekend      Saturday
##  4 2021-02-07 weekend      Sunday
##  5 2021-02-08 monday       Monday
##  6 2021-02-09 rest_of_week Tuesday
##  7 2021-02-10 rest_of_week Wednesday
```

```
##  8 2021-02-11 rest_of_week Thursday
##  9 2021-02-12 rest_of_week Friday
## 10 2021-02-13 weekend      Saturday
```

4.4 Capturing trend changes in the past

ASMODEE implicitly assumes that the **fitting** set can be used to capture a single trend. It frequently happens, however, that this time period actually saw a change in trend, which then cannot be captured by a simple model (e.g. Figure 4.3).

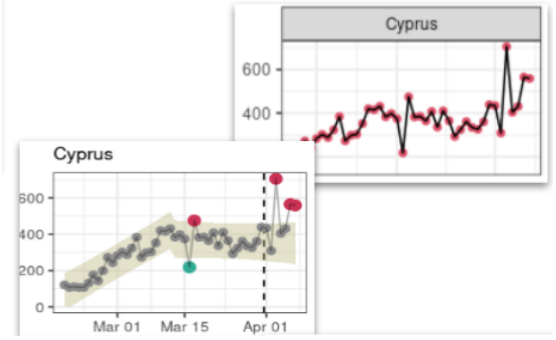


Figure 4.3: Example of trend change in COVID-19 data. This figure illustrates a case of change in temporal trends having occured during the *fitted* time period in the raw data (red dots and plain black line) captured by ASMODEE (grey dots and model envelope).

The strategy we employ to address this issue is to:

1. define a breaking point date marking the change in trend
2. building a categorical variable `period` marking dates `before` and `after`
3. using an interaction term for the effect of time (variable `date`) combined with `period` so that the model will fit a slope `before` and `after` the changing point

In practice, we often ignore how to define 1). This can be addressed by generating many models, each with a different breaking point, and leaving to ASMODEE the task to select the best fitting model.

This approach is not entirely straightforward to implement; we illustrate it below:

```
# generate some dates for the example
some_dates <- as.Date("2021-02-04") + 0:30
some_dates
```

```
##  [1] "2021-02-04" "2021-02-05" "2021-02-06" "2021-02-07" "2021-02-08"
##  [6] "2021-02-09" "2021-02-10" "2021-02-11" "2021-02-12" "2021-02-13"
## [11] "2021-02-14" "2021-02-15" "2021-02-16" "2021-02-17" "2021-02-18"
## [16] "2021-02-19" "2021-02-20" "2021-02-21" "2021-02-22" "2021-02-23"
## [21] "2021-02-24" "2021-02-25" "2021-02-26" "2021-02-27" "2021-02-28"
## [26] "2021-03-01" "2021-03-02" "2021-03-03" "2021-03-04" "2021-03-05"
## [31] "2021-03-06"

# build a dummy dataset
library(magrittr)
library(dplyr)

x <- tibble(date = some_dates) %>%
  mutate(day = as.integer(date - min(date)))

# build all changepoint variables between 10 and 20 days
min_k <- 5
max_k <- 25
k_values <- min_k:max_k
df_changepoints <- lapply(k_values,
                          function(k)
                            x %>%
                            transmute(if_else(day <= k, "before", "after")) %>%
                            pull(1)) %>%
  data.frame() %>%
  tibble() %>%
  setNames(paste0("change_", k_values))

# add changepoint variables to main data
x <- x %>%
  bind_cols(df_changepoints)
x
## # A tibble: 31 x 23
##    date         day change_5 change_6 change_7 change_8 change_9 change_10
##    <date>     <int> <chr>    <chr>    <chr>    <chr>    <chr>    <chr>
##  1 2021-02-04     0 before   before   before   before   before   before
##  2 2021-02-05     1 before   before   before   before   before   before
##  3 2021-02-06     2 before   before   before   before   before   before
##  4 2021-02-07     3 before   before   before   before   before   before
##  5 2021-02-08     4 before   before   before   before   before   before
##  6 2021-02-09     5 before   before   before   before   before   before
##  7 2021-02-10     6 after    before   before   before   before   before
##  8 2021-02-11     7 after    after    before   before   before   before
##  9 2021-02-12     8 after    after    after    before   before   before
## 10 2021-02-13     9 after    after    after    after    before   before
```

```
## # ... with 21 more rows, and 15 more variables: change_11 <chr>,
## #   change_12 <chr>, change_13 <chr>, change_14 <chr>, change_15 <chr>,
## #   change_16 <chr>, change_17 <chr>, change_18 <chr>, change_19 <chr>,
## #   change_20 <chr>, change_21 <chr>, change_22 <chr>, change_23 <chr>,
## #   change_24 <chr>, change_25 <chr>
```

Once these variables have been created, one can build the corresponding models using the same approach as before:

```
library(trending)

# step 1
mod_content <- paste("date * change", k_values, sep = "_")
mod_content
##  [1] "date * change_5"  "date * change_6"  "date * change_7"  "date * change_8"
##  [5] "date * change_9"  "date * change_10" "date * change_11" "date * change_12"
##  [9] "date * change_13" "date * change_14" "date * change_15" "date * change_16"
## [13] "date * change_17" "date * change_18" "date * change_19" "date * change_20"
## [17] "date * change_21" "date * change_22" "date * change_23" "date * change_24"
## [21] "date * change_25"

# step 2
models_txt <- sprintf("glm_model(cases ~ %s, family = poisson)", mod_content)

# step 3
models <- lapply(models_txt, function(e) eval(parse(text = e)))
class(models) # this is a list
## [1] "list"
length(models) # each component is a model
## [1] 21
models %>%
  head(4) %>% # only check for 4 models
  lapply(get_formula) # check formulas
## [[1]]
## cases ~ date * change_5
## <environment: 0x56247597e3a8>
##
## [[2]]
## cases ~ date * change_6
## <environment: 0x56247598d570>
##
## [[3]]
## cases ~ date * change_7
## <environment: 0x562475aabcd8>
##
```

```
## [[4]]
## cases ~ date * change_8
## <environment: 0x562475abc900>
```

Note that of course, changes in trend may happen in conjunction with other processes, such as periodicity (e.g. Figure 4.4). In such cases, the approach illustrated at the beginning of this chapter can be used to generate model contents with/without change and with/without periodic effect.

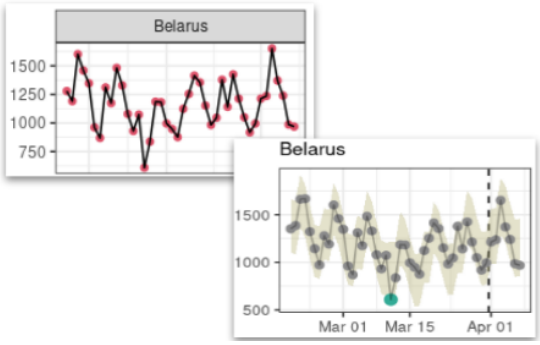


Figure 4.4: Example of trend change with periodicity in COVID-19 data. This figure illustrates a case of change in temporal trends having occured during the *fitted* time period, combined with weekly periodicity in the raw data (red dots and plain black line) captured by ASMODEE (grey dots and model envelope).

4.5 Final considerations

The approaches illustrated in this chapter should help characterise a majority of temporal trends in other disease surveillance data. In this last section, we provide additional insights into implementing ASMODEE for other data:

4.5.1 Use AIC

The original ASMODEE publication (Jombart et al., 2021) introduces different approaches for selecting the best model to characterise past trends. In this, we were suggesting that **repeated K-fold cross-validation** might lead to selecting models with better predictive power. However, we have since realised that while this approach indeed selects models with good average predictions, it ignores model variability, and might retain models which completely underestimate the variation in the data. For instance, it may retain a Poisson model over a Negative Binomial GLM, both with similar average predictions, but the Poisson having a much too narrow prediction interval, resulting in most data points being classified as outliers.

The alternative is to use **Akaike's Information Criterion** (AIC, (Akaike, 1974)). This approach is much faster, and as it tries to minimize the deviance not explained by the model, it is able to select models which better account for the variation in the data.

4.5.2 Negative Binomial: the good and the bad

In many instances, the Negative Binomial (NegBin) GLM is the most appropriate model for case counts data, as it better accounts for the variation in the data than the Poisson GLM. So in principle, one would like to use this model for most data. Unfortunately, the NegBin GLM is also prone to convergence issues, in which case it merely issues a *warning* during the fitting phase. This is especially frequent when there are zeros in the data (e.g. backlog effect).

By default, ASMODEE will ignore these models, treating them as failure (see argument `include_fitting_warnings` in `?asmodee`). We recommend keeping this behaviour, and ensuring as a 'backup' plan that all models formulated as a NegBin GLM also have at least one counterpart as another type of model, such as a Gaussian GLM or a linear regression.

4.5.3 Keep it simple

ASMODEE performs best by using many simple models as candidates, rather than a few complex ones. Indeed, complex models are prone to over-fitting, and may have poor predictive value, so that they will not be useful to identify outliers in the recent days. In this infrastructure, the most complex model would be that of an exponential growth/decline (1 parameter) with a change point (2 parameters), and effect of testing (1 parameter), and weekly periodicity with a different offset for each day of the week (6 parameters). As our *fitting* dataset contains 6 weeks of data (42 data points), the most complex model still has 32 degrees of freedom, which means we are unlikely to over-fit the data.

It is also important to ensure that at least one model will work in any case. When analysing a range of locations (e.g. countries in this infrastructure), ASMODEE will attempt to fit all candidate models to a given country, and retain the best fitting one, ignoring models which errored or issued warnings. However, ASMODEE will generate an error if not a single model could be fitted to a given country. To avoid this situation, it is best to make sure at least one model will always work. This can be achieved by using a simple, constant model, e.g. by including the one of the following in the candidate models:

```
library(trending)
cst_lm <- lm_model(cases ~ 1)
cst_gaussian <- glm_model(cases ~ 1, family = gaussian)
```

Chapter 5

Automation

There are many ways to automate data infrastructures. Here, we use github actions, which seem like a natural choice given that the whole infrastructure is hosted on github. However, note that alternatives could be considered especially for group owning a dedicated server which they can administrate at will - this will likely be a more robust solution for achieving automation. In particular, it would make troubleshooting much easier, as github actions do not let the user directly interact with the server running the jobs.

In this chapter, we provide a few insights into how github actions have been implemented.

5.1 Overview of the github actions

Github actions permit to run parts of data pipelines automatically, as summarised in figure 2.1. The results of github action runs as well as the corresponding files are available on github under the 'actions' tab. In particular, this page shows the current status of the different actions:

- running (yellow): the job is currently running
- passed (green): the last job ran successfully
- failed (red): the last job errored

In every case, logs are available. These are particularly useful for troubleshooting issues.

Each github action is defined by a YAML file stored in the hidden folder *.github/workflows/*, currently containing:

```
## [1] "auto_synthesis.yml"    "auto_update_afro.yml"  "auto_update_all.yml"
## [4] "auto_update_emro.yml"  "auto_update_euro.yml"  "auto_update_paho.yml"
## [7] "auto_update_searo.yml" "auto_update_wpro.yml"  "refresh_readme.yml"
```

- *auto_update_[region].yml*: analysis updates for a given region
- *auto_update_all.yml*: same, but for all regions
- *auto_synthesis.yml*: collects existing results for all regions, generate ELR report and send information products by email to a list of recipients
- *refresh_readme.yml*: update the *README.md* file

Sub-sections below provide more information on these actions.

5.1.1 Regional updates

5.1.1.1 Outline

This set of actions (one per WHO region) are implemented in the *auto_update_[region].yml* files. They automate the following workflow:

- download and prepare the current data
- run all analyses for the region under consideration
- commit and push to github the resulting information products (i.e. the *asmodee_outputs/* folder) - see details on the regional analyses in the report chapter 3

note: Neither the data nor the compiled report (usually stored in the *outputs/* folder) are commited or pushed to github.

5.1.1.2 Schedule

These actions run every day at 12:30, 14:30, and 18:30 GMT. It can also be triggered manually from the github action page (see figure @(fig:action-trigger) below.

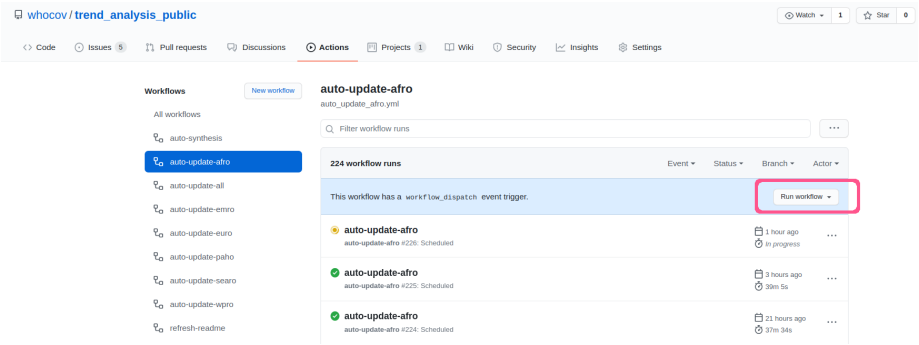


Figure 5.1: Manual trigger of actions from the github action page.

5.1.2 Updates of all regions

5.1.2.1 Outline

This action is implemented in the *auto_update_all.yml* and automates the same workflow as the regional updates, but for all regions.

5.1.2.2 Schedule

This action runs every day at 16:00 and 22:00 GMT. It can also be triggered manually from the github action page (see figure 5.1. As an option, the user can specify a single WHO region (using the upper case abbreviations) when manually triggering a run of this action.

5.1.3 ELR synthesis

5.1.3.1 Outline

This action is implemented in the *auto_synthesis.yml* and automates the following workflow:

- compile results of all regions using the latest analyses on github from the *asmodee_outputs/* folder
- generates an ELR report - see details on the ELR synthesis in the report chapter 3
- creates a zip archive of:
  - the ELR report as a self-contained *html* file
  - an *xlsx* spreadsheet containing ELRs for all countries alongside key indicators
  - a text file listing countries excluded from the analysis as well as the rationale for their exclusion
- sends an email to a list of recipients hard-coded in the script *run_synthesis.R* (located in the root folder) with this zip attached

note: this job does not commit anything to github; the only end-result persisting after the job has run is the email and its attachments.

5.1.3.2 Schedule

This action runs every Monday at 6:00 GMT. It can also be triggered manually from the github action page (see figure 5.1.

5.1.3.3 Automated emails

Automated emails, as currently implemented, require SMTP authentication for the free SMTP provider *sendinblue*. This is currently done through Thibaut Jombart's account, linked to the gmail address *thibautjombart@gmail.com*. While emails will look like they were sent from this email, they were actually sent by a different provider, which will often result in these emails being marked

as SPAM. Double-check your SPAM folder if you are on the list of recipients and seem to have missed the email.

5.1.4 README updates

5.1.4.1 Outline

This action is implemented in the *refresh_readme.yml* and ensures that the *README.md* displaying results on the landing page of the github project is updated every time new results are generated. The README is updated by recompiling the file *README.Rmd*.

5.1.4.2 Schedule

This action runs every time there is a push to the main branch resulting in a change of files, excluding:

- *README.md*
- *run_synthesis.R*
- *report_sources/elr_review.Rmd*

5.2 Using secrets

5.2.1 Github secrets in a nutshell

The automation process may require inputs such as passwords and authentication tokens which cannot be stored on a public github repository for obvious security reasons. Github actions provide a workaround for such issues through github secrets. Secrets are essentially small bits of data stored encrypted on github, which are tied to a specific repository (figure 5.2). Once set up, a secret cannot be viewed again, but it can be modified or deleted.

5.2.2 Using secrets in R workflows

The data stored in a secret can be used inside *github actions* by referring the their value as `${{ secrets.SECRET_NAME }}` where `SECRET_NAME` is the name of the secret. The approach to use this in an R workflow is then:

1. set the secret value (e.g. a password) as an environment variable
2. retrieve the value of the environment variable from R via `Sys.getenv()`
3. use the value in the R workflow, making sure that it is never output to the console, a file, or a persistent object, as these could then become public (e.g. in logs of actions)

At the time of writing, secrets are used for two purposes:

- to store the **phifunc_token** required to install the *phifunc* package; this enables github actions to use *phifunc* to download and process data

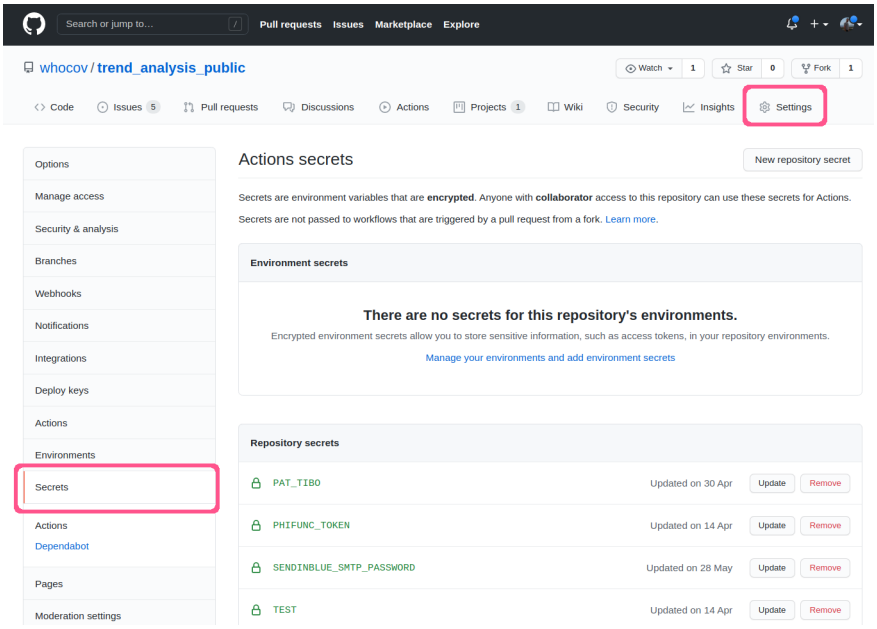
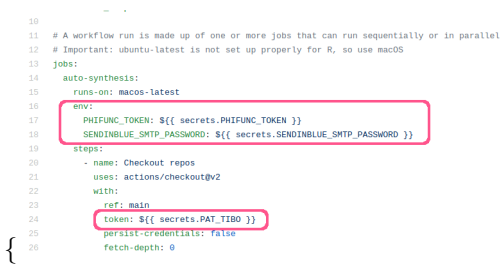


Figure 5.2: Github secret page. This screenshot shows the location of github secrets on the infrastructure's page, underthe *Settings* tab. New secrets can be added there, and old secrets can be modified, but not visualised.

- to store the **SMTP password** used to send automated emails

Outside of R workflows, we also use another secret, a **Personal Authentication Token** (**PAT**), to enable pushing to different branches of the github repository. Figure 5.2.2 shows how secrets can be used in the configuration file of a github action.

\begin{figure}



{

}

\caption{Sample of the *auto-synthesis* github action using github secrets. This screenshot was taken from the github action configation file *.github/workflows/auto_synthesis.yml*. Secrets are outlined in red. The first two (*PHIFUNC_TOKEN* and *SENDINBLUE_SMTP_PASSWORD*) are set up as environment variables, while the last one (*PAT_TIBO*) is passed directly as an input to the *checkout* action.} \end{figure}

Note that in the case of *phifunc_token*, we need to handle a non-trivial situation as the token could be available in two different ways:

- **stored locally** in the root folder, in the case of a person's computer who was granted access and given the token file; this file is git-ignored and so should never end up on github
- available as an **environment variable**, in the case of github actions

This explains the circumvoluted code handling the installation of *phifunc* in *scripts/remote_packages.R*:

```
# Here we handle the phifunc install token as follows, by order of decreasing
# priority:

## 1. Grab the token from a local 'phifunc_token' file if it exists
## 2. Grab the token the environment variable 'PHIFUNC_TOKEN'
## 3. If the above is empty, set the value to NULL

phifunc_token_file <- here::here("phifunc_token")
if (file.exists(phifunc_token_file)) {
  phifunc_token <- scan(phifunc_token_file, what = "character")
} else {
```

```
  phifunc_token <- Sys.getenv("PHIFUNC_TOKEN")
  if (phifunc_token == "") phifunc_token <- NULL
}

# Install phifunc
if (!is.null(phifunc_token) & !require("phifunc")) {
  remotes::install_github(
    "whocov/phifunc",
    auth_token = phifunc_token,
    subdir = "phifunc", upgrade = "never")
}
```

Bibliography

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Trans. Automat. Contr.*, 19(6):716–723.

Jombart, T., Ghozzi, S., Schumacher, D., Taylor, T. J., Leclerc, Q. J., Jit, M., Flasche, S., Greaves, F., Ward, T., Eggo, R. M., Nightingale, E., Meakin, S., Brady, O. J., Centre for Mathematical Modelling of Infectious Diseases COVID-19 Working Group, Medley, G. F., Höhle, M., and Edmunds, W. J. (2021). Real-time monitoring of COVID-19 dynamics using automated trend fitting and anomaly detection. *Philos. Trans. R. Soc. Lond. B Biol. Sci.*, 376(1829):20200266.

Schumacher, D. and Jombart, T. (2021a). *trendbreaker: Detect Changes in Temporal Trends*. R package version 0.0.4.

Schumacher, D. and Jombart, T. (2021b). *trending: Model Temporal Trends*. R package version 0.0.3.