Sykdomspulsen Core in Depth

members of the Sykdomspulsen Team

Contents

Sykdomspulsen Core in Depth **5**

Authors **7**

1 DB Schemas **9**
1.1 Introduction . . . 9
1.2 Database servers . . . 9
1.3 Access level (anon/restr/redirect) . . . 9
1.4 Creating your own . . . 10
1.5 Loading data into a db schema . . . 14
1.6 Accessing the data in a db schema . . . 15
1.7 Accessing the data in ad-hoc analyses . . . 18
1.8 Exploring data in schemas . . . 18

2 Tasks **23**
2.1 Introduction . . . 23
2.2 Definitions . . . 23
2.3 General tasks . . . 24
2.4 Putting it together . . . 25
2.5 Weather example . . . 27
2.6 action_fn . . . 34
2.7 Run the task . . . 38
2.8 Examples of different types of tasks . . . 39

3 File Layout **43**
3.1 Introduction . . . 43
3.2 00_env_and_namespace.r . . . 43
3.3 01_definitions.r . . . 45
3.4 02_permissions.r . . . 46
3.5 03_db_schemas.r . . . 47
3.6 04_tasks.r . . . 49
3.7 05_deliverables.r . . . 52
3.8 06_config.r . . . 52

3.9 07_onLoad.r 53
3.10 08_onAttach.r 54
3.11 99_util_*.r 54
3.12 Task files 55

4 Tutorial 1: Introduction **67**
4.1 Setup 67
4.2 Load the code 67
4.3 Weather data example 68
4.4 Developing weather_download_and_import_rawdata 69
4.5 Developing weather_clean_data 88
4.6 Developing weather_export_plots 104
4.7 Final package 110
4.8 What now? 111

Sykdomspulsen Core in Depth

Sykdomspulsen Core is the free and open-source backbone of Sykdomspulsen.

Sykdomspulsen Core is a standalone R package, which means it is easy for other teams to build up their own surveillance infrastructure based on Sykdomspulsen Core.

Authors

Written by Richard Aubrey White and Aurora Christine Hofman.

Chapter 1

DB Schemas

1.1 Introduction

A database schema is our way of representing how the database is constructed. In short, you can think of these as database tables.

1.2 Database servers

Normally, an implementation of Sykdomspulsen Core would have two database servers that run parallel systems. One database server is `auto` and the other is `interactive`.

If you run code in RStudio Workbench or on Airflow interactive, you should be automatically be connected to the interactive database server. If you run code on Airflow auto, you should be automatically be connected to the auto database server. This is something that your implementation will have to solve.

1.3 Access level (anon/restr/redirect)

Within each database server, there are multiple databases with different access levels and censoring requirements.

Censoring is performed via the db schema.

1.3.1 anon

The "anonymous" database contains data that is anonymous. All team members should have access to this database.

1.3.2 restr

The “restricted” database contains data that is:

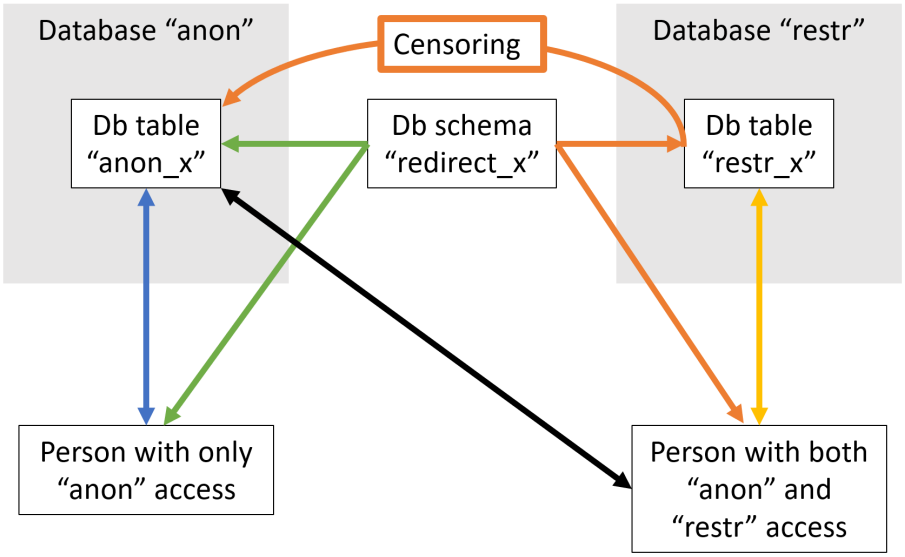
- Indirectly identifiable
- Anonymous

Only a restricted number of team members should have access to this database.

1.3.3 redirect

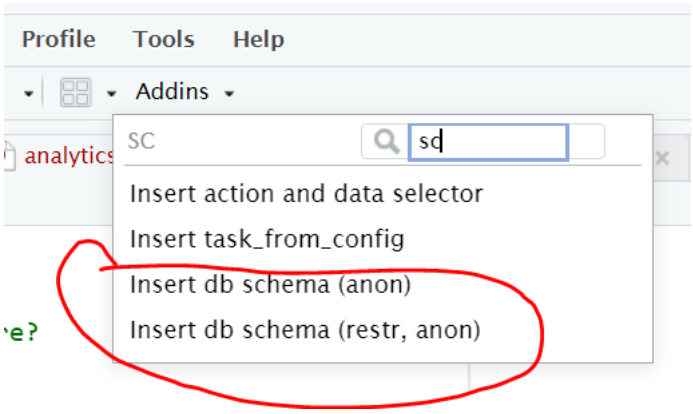
This is not technically a database, however, it is treated as one.

If a person creates a db schema that exists in both the anonymous and restricted databases, then Sykdomspulsen Core will automatically detect the highest level of access and connect to that database when working with redirect schemas.



1.4 Creating your own

Sykdomspulsen Core requires a lot of boilerplate code. It is strongly recommended that you use the RStudio `Addins` menu to help you quickly insert code templates.



We will generate three database schemas:

- restr_example (specified via `name_access`)
- anon_example (specified via `name_access`)
- redirect_example (automatically created when both `restr` and `anon` are used)

```
sc::add_schema_v8(
  name_access = c("restr", "anon"),
  name_grouping = "example",
  name_variant = NULL,
  db_configs = sc::config$db_configs,
  field_types =  c(
    "granularity_time" = "TEXT",
    "granularity_geo" = "TEXT",
    "country_iso3" = "TEXT",
    "location_code" = "TEXT",
    "border" = "INTEGER",
    "age" = "TEXT",
    "sex" = "TEXT",

    "date" = "DATE",

    "isoyear" = "INTEGER",
    "isoweek" = "INTEGER",
    "isoyearweek" = "TEXT",
    "season" = "TEXT",
    "seasonweek" = "DOUBLE",

    "calyear" = "INTEGER",
    "calmonth" = "INTEGER",
    "calyearmonth" = "TEXT",
```

```
    "value_n" = "INTEGER"
  ),
  keys = c(
    "granularity_time",
    "location_code",
    "date",
    "age",
    "sex"
  ),
  censors = list(
    restr = list(
      value_n = sc::censor_function_factory_nothing("value_n")
    ),
    anon = list(
      value_n = sc::censor_function_factory_values_0_4("value_n")
    )
  ),
  validator_field_types = sc::validator_field_types_sykdomspulsen,
  validator_field_contents = sc::validator_field_contents_sykdomspulsen,
  info = "This db table is used for..."
)
```

This `schema` has a few main parts.

1.4.1 Naming

The db schemas and tables will be given the names: `name_access_name_grouping_name_variant`

In this example, there will be three db schemas:

- restr_example (accessible at `sc::config$schemas$restr_example`)
- anon_example (accessible at `sc::config$schemas$anon_example`)
- redirect_example (accessible at `sc::config$schemas$redirect_example`)

Corresponding to two db tables:

- restr_example
- anon_example

1.4.1.1 name_access

Either `restr` or `anon`

1.4.1.2 name_grouping

A descriptive name

1.4.1.3 name_variant

A descriptive name

1.4.2 db_configs

A list that contains information about the database:

```
names(sc::config$db_configs)
## [1] "restr"         "anon"          "specific_daar" "config"
```

1.4.3 db_field_types

A vector containing the names and variable types of the columns of the database table.

In the vast majority of cases, the first 16 columns are standardized and will always be the same.

Permitted variable types are:

- TEXT
- DOUBLE
- INTEGER
- BOOLEAN
- DATE
- DATETIME

1.4.4 keys

The columns that will form the primary key of the database table (i.e. identify unique rows).

1.4.5 censors

1.4.6 validator_field_types

A validator that is useful for ensuring that your database table names are consistent with predetermined rules. For example, in Sykdomspulsen we have decided that we always want the first 16 columns to be:

- granularity_time
- granularity_geo
- country_iso3
- location_code
- border
- age
- sex
- date

- isoyear
- isoweek
- isoyearweek
- season
- seasonweek
- calyear
- calmonth
- calyearmonth

While developing new code we found that it was difficult to force all developers to remember to include these 16 columns in the correct order. The validator `sc::validator_field_types_sykdomspulsen` ensures that the first 16 columns are as expected, and otherwise the developer will not be able to run their code.

`validator_field_contents` is a validator that ensures that the contents of your data is correct. We experienced that there were issues with `granularity_time` sometimes containing the value `week` and sometimes containing the value `weekly`. To maintain consistency in our data, the validator `sc::validator_field_contents_sykdomspulsen` will throw an error if it observes non-accepted values for certain variables.

1.5 Loading data into a db schema

Checklist:

1. Remember that "keys" (as defined in sc::add_schema_v8) defines the uniquely identifying rows of data that are allowed in the db table
2. Use `sc::fill_in_missing_v8(d)`
3. Choose your method of loading the data (upsert/insert/drop_all_rows_and_then_upsert_data)

We check to see what schemas are available:

```
stringr::str_subset(names(sc::config$schemas), "_example$")
## [1] "restr_example"    "anon_example"     "redirect_example"
```

We then create a fictional dataset and work with it.

Remember that "keys" (as defined in `sc::add_schema_v8`) defines the uniquely identifying rows of data that are allowed in the db table!

```
options(width = 150)
# fictional dataset
d <- data.table(
  granularity_time = "day",
  granularity_geo = "nation",
  country_iso3 = "nor",
  location_code = "norge",
  border = 2020,
  age = "total",
```

```
  sex = "total",

  date = c(as.Date("1990-01-07"),as.Date("1990-01-08")),

  isoyear = 1990,
  isoweek = 1,
  isoyearweek = "1990-01",
  season = "1990/1991",
  seasonweek = 24,

  calyear = NA,
  calmonth = NA,
  calyearmonth = NA,

  value_n = c(3,6)
)

# display the raw data
d[]
##    granularity_time granularity_geo country_iso3 location_code border   age   sex       date i
## 1:              day          nation          nor         norge   2020 total total 1990-01-07
## 2:              day          nation          nor         norge   2020 total total 1990-01-08
##    calmonth calyearmonth value_n
## 1:       NA           NA       3
## 2:       NA           NA       6

# always fill in missing data!
sc::fill_in_missing_v8(d)

# we have four options to get the data into the db table
# remember that "keys" defines the uniquely identifying rows of data that are allowed in the db t
# - upsert means "update if data exists, otherwise append"
# - insert means "append" (data cannot already exist)

sc::config$schemas$redirect_example$upsert_data(d)
## Creating table restr_example
## Creating table anon_example
#sc::config$schemas$redirect_example$insert_data(d)
#sc::config$schemas$redirect_example$drop_all_rows_and_then_upsert_data(d)
#sc::config$schemas$redirect_example$drop_all_rows_and_then_insert_data(d)
```

1.6 Accessing the data in a db schema

Checklist:

1. `sc::mandatory_db_filter`
2. `dplyr::select`

We extract data from db schemas using dplyr with a dbplyr backend.

```
options(width = 150)
sc::config$schemas$redirect_example$tbl() %>%
  sc::mandatory_db_filter(
    granularity_time = "day",
    granularity_time_not = NULL,
    granularity_geo = NULL,
    granularity_geo_not = NULL,
    country_iso3 = NULL,
    location_code = "norge",
    age = "total",
    age_not = NULL,
    sex = "total",
    sex_not = NULL
  ) %>%
  dplyr::select(
    granularity_time,
    location_code,
    date,
    value_n,
    value_n_censored
  ) %>%
  dplyr::collect() %>%
  as.data.table() %>%
  print()
##    granularity_time location_code       date value_n value_n_censored
## 1:              day         norge 1990-01-07       3            FALSE
## 2:              day         norge 1990-01-08       6            FALSE
```

We can observe the effects of censoring as defined in `sc::add_schema_v8`

```
options(width = 150)
sc::config$schemas$restr_example$tbl() %>%
  sc::mandatory_db_filter(
    granularity_time = "day",
    granularity_time_not = NULL,
    granularity_geo = NULL,
    granularity_geo_not = NULL,
    country_iso3 = NULL,
    location_code = "norge",
    age = "total",
    age_not = NULL,
    sex = "total",
```

```
    sex_not = NULL
  ) %>%
  dplyr::select(
    granularity_time,
    location_code,
    date,
    value_n,
    value_n_censored
  ) %>%
  dplyr::collect() %>%
  as.data.table() %>%
  print()
##    granularity_time location_code       date value_n value_n_censored
## 1:              day         norge 1990-01-07       3            FALSE
## 2:              day         norge 1990-01-08       6            FALSE

sc::config$schemas$anon_example$tbl() %>%
  sc::mandatory_db_filter(
    granularity_time = "day",
    granularity_time_not = NULL,
    granularity_geo = NULL,
    granularity_geo_not = NULL,
    country_iso3 = NULL,
    location_code = "norge",
    age = "total",
    age_not = NULL,
    sex = "total",
    sex_not = NULL
  ) %>%
  dplyr::select(
    granularity_time,
    location_code,
    date,
    value_n,
    value_n_censored
  ) %>%
  dplyr::collect() %>%
  as.data.table() %>%
  print()
##    granularity_time location_code       date value_n value_n_censored
## 1:              day         norge 1990-01-07       0             TRUE
## 2:              day         norge 1990-01-08       6            FALSE
```

1.7 Accessing the data in ad-hoc analyses

When doing ad-hoc analyses, you may access the database tables via the helper function `sc::tbl`

IT IS STRICTLY FORBIDDEN TO USE THIS INSIDE SYKDOMSPULSEN TASKS!!!

This is because `sc::tbl`:

- is NOT SAFE to use in parallel programming
- bypasses the input/output control mechanisms that we apply in `sc::task_from_config_v8`

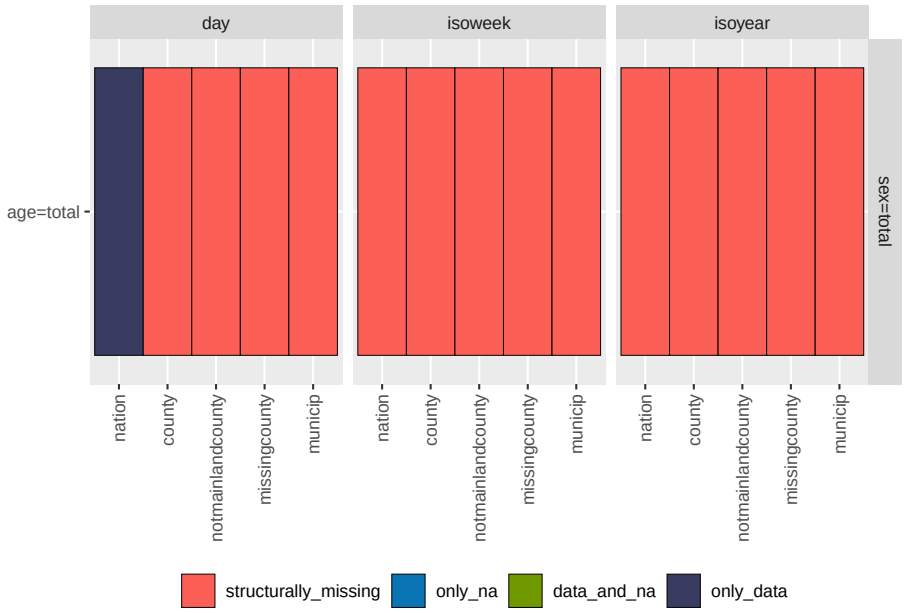
```
options(width = 150)
sc::tbl("restr_example") %>%
  sc::mandatory_db_filter(
    granularity_time = "day",
    granularity_time_not = NULL,
    granularity_geo = NULL,
    granularity_geo_not = NULL,
    country_iso3 = NULL,
    location_code = "norge",
    age = "total",
    age_not = NULL,
    sex = "total",
    sex_not = NULL
  ) %>%
  dplyr::select(
    granularity_time,
    location_code,
    date,
    value_n,
    value_n_censored
  ) %>%
  dplyr::collect() %>%
  as.data.table() %>%
  print()
##    granularity_time location_code       date value_n value_n_censored
## 1:              day         norge 1990-01-07       3            FALSE
## 2:              day         norge 1990-01-08       6            FALSE
```

1.8 Exploring data in schemas

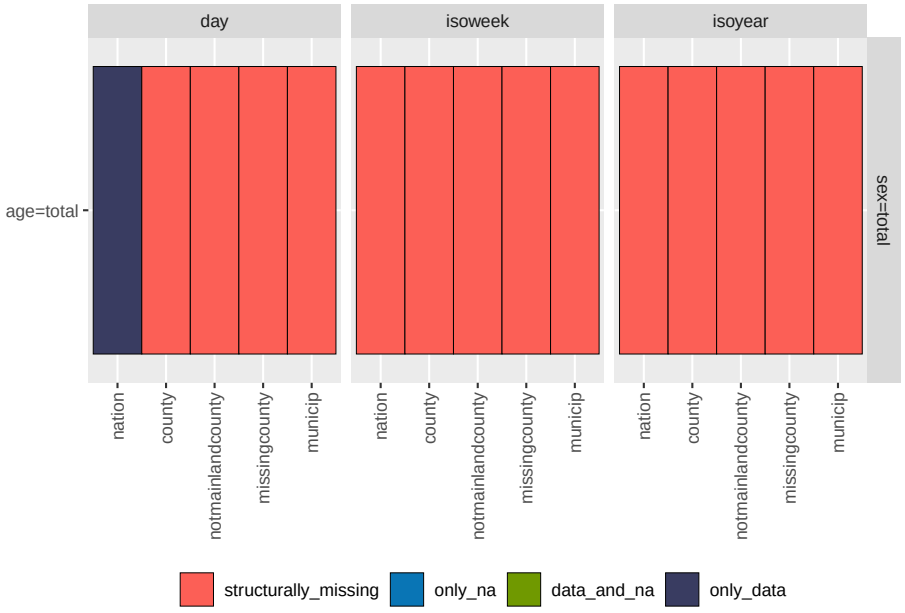
DB Schemas obviously contain a lot of data. It can be very overwhelming to try and understand what is inside the schema.

```r
options(width = 150)

# Get the first few lines of the schema (use $tbl())
sc::config$schemas$restr_example$tbl()
## # Source:   table<restr_example> [?? x 18]
## # Database: Microsoft SQL Server 12.00.6433[FHI\RIWH@OFY-GN-SQL01/sykdomspulsen_interactive_re
##   granularity_time granularity_geo country_iso3 location_code border age   sex   date       is
##   <chr>            <chr>           <chr>        <chr>          <int> <chr> <chr> <date>
## 1 day              nation          nor          norge           2020 total total 1990-01-07
## 2 day              nation          nor          norge           2020 total total 1990-01-08
## # ... with 4 more variables: calmonth <int>, calyearmonth <chr>, value_n <int>, value_n_censor

# Get a summary of the schema (referencing the schema directly)
sc::config$schemas$restr_example
## [sykdomspulsen_interactive_restr].[dbo].[restr_example]   (connected)
##
## granularity_time (TEXT):
##  - day (n = 2)
## granularity_geo (TEXT):
##  - nation (n = 2)
## country_iso3 (TEXT):
##  - nor (n = 2)
## location_code (TEXT)
## border (INTEGER):
##  - 2020 (n = 2)
## age (TEXT):
##  - total (n = 2)
## sex (TEXT):
##  - total (n = 2)
## date (DATE)
## isoyear (INTEGER):
##  - 1990 (n = 2)
## isoweek (INTEGER)
## isoyearweek (TEXT)
## season (TEXT):
##  - 1990/1991 (n = 2)
## seasonweek (DOUBLE)
## calyear (INTEGER)
## calmonth (INTEGER)
## calyearmonth (TEXT)
## value_n (INTEGER)
## value_n_censored (BOOLEAN)

# Get a summary of a variable inside the schema via 'hashing the data structure'
sc::config$schemas$restr_example %>%
```

```
  spltidy::hash_data_structure("value_n") %>%
  plot()
```




```
# This can also be done directly on a dbplyr table
sc::tbl("restr_example") %>%
  spltidy::hash_data_structure("value_n") %>%
  plot()
```


day
isoweek
isoyear
age=total
sex=total
nation
county
notmainlandcounty
missingcounty
municip
structurally_missing
only_na
data_and_na
only_data

Chapter 2

Tasks

2.1 Introduction

A task is the basic operational unit of Sykdomspulsen Core. It is based on plnr.

In short, you can think of a Sykdomspulsen Core task as multiple plnr plans plus Sykdomspulsen Core db schemas.

2.2 Definitions

Object

Description

argset

A named list containing arguments.

plnr analysis

These are the fundamental units that are scheduled in plnr:

1 argset

1 function that takes two (or more) arguments:

data (named list)

argset (named list)

... (optional arguments)

data_selector_fn

A function that takes two arguments:

argset (named list)

schema (named list)

This function provides a named list to be used as the `data` argument to `action_fn`

action_fn

A function that takes three arguments:

data (named list, returned from data_selector_fn)

argset (named list)

schema (named list)

This is the thing that **'does stuff'** in Sykdomspulsen Core.

sc analysis

A `sc analysis` is essentially a `plnr analysis` with database schemas:

1 argset

1 action_fn

plan

1 data-pull (using data_selector_fn)

1 list of sc analyses

task

This is is the unit that Airflow schedules.

1 list of plans

We sometimes run the list of plans in parallel.

2.3 General tasks

Figure 2.1 shows us the full potential of a task.

Data can be read from any sources, then within a plan the data will be extracted **once** by `data_selector_fn` (i.e. "one data-pull"). The data will then be provided to each analysis, which will run `action_fn` on:

- The provided data
- The provided argset
- The provided schemas

The `action_fn` can then:

- Write data/results to db schemas
- Send emails

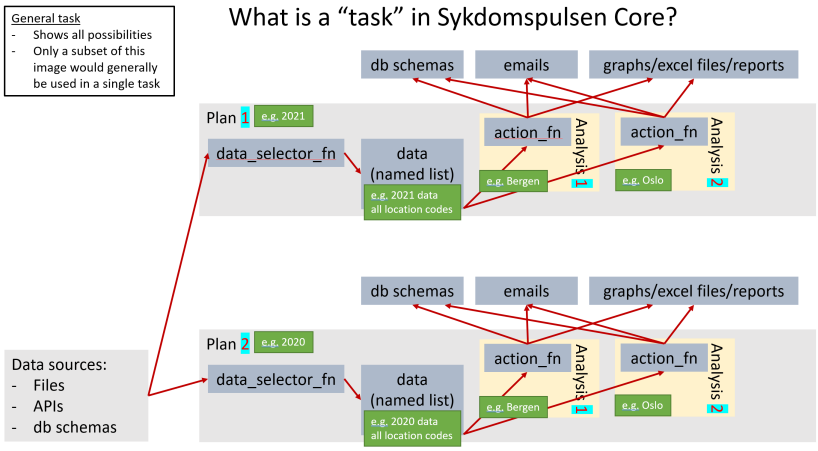


Figure 2.1: A general task showing the many options of a task.

- Export graphs, excel files, reports, or other physical files

Typically only a subset of this would be done in a single task.

2.3.1 Plan-heavy or analysis-heavy tasks?

A plan-heavy task is one that has many plans and a few analyses per plan.

An analysis-heavy task is one that has few plans and many analyses per plan.

In general, a data-pull is slow and wastes time. This means that it is preferable to reduce the number of data-pulls performed by having each data-pull extract larger quantities of data. The analysis can then subset the data as required (identifed via argsets). i.e. If possible, an analysis-heavy task is preferable because it will be faster (at the cost of needing more RAM).

Obviously, if a plan's data-pull is larger, it will use more RAM. If you need to conserve RAM, then you should use a plan-heavy approach.

Figure 2.1 shows only 2 location based analyses, but in reality there are 356 municipalities in Norway in 2021. If figure 2.1 had 2 plans (1 for 2021 data, 1 for 2020 data) and 356 analyses for each plan (1 for each location_code) then we would be taking an analysis-heavy approach.

2.4 Putting it together

Figure 2.2 shows a typical implementation of Sykdomspulsen Core.

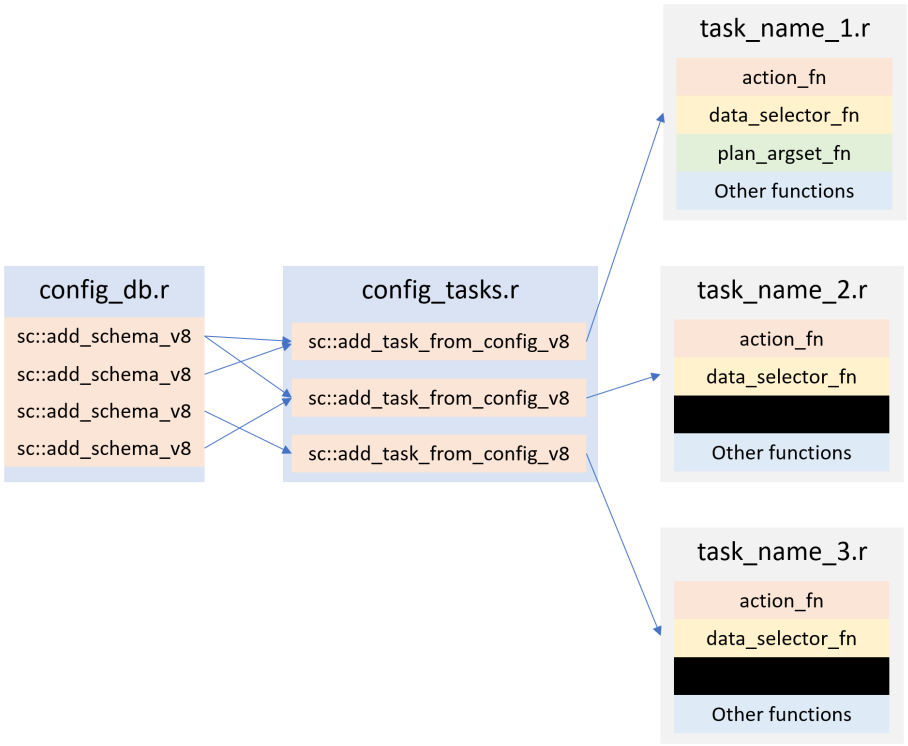


Figure 2.2: A typical file setup for an implementation of Sykdomspulsen Core. $plan_argset_fn$ is rarely used, and is therefore shown as blacked out in the most of the tasks.

`config_db.r` contains all of the Sykdomspulsen Core db schemas definitions. i.e. A long list of `sc::add_schema_v8` commands.

`config_tasks.r` contains all of the task definitions. i.e. A long list of `sc::add_task_from_config_v8` commands.

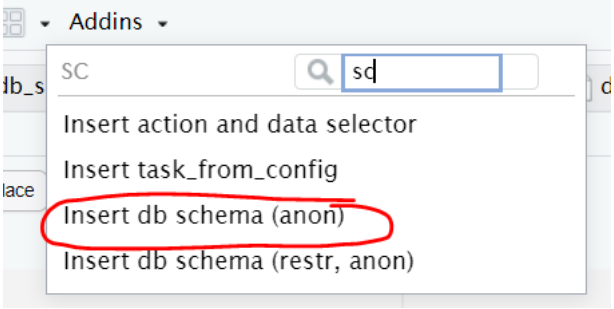
Then we have a one file for each task that contains the `action_fn`, `data_selector_fn` and other functions that are relevant to the task at hand.

2.5 Weather example

We will now go through an example of how a person would design and implement tasks relating to weather

2.5.1 db schema

As documented in more detail here, we create a db schema that fits our needs (recording weather data).

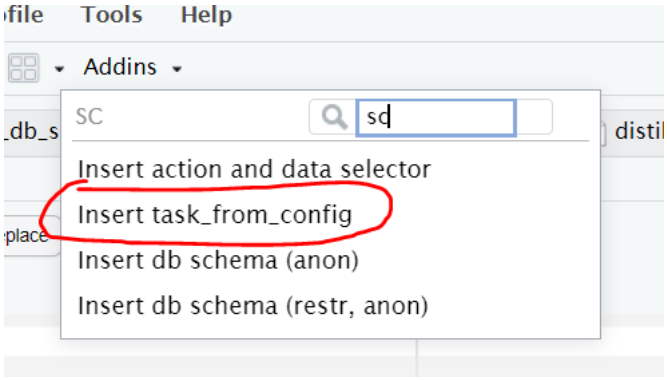



```
sc::add_schema_v8(
  name_access = c("anon"),
  name_grouping = "example_weather",
  name_variant = NULL,
  db_configs = sc::config$db_configs,
  field_types =  c(
    "granularity_time" = "TEXT",
    "granularity_geo" = "TEXT",
    "country_iso3" = "TEXT",
    "location_code" = "TEXT",
    "border" = "INTEGER",
    "age" = "TEXT",
    "sex" = "TEXT",

    "date" = "DATE",

    "isoyear" = "INTEGER",
```

```
    "isoweek" = "INTEGER",
    "isoyearweek" = "TEXT",
    "season" = "TEXT",
    "seasonweek" = "DOUBLE",

    "calyear" = "INTEGER",
    "calmonth" = "INTEGER",
    "calyearmonth" = "TEXT",

    "tg" = "DOUBLE",
    "tx" = "DOUBLE",
    "tn" = "DOUBLE",
    "rr" = "DOUBLE"
  ),
  keys = c(
    "granularity_time",
    "location_code",
    "date",
    "age",
    "sex"
  ),
  censors = list(
    anon = list(

    )
  ),
  validator_field_types = sc::validator_field_types_sykdomspulsen,
  validator_field_contents = sc::validator_field_contents_sykdomspulsen,
  info = "This db table is used for..."
)
```

2.5.2 task_from_config_v8

To "register" our task, we use the RStudio addin `task_from_config`.




```r
# tm_run_task("example_weather_import_data_from_api")
sc::add_task_from_config_v8(
  name_grouping = "example_weather",
  name_action = "import_data_from_api",
  name_variant = NULL,
  cores = 1,
  plan_analysis_fn_name = NULL, # "PACKAGE::TASK_NAME_plan_analysis"
  for_each_plan = plnr::expand_list(
    location_code = "county03" # fhidata::norway_locations_names()[granularity_geo %in% c("county
  ),
  for_each_analysis = NULL,
  universal_argset = NULL,
  upsert_at_end_of_each_plan = FALSE,
  insert_at_end_of_each_plan = FALSE,
  action_fn_name = "example_weather_import_data_from_api_action",
  data_selector_fn_name = "example_weather_import_data_from_api_data_selector",
  schema = list(
    # input

    # output
    "anon_example_weather" = sc::config$schemas$anon_example_weather
  ),
  info = "This task does..."
)
```

There are a number of important things in this code that need highlighting.

2.5.2.1 for_each_plan

`for_each_plan` expects a list. Each component of the list will correspond to a plan, with the values added to the argset of all the analyses inside the plan.

For example, the following code would give 4 plans, with 1 analysis per each plan, with each analysis containing `argset$var_1` and `argset$var_2` as appropriate.

```
for_each_plan <- list()
for_each_plan[[1]] <- list(
  var_1 = 1,
  var_2 = "a"
)
for_each_plan[[2]] <- list(
  var_1 = 2,
  var_2 = "b"
)
for_each_plan[[3]] <- list(
  var_1 = 1,
  var_2 = "a"
)
for_each_plan[[4]] <- list(
  var_1 = 2,
  var_2 = "b"
)
```

You **always** need at least 1 plan. The most simple plan possible is:

```
plnr::expand_list(
  x = 1
)
## [[1]]
## [[1]]$x
## [1] 1
```

2.5.2.2 plnr::expand_list

`plnr::expand_list` is esentially the same as `expand.grid`, except that its return values are lists instead of data.frame.

The code above could be simplified as follows.

```
for_each_plan <- plnr::expand_list(
  var_1 = c(1,2),
  var_2 = c("a", "b")
)
for_each_plan
## [[1]]
## [[1]]$var_1
## [1] 1
##
## [[1]]$var_2
## [1] "a"
##
##
```

```
## [[2]]
## [[2]]$var_1
## [1] 2
##
## [[2]]$var_2
## [1] "a"
##
##
## [[3]]
## [[3]]$var_1
## [1] 1
##
## [[3]]$var_2
## [1] "b"
##
##
## [[4]]
## [[4]]$var_1
## [1] 2
##
## [[4]]$var_2
## [1] "b"
```

2.5.2.3 for_each_analysis

`for_each_plan` expects a list, which will generate `length(for_each_plan)` plans.

`for_each_analysis` is the same, except it will generate **analyses** within each of the plans.

2.5.2.4 universal_argset

A named list that will add the values to the argset of all the analyses.

2.5.2.5 upsert_at_end_of_each_plan

If `TRUE` and `schema` contains a schema called `output`, then the returned values of `action_fn` will be stored and upserted to `schema$output` at the end of each **plan**.

If you choose to upsert/insert manually from within `action_fn`, you can only do so at the end of each **analysis**.

2.5.2.6 insert_at_end_of_each_plan

If `TRUE` and `schema` contains a schema called `output`, then the returned values of `action_fn` will be stored and inserted to `schema$output` at the end of each **plan**.

If you choose to upsert/insert manually from within `action_fn`, you can only do so at the end of each **analysis**.

2.5.2.7 action_fn_name

A character string of the action_fn, preferably including the package name.

2.5.2.8 data_selector_fn_name

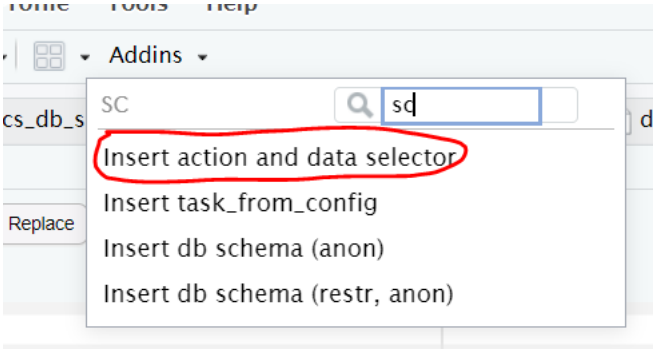
A character string of the data_selector_fn, preferably including the package name.

2.5.2.9 schema

A named list containing the schemas used in this task.

2.5.3 data_selector_fn

Use the addins dropdown to easily add in boilerplate code.



The `data_selector_fn` is used to extract the data for each plan.

The lines inside `if(plnr::is_run_directly()){` are used to help developers. You can run the code manually/interactively to "load" the values of `argset` and `schema`.

```
index_plan <- 1

argset <- sc::tm_get_argset("example_weather_import_data_from_api", index_plan = index
schema <- sc::tm_get_schema("example_weather_import_data_from_api")

print(argset)
```

```
## $`**universal**`
## [1] "*"
##
## $`**plan**`
## [1] "*"
##
## $location_code
## [1] "county03"
##
## $`**analysis**`
## [1] "*"
##
## $`**automatic**`
## [1] "*"
##
## $index
## [1] 1
##
## $today
## [1] "2022-03-01"
##
## $yesterday
## [1] "2022-02-28"
##
## $first_analysis
## [1] TRUE
##
## $first_argset
## [1] TRUE
##
## $last_analysis
## [1] TRUE
##
## $last_argset
## [1] TRUE
print(names(schema))
## [1] "anon_example_weather"
```

```
# **** data_selector **** ----
#' example_weather_import_data_from_api (data selector)
#' @param argset Argset
#' @param schema DB Schema
#' @export
example_weather_import_data_from_api_data_selector = function(argset, schema){
  if(plnr::is_run_directly()){
```

```
    # sc::tm_get_plans_argsets_as_dt("example_weather_import_data_from_api")

    index_plan <- 1

    argset <- sc::tm_get_argset("example_weather_import_data_from_api", index_plan = in
    schema <- sc::tm_get_schema("example_weather_import_data_from_api")
  }

  # find the mid lat/long for the specified location_code
  gps <- fhimaps::norway_nuts3_map_b2020_default_dt[location_code == argset$location_co
    lat = mean(lat),
    long = mean(long)
  )]

  # download the forecast for the specified location_code
  d <- httr::GET(glue::glue("https://api.met.no/weatherapi/locationforecast/2.0/classic
  d <- xml2::read_xml(d$content)

  # The variable returned must be a named list
  retval <- list(
    "data" = d
  )
  retval
}
```

2.6 action__fn

The lines inside `if(plnr::is_run_directly()){` are used to help developers. You can run the code manually/interactively to "load" the values of `argset` and `schema`.

```
index_plan <- 1
index_analysis <- 1

data <- sc::tm_get_data("example_weather_import_data_from_api", index_plan = index_plan
argset <- sc::tm_get_argset("example_weather_import_data_from_api", index_plan = index_
schema <- sc::tm_get_schema("example_weather_import_data_from_api")

print(data)
## $data
## {xml_document}
## <weatherdata noNamespaceSchemaLocation="https://schema.api.met.no/schemas/weatherapi
## [1] <meta>\n  <model name="met_public_forecast" termin="2022-03-01T10:00:00Z" runend
## [2] <product class="pointData">\n  <time datatype="forecast" from="2022-03-01T10:00
print(argset)
```

```
## $`**universal**`
## [1] "*"
##
## $`**plan**`
## [1] "*"
##
## $location_code
## [1] "county03"
##
## $`**analysis**`
## [1] "*"
##
## $`**automatic**`
## [1] "*"
##
## $index
## [1] 1
##
## $today
## [1] "2022-03-01"
##
## $yesterday
## [1] "2022-02-28"
##
## $first_analysis
## [1] TRUE
##
## $first_argset
## [1] TRUE
##
## $last_analysis
## [1] TRUE
##
## $last_argset
## [1] TRUE
print(names(schema))
## [1] "anon_example_weather"
```

```
# **** action **** ----
#' example_weather_import_data_from_api (action)
#' @param data Data
#' @param argset Argset
#' @param schema DB Schema
#' @export
example_weather_import_data_from_api_action <- function(data, argset, schema) {
```

```r
# tm_run_task("example_weather_import_data_from_api")

if(plnr::is_run_directly()){
  # sc::tm_get_plans_argsets_as_dt("example_weather_import_data_from_api")

  index_plan <- 1
  index_analysis <- 1

  data <- sc::tm_get_data("example_weather_import_data_from_api", index_plan = index
  argset <- sc::tm_get_argset("example_weather_import_data_from_api", index_plan = in
  schema <- sc::tm_get_schema("example_weather_import_data_from_api")
}

# code goes here
# special case that runs before everything
if(argset$first_analysis == TRUE){

}

a <- data$data

baz <- xml2::xml_find_all(a, ".//maxTemperature")
res <- vector("list", length = length(baz))
for (i in seq_along(baz)) {
  parent <- xml2::xml_parent(baz[[i]])
  grandparent <- xml2::xml_parent(parent)
  time_from <- xml2::xml_attr(grandparent, "from")
  time_to <- xml2::xml_attr(grandparent, "to")
  x <- xml2::xml_find_all(parent, ".//minTemperature")
  temp_min <- xml2::xml_attr(x, "value")
  x <- xml2::xml_find_all(parent, ".//maxTemperature")
  temp_max <- xml2::xml_attr(x, "value")
  x <- xml2::xml_find_all(parent, ".//precipitation")
  precip <- xml2::xml_attr(x, "value")
  res[[i]] <- data.frame(
    time_from = as.character(time_from),
    time_to = as.character(time_to),
    tx = as.numeric(temp_max),
    tn = as.numeric(temp_min),
    rr = as.numeric(precip)
  )
}
res <- rbindlist(res)
res <- res[stringr::str_sub(time_from, 12, 13) %in% c("00", "06", "12", "18")]
res[, date := as.Date(stringr::str_sub(time_from, 1, 10))]
```

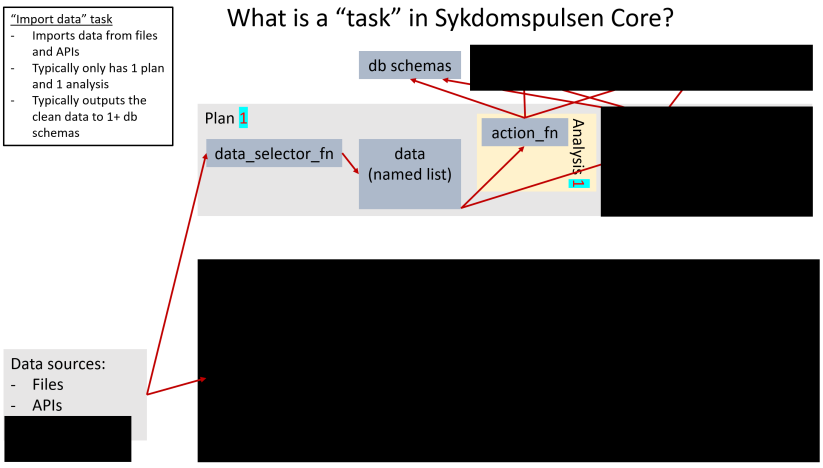
```
  res[, N := .N, by = date]
  res <- res[N == 4]
  res <- res[
    ,
    .(
      tg = NA,
      tx = max(tx),
      tn = min(tn),
      rr = sum(rr)
    ),
    keyby = .(date)
  ]

  # we look at the downloaded data
  print("Data after downloading")
  print(res)

  # we now need to format it
  res[, granularity_time := "day"]
  res[, sex := "total"]
  res[, age := "total"]
  res[, location_code := argset$location_code]

  # fill in missing structural variables
  sc::fill_in_missing_v8(res, border = 2020)

  # we look at the downloaded data
  print("Data after missing structural variables filled in")
  print(res)

  # put data in db table
  # schema$SCHEMA_NAME$insert_data(d)
  schema$anon_example_weather$upsert_data(res)
  # schema$SCHEMA_NAME$drop_all_rows_and_then_upsert_data(d)

  # special case that runs after everything
  # copy to anon_web?
  if(argset$last_analysis == TRUE){
    # sc::copy_into_new_table_where(
    #   table_from = "anon_X",
    #   table_to = "anon_webkht"
    # )
  }
}
```

2.7 Run the task

```
tm_run_task("example_weather_import_data_from_api")
## task: example_weather_import_data_from_api
## Running task=example_weather_import_data_from_api with plans=1 and analyses=1
## plans=sequential, argset=sequential with cores=1
## [----------------------------------------------------------------------------------------
## [========================================================================================
## [1] "Data after downloading"
##          date tg  tx   tn rr
## 1: 2022-03-02 NA 6.2 -3.0  0
## 2: 2022-03-03 NA 5.6 -2.3  0
## 3: 2022-03-04 NA 4.7 -3.3  0
## 4: 2022-03-05 NA 4.1 -1.9  0
## 5: 2022-03-06 NA 5.8 -2.9  0
## 6: 2022-03-07 NA 5.0 -2.6  0
## 7: 2022-03-08 NA 4.4 -1.1  0
## 8: 2022-03-09 NA 3.2 -1.4  0
## [1] "Data after missing structural variables filled in"
##          date tg  tx   tn rr granularity_time   sex   age location_code granularity_
## 1: 2022-03-02 NA 6.2 -3.0  0              day total total      county03          cou
## 2: 2022-03-03 NA 5.6 -2.3  0              day total total      county03          cou
## 3: 2022-03-04 NA 4.7 -3.3  0              day total total      county03          cou
## 4: 2022-03-05 NA 4.1 -1.9  0              day total total      county03          cou
## 5: 2022-03-06 NA 5.8 -2.9  0              day total total      county03          cou
## 6: 2022-03-07 NA 5.0 -2.6  0              day total total      county03          cou
## 7: 2022-03-08 NA 4.4 -1.1  0              day total total      county03          cou
## 8: 2022-03-09 NA 3.2 -1.4  0              day total total      county03          cou
##    calyear calmonth calyearmonth country_iso3
## 1:    2022        3     2022-M03          nor
## 2:    2022        3     2022-M03          nor
## 3:    2022        3     2022-M03          nor
## 4:    2022        3     2022-M03          nor
## 5:    2022        3     2022-M03          nor
## 6:    2022        3     2022-M03          nor
## 7:    2022        3     2022-M03          nor
## 8:    2022        3     2022-M03          nor
## Task ran in 0 mins
```

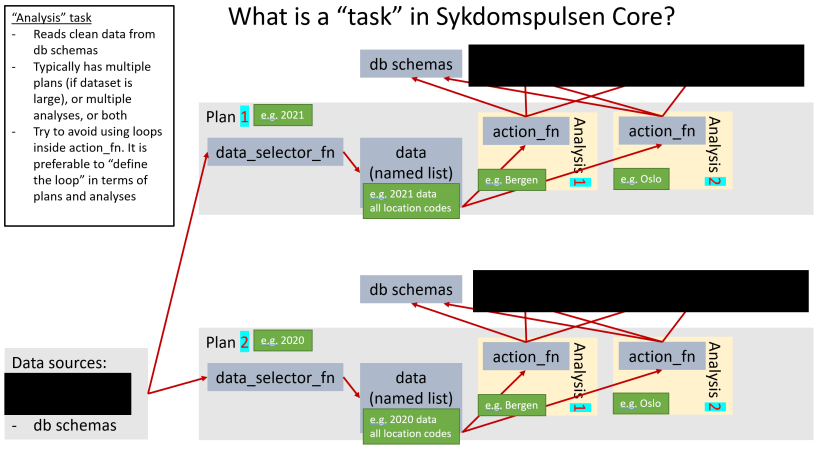
2.8 Examples of different types of tasks

2.8.1 Importing data



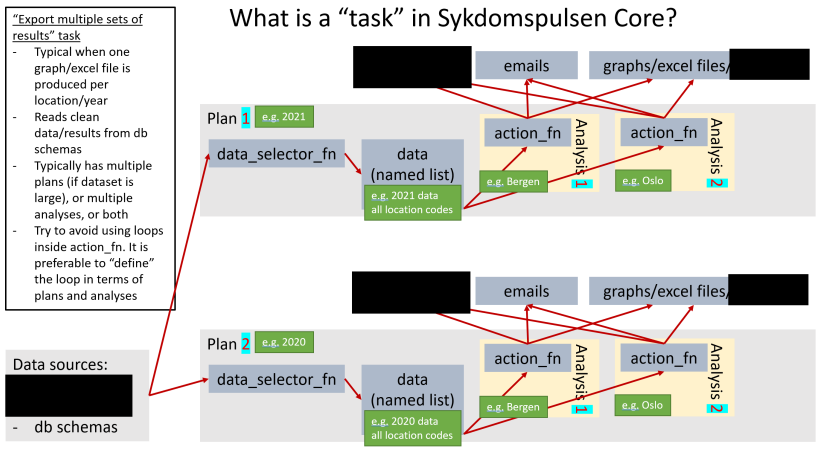

```
sc::add_task_from_config_v8(
  name_grouping = "example",
  name_action = "import_data",
  name_variant = NULL,
  cores = 1,
  plan_analysis_fn_name = NULL,
  for_each_plan = plnr::expand_list(
    x = 1
  ),
  for_each_analysis = NULL,
  universal_argset = list(
    folder = sc::path("input", "example")
  ),
  upsert_at_end_of_each_plan = FALSE,
  insert_at_end_of_each_plan = FALSE,
  action_fn_name = "example_import_data_action",
  data_selector_fn_name = "example_import_data_data_selector",
  schema = list(
    # input

    # output
    "output" = sc::config$schemas$output
  ),
  info = "This task does..."
)
```

2.8.2 Analysis



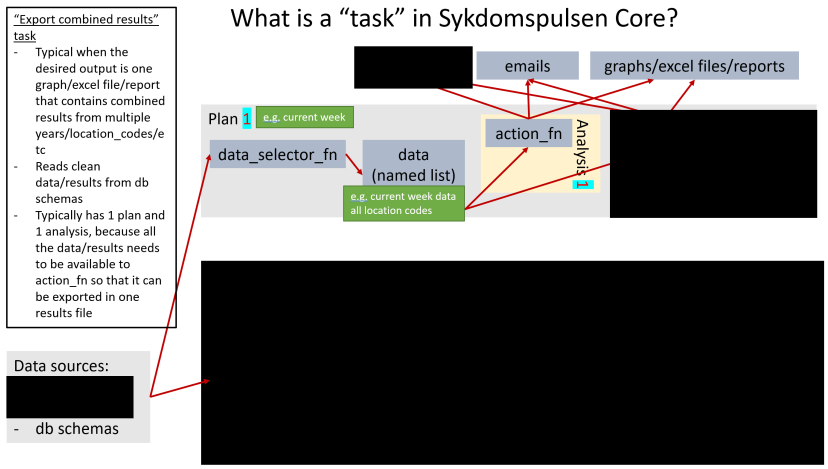

```
sc::add_task_from_config_v8(
  name_grouping = "example",
  name_action = "analysis",
  name_variant = NULL,
  cores = 1,
  plan_analysis_fn_name = NULL,
  for_each_plan = plnr::expand_list(
    location_code = fhidata::norway_locations_names()[granularity_geo %in% c("county")]
  ),
  for_each_analysis = NULL,
  universal_argset = NULL,
  upsert_at_end_of_each_plan = FALSE,
  insert_at_end_of_each_plan = FALSE,
  action_fn_name = "example_analysis_action",
  data_selector_fn_name = "example_analysis_data_selector",
  schema = list(
    # input
    "input" = sc::config$schemas$input,

    # output
    "output" = sc::config$schemas$output
  ),
  info = "This task does..."
)
```

2.8.3 Exporting multiple sets of results




```
sc::add_task_from_config_v8(
  name_grouping = "example",
  name_action = "export_results",
  name_variant = NULL,
  cores = 1,
  plan_analysis_fn_name = NULL,
  for_each_plan = plnr::expand_list(
    location_code = fhidata::norway_locations_names()[granularity_geo %in% c("county")]$location_
  ),
  for_each_analysis = NULL,
  universal_argset = list(
    folder = sc::path("output", "example")
  ),
  upsert_at_end_of_each_plan = FALSE,
  insert_at_end_of_each_plan = FALSE,
  action_fn_name = "example_export_results_action",
  data_selector_fn_name = "example_export_results_data_selector",
  schema = list(
    # input
    "input" = sc::config$schemas$input

    # output
  ),
  info = "This task does..."
)
```

2.8.4 Exporting combined results




```
sc::add_task_from_config_v8(
  name_grouping = "example",
  name_action = "export_results",
  name_variant = NULL,
  cores = 1,
  plan_analysis_fn_name = NULL,
  for_each_plan = plnr::expand_list(
    x = 1
  ),
  for_each_analysis = NULL,
  universal_argset = list(
    folder = sc::path("output", "example"),
    granularity_geos = c("nation", "county")
  ),
  upsert_at_end_of_each_plan = FALSE,
  insert_at_end_of_each_plan = FALSE,
  action_fn_name = "example_export_results_action",
  data_selector_fn_name = "example_export_results_data_selector",
  schema = list(
    # input
    "input" = sc::config$schemas$input

    # output
  ),
  info = "This task does..."
)
```

Chapter 3

File Layout

3.1 Introduction

Implementing Sykdomspulsen Core requires a number of functions to be called in the correct order. To make this as simple as possible, we have provided a skeleton implementation at https://github.com/sykdomspulsen-org/scskeleton

We suggest that you clone this GitHub repo to your server, and then do a global find/replace on `scskeleton` with the name you want for your R package.

Descriptions of the required files/functions are detailed below.

3.2 00_env_and_namespace.r

https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/00_env_and_namespace.r

```
## https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/00_env_and_namespace.r
##
##  1 | # ************************************************************************
##  2 | # ************************************************************************
##  3 | #
##  4 | # 00_env_and_namespace.r
##  5 | #
##  6 | # PURPOSE 1:
##  7 | #   Use roxygen2 to import ggplot2, data.table, %>%, and %<>% into the namespace,
##  8 | #   because these are the most commonly used packages/functions.
##  9 | #
## 10 | # PURPOSE 2:
## 11 | #   Declaring our own "tm_run_task" inside this package, as a wrapper around
## 12 | #   sc::tm_run_task.
```

```
## 13 | #
## 14 | #   We cannot run sc::tm_run_task directly, because we need to load all of the
## 15 | #   database connections, db schemas, tasks, etc. *before* we run the task.
## 16 | #   Hence, this wrapper ensures that all of this package's configs files are
## 17 | #   loaded via OURPACKAGE::.onLoad() first, and then sc::tm_run_task can run.
## 18 | #
## 19 | # PURPOSE 3:
## 20 | #   Declaration of environments that can be used globally.
## 21 | #
## 22 | # PURPOSE 4:
## 23 | #   Fix issues/integration with other packages.
## 24 | #
## 25 | #   Most notably is the issue with rmarkdown, where an error is thrown when
## 26 | #   rendering multiple rmarkdown documents in parallel.
## 27 | #
## 28 | # ********************************************************************************
## 29 | # ********************************************************************************
## 30 |
## 31 | #' @import ggplot2
## 32 | #' @import data.table
## 33 | #' @importFrom magrittr %>% %<>%
## 34 | 1
## 35 |
## 36 | #' Shortcut to run task
## 37 | #'
## 38 | #' This task is needed to ensure that all the definitions/db schemas/tasks/etc
## 39 | #' are loaded from the package scskeleton. We cannot run sc::tm_run_task direct
## 40 | #' because we need to load all of the database connections, db schemas, tasks,
## 41 | #' etc. *before* we run the task. Hence, this wrapper ensures that all of this
## 42 | #' package's configs files are loaded via OURPACKAGE::.onLoad() first, and ther
## 43 | #' sc::tm_run_task can run.
## 44 | #'
## 45 | #' @param task_name Name of the task
## 46 | #' @param index_plan Not used
## 47 | #' @param index_analysis Not used
## 48 | #' @export
## 49 | tm_run_task <- function(task_name, index_plan = NULL, index_analysis = NULL) {
## 50 |   sc::tm_run_task(
## 51 |     task_name = task_name,
## 52 |     index_plan = index_plan,
## 53 |     index_analysis = index_analysis
## 54 |   )
## 55 | }
## 56 |
## 57 | #' Declaration of environments that can be used globally
## 58 | #' @export config
```

```
## 59 | config <- new.env()
## 60 |
## 61 | # https://github.com/rstudio/rmarkdown/issues/1632
## 62 | # An error is thrown when rendering multiple rmarkdown documents in parallel.
## 63 | clean_tmpfiles_mod <- function() {
## 64 |   # message("Calling clean_tmpfiles_mod()")
## 65 | }
```

3.3 01_definitions.r

https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/01_definitio
ns.r

```
## https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/01_definitions.r
##
##  1 | # ****************************************************************************
##  2 | # ****************************************************************************
##  3 | #
##  4 | # 01_definitions.r
##  5 | #
##  6 | # PURPOSE 1:
##  7 | #   Set global definitions that are used throughout the package, and further
##  8 | #   (e.g. in shiny/plumber creations).
##  9 | #
## 10 | #   Examples of global definitions are:
## 11 | #     - Border years
## 12 | #     - Age definitions
## 13 | #     - Diagnosis mappings (e.g. "R80" = "Influenza")
## 14 | #
## 15 | # ****************************************************************************
## 16 | # ****************************************************************************
## 17 |
## 18 | #' Set global definitions
## 19 | set_definitions <- function() {
## 20 |
## 21 |   # Norway's last redistricting occurred 2020-01-01
## 22 |   config$border <- 2020
## 23 |
## 24 |   # fhidata needs to know which border is in use
## 25 |   # fhidata should also replace the population of 1900 with the current year,
## 26 |   # because year = 1900 is shorthand for granularity_geo = "total".
## 27 |   # This means that it is more appropriate to use the current year's population
## 28 |   # for year = 1900.
## 29 |   fhidata::set_config(
## 30 |     border = config$border,
```

```
## 31 |     use_current_year_as_1900_pop = TRUE
## 32 |   )
## 33 | }
```

3.4 02_permissions.r

https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/02_permissions.r

```
## https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/02_permissions.r
##
##  1 | # ********************************************************************************
##  2 | # ********************************************************************************
##  3 | #
##  4 | # 02_permissions.r
##  5 | #
##  6 | # PURPOSE 1:
##  7 | #   Set permissions that can be used in this package.
##  8 | #
##  9 | # PURPOSE 2:
## 10 | #   Permissions are a way of ensuring that a task only runs once per hour/day/w
## 11 | #   This can be useful when you want to be 100% sure that you don't want to spa
## 12 | #   emails to your recipients.
## 13 | #
## 14 | # PURPOSE 3:
## 15 | #   Permissions can also be used to differentiate between "production days" and
## 16 | #   "preliminary days". This can be useful when you have different email lists
## 17 | #   for production days (everyone) and preliminary days (a smaller group).
## 18 | #
## 19 | # ********************************************************************************
## 20 | # ********************************************************************************
## 21 |
## 22 | set_permissions <- function() {
## 23 |   # sc::add_permission(
## 24 |   #   name = "khtemails_send_emails",
## 25 |   #   permission = sc::Permission$new(
## 26 |   #     key = "khtemails_send_emails",
## 27 |   #     value = as.character(lubridate::today()),  # one time per day
## 28 |   #     production_days = c(3) # wed, send to everyone, otherwise prelim
## 29 |   #   )
## 30 |   # )
## 31 | }
```

3.5 03__db__schemas.r

https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/03__db__sch
emas.r

```
## https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/03_db_schemas.r
##
##   1 | # ****************************************************************************
##   2 | # ****************************************************************************
##   3 | #
##   4 | # 03_db_schemas.r
##   5 | #
##   6 | # PURPOSE 1:
##   7 | #   Set db schemas that are used throughout the package.
##   8 | #
##   9 | #   These are basically all of the database tables that you will be writing to,
##  10 | #   and reading from.
##  11 | #
##  12 | # ****************************************************************************
##  13 | # ****************************************************************************
##  14 |
##  15 | set_db_schemas <- function() {
##  16 |   # __________ ----
##  17 |   # Weather  ----
##  18 |   ## > anon_example_weather_rawdata ----
##  19 |   sc::add_schema_v8(
##  20 |     name_access = c("anon"),
##  21 |     name_grouping = "example_weather",
##  22 |     name_variant = "rawdata",
##  23 |     db_configs = sc::config$db_configs,
##  24 |     field_types =  c(
##  25 |       "granularity_time" = "TEXT",
##  26 |       "granularity_geo" = "TEXT",
##  27 |       "country_iso3" = "TEXT",
##  28 |       "location_code" = "TEXT",
##  29 |       "border" = "INTEGER",
##  30 |       "age" = "TEXT",
##  31 |       "sex" = "TEXT",
##  32 |
##  33 |       "date" = "DATE",
##  34 |
##  35 |       "isoyear" = "INTEGER",
##  36 |       "isoweek" = "INTEGER",
##  37 |       "isoyearweek" = "TEXT",
##  38 |       "season" = "TEXT",
##  39 |       "seasonweek" = "DOUBLE",
```

```
##  40 |
##  41 |       "calyear" = "INTEGER",
##  42 |       "calmonth" = "INTEGER",
##  43 |       "calyearmonth" = "TEXT",
##  44 |
##  45 |       "temp_max" = "DOUBLE",
##  46 |       "temp_min" = "DOUBLE",
##  47 |       "precip" = "DOUBLE"
##  48 |     ),
##  49 |     keys = c(
##  50 |       "granularity_time",
##  51 |       "location_code",
##  52 |       "date",
##  53 |       "age",
##  54 |       "sex"
##  55 |     ),
##  56 |     censors = list(
##  57 |       anon = list(
##  58 |
##  59 |       )
##  60 |     ),
##  61 |     validator_field_types = sc::validator_field_types_sykdomspulsen,
##  62 |     validator_field_contents = sc::validator_field_contents_sykdomspulsen,
##  63 |     info = "This db table is used for..."
##  64 |   )
##  65 |
##  66 |   ## > anon_example_weather_data ----
##  67 |   sc::add_schema_v8(
##  68 |     name_access = c("anon"),
##  69 |     name_grouping = "example_weather",
##  70 |     name_variant = "data",
##  71 |     db_configs = sc::config$db_configs,
##  72 |     field_types =  c(
##  73 |       "granularity_time" = "TEXT",
##  74 |       "granularity_geo" = "TEXT",
##  75 |       "country_iso3" = "TEXT",
##  76 |       "location_code" = "TEXT",
##  77 |       "border" = "INTEGER",
##  78 |       "age" = "TEXT",
##  79 |       "sex" = "TEXT",
##  80 |
##  81 |       "date" = "DATE",
##  82 |
##  83 |       "isoyear" = "INTEGER",
##  84 |       "isoweek" = "INTEGER",
##  85 |       "isoyearweek" = "TEXT",
```

```
##  86 |       "season" = "TEXT",
##  87 |       "seasonweek" = "DOUBLE",
##  88 |
##  89 |       "calyear" = "INTEGER",
##  90 |       "calmonth" = "INTEGER",
##  91 |       "calyearmonth" = "TEXT",
##  92 |
##  93 |       "temp_max" = "DOUBLE",
##  94 |       "temp_min" = "DOUBLE",
##  95 |       "precip" = "DOUBLE"
##  96 |     ),
##  97 |     keys = c(
##  98 |       "granularity_time",
##  99 |       "location_code",
## 100 |       "date",
## 101 |       "age",
## 102 |       "sex"
## 103 |     ),
## 104 |     censors = list(
## 105 |       anon = list(
## 106 |
## 107 |       )
## 108 |     ),
## 109 |     validator_field_types = sc::validator_field_types_sykdomspulsen,
## 110 |     validator_field_contents = sc::validator_field_contents_sykdomspulsen,
## 111 |     info = "This db table is used for..."
## 112 |   )
## 113 | }
```

3.6 04__tasks.r

https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/04__tasks.r

```
## https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/04_tasks.r
##
##   1 | # ****************************************************************************
##   2 | # ****************************************************************************
##   3 | #
##   4 | # 04_tasks.r
##   5 | #
##   6 | # PURPOSE 1:
##   7 | #   Set all the tasks that are run by the package.
##   8 | #
##   9 | #   These are basically all of the "things" that you want to do.
##  10 | #   E.g. Downloading data, cleaning data, importing data, analyzing data,
```

```
##  11 | #   making Excel files, making docx/pdf reports, sending emails, etc.
##  12 | #
##  13 | # ******************************************************************************
##  14 | # ******************************************************************************
##  15 |
##  16 | set_tasks <- function() {
##  17 |   # __________ ----
##  18 |   # Weather  ----
##  19 |   ## > weather_download_and_import_rawdata ----
##  20 |   # tm_run_task("weather_download_and_import_rawdata")
##  21 |   sc::add_task_from_config_v8(
##  22 |     name_grouping = "weather",
##  23 |     name_action = "download_and_import_rawdata",
##  24 |     name_variant = NULL,
##  25 |     cores = 1,
##  26 |     plan_analysis_fn_name = NULL,
##  27 |     for_each_plan = plnr::expand_list(
##  28 |       location_code = fhidata::norway_locations_names()[granularity_geo %in% c
##  29 |     ),
##  30 |     for_each_analysis = NULL,
##  31 |     universal_argset = NULL,
##  32 |     upsert_at_end_of_each_plan = FALSE,
##  33 |     insert_at_end_of_each_plan = FALSE,
##  34 |     action_fn_name = "scskeleton::weather_download_and_import_rawdata_action"
##  35 |     data_selector_fn_name = "scskeleton::weather_download_and_import_rawdata_c
##  36 |     schema = list(
##  37 |       # input
##  38 |
##  39 |       # output
##  40 |       "anon_example_weather_rawdata" = sc::config$schemas$anon_example_weather
##  41 |     ),
##  42 |     info = "This task downloads and imports the raw weather data from MET's AP
##  43 |   )
##  44 |
##  45 |   ## > weather_clean_data ----
##  46 |   # tm_run_task("weather_clean_data")
##  47 |   sc::add_task_from_config_v8(
##  48 |     name_grouping = "weather",
##  49 |     name_action = "clean_data",
##  50 |     name_variant = NULL,
##  51 |     cores = 1,
##  52 |     plan_analysis_fn_name = NULL,
##  53 |     for_each_plan = plnr::expand_list(
##  54 |       x = 1
##  55 |     ),
##  56 |     for_each_analysis = NULL,
```

```
##  57 |     universal_argset = NULL,
##  58 |     upsert_at_end_of_each_plan = FALSE,
##  59 |     insert_at_end_of_each_plan = FALSE,
##  60 |     action_fn_name = "scskeleton::weather_clean_data_action",
##  61 |     data_selector_fn_name = "scskeleton::weather_clean_data_data_selector",
##  62 |     schema = list(
##  63 |       # input
##  64 |       "anon_example_weather_rawdata" = sc::config$schemas$anon_example_weather_rawdata,
##  65 |
##  66 |       # output
##  67 |       "anon_example_weather_data" = sc::config$schemas$anon_example_weather_data
##  68 |     ),
##  69 |     info = "This task cleans the raw data and aggregates it to county and national level
##  70 |   )
##  71 |
##  72 |   ## > weather_clean_data ----
##  73 |   # tm_run_task("weather_export_plots")
##  74 |   sc::add_task_from_config_v8(
##  75 |     name_grouping = "weather",
##  76 |     name_action = "export_plots",
##  77 |     name_variant = NULL,
##  78 |     cores = 1,
##  79 |     plan_analysis_fn_name = NULL,
##  80 |     for_each_plan = plnr::expand_list(
##  81 |       location_code = fhidata::norway_locations_names()[granularity_geo %in% c("county")
##  82 |     ),
##  83 |     for_each_analysis = NULL,
##  84 |     universal_argset = list(
##  85 |       output_dir = tempdir(),
##  86 |       output_filename = "weather_{argset$location_code}.png",
##  87 |       output_absolute_path = fs::path("{argset$output_dir}", "{argset$output_filename}")
##  88 |     ),
##  89 |     upsert_at_end_of_each_plan = FALSE,
##  90 |     insert_at_end_of_each_plan = FALSE,
##  91 |     action_fn_name = "scskeleton::weather_export_plots_action",
##  92 |     data_selector_fn_name = "scskeleton::weather_export_plots_data_selector",
##  93 |     schema = list(
##  94 |       # input
##  95 |       "anon_example_weather_data" = sc::config$schemas$anon_example_weather_data
##  96 |
##  97 |       # output
##  98 |     ),
##  99 |     info = "This task ploduces plots"
## 100 |   )
## 101 | }
```

3.7 05__deliverables.r

https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/05__deliverables.r

```
## https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/05_deliverables.r
##
##  1 | # ****************************************************************************
##  2 | # ****************************************************************************
##  3 | #
##  4 | # 05_deliverables.r
##  5 | #
##  6 | # PURPOSE 1:
##  7 | #   Set all the deliverables that team members are supposed to manually do/che
##  8 | #   every day/week/month.
##  9 | #
## 10 | # ****************************************************************************
## 11 | # ****************************************************************************
## 12 |
## 13 | set_deliverables <- function() {
## 14 |
## 15 | }
```

3.8 06__config.r

https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/06__config.r

```
## https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/06_config.r
##
##  1 | # ****************************************************************************
##  2 | # ****************************************************************************
##  3 | #
##  4 | # 06_config.r
##  5 | #
##  6 | # PURPOSE 1:
##  7 | #   Call all the functions defined in 01, 02, 03, 04, and 05 in the correct or
##  8 | #
##  9 | # PURPOSE 2:
## 10 | #   Set all necessary configs that do not belong anywhere else.
## 11 | #
## 12 | #   E.g. Formatting for progress bars.
## 13 | #
## 14 | # ****************************************************************************
## 15 | # ****************************************************************************
## 16 |
## 17 | set_config <- function() {
```

```
## 18 |   # 01_definitions.r
## 19 |   set_definitions()
## 20 |
## 21 |   # 02_permissions.r
## 22 |   set_permissions()
## 23 |
## 24 |   # 03_db_schemas.r
## 25 |   set_db_schemas()
## 26 |
## 27 |   # 04_tasks.r
## 28 |   set_tasks()
## 29 |
## 30 |   # 05_deliverables.r
## 31 |   set_deliverables()
## 32 |
## 33 |   # 06_config.r
## 34 |   set_progressr()
## 35 | }
## 36 |
## 37 | set_progressr <- function() {
## 38 |   progressr::handlers(progressr::handler_progress(
## 39 |     format = "[:bar] :current/:total (:percent) in :elapsedfull, eta: :eta",
## 40 |     clear = FALSE
## 41 |   ))
## 42 | }
```

3.9 07_onLoad.r

https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/07_onLoad
.r

```
## https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/07_onLoad.r
##
##  1 | # ****************************************************************************
##  2 | # ****************************************************************************
##  3 | #
##  4 | # 07_onLoad.r
##  5 | #
##  6 | # PURPOSE 1:
##  7 | #   Initializing everything that happens when the package is loaded.
##  8 | #
##  9 | #   E.g. Calling bash scripts that authenticate against Kerebros, setting the
## 10 | #   configs as defined in 06_config.r.
## 11 | #
## 12 | # ****************************************************************************
```

```
## 13 | # ********************************************************************************
## 14 |
## 15 | .onLoad <- function(libname, pkgname) {
## 16 |   # Mechanism to authenticate as necessary (e.g. Kerebros)
## 17 |   try(system2("/bin/authenticate.sh", stdout = NULL), TRUE)
## 18 |
## 19 |   # 5_config.r
## 20 |   set_config()
## 21 |
## 22 |   # https://github.com/rstudio/rmarkdown/issues/1632
## 23 |   assignInNamespace("clean_tmpfiles", clean_tmpfiles_mod, ns = "rmarkdown")
## 24 |
## 25 |   invisible()
## 26 | }
```

3.10 08_onAttach.r

https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/08_onAtta
ch.r

```
## https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/08_onAttach.r
##
##  1 | # ********************************************************************************
##  2 | # ********************************************************************************
##  3 | #
##  4 | # 08_onAttach.r
##  5 | #
##  6 | # PURPOSE 1:
##  7 | #   What you want to happen when someone types library(yourpackage)
##  8 | #
##  9 | # ********************************************************************************
## 10 | # ********************************************************************************
## 11 |
## 12 | .onAttach <- function(libname, pkgname) {
## 13 |
## 14 | }
```

3.11 99_util_*.r

https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/99_util_n
o_data_plot.r

```
## https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/99_util_no_data_plot.r
##
##  1 | # ********************************************************************************
```

```
##  2 | # ****************************************************************************
##  3 | #
##  4 | # 99_util_*.r
##  5 | #
##  6 | # PURPOSE 1:
##  7 | #   Utility functions that are used across multiple tasks
##  8 | #
##  9 | # ****************************************************************************
## 10 | # ****************************************************************************
## 11 |
## 12 | no_data_plot <- function(){
## 13 |   data=data.frame(x=0,y=0)
## 14 |   q <- ggplot(data=data)
## 15 |   q <- q + theme_void()
## 16 |   q <- q + annotate("text", label=glue::glue("Ikke noe data {fhi::nb$aa} vise"), x=0, y=0
## 17 |   q
## 18 | }
```

3.12 Task files

Task files are placed in .r files under their own names.

3.12.1 weather_download_and_import_rawdata.r

https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/weather_d
ownload_and_import_rawdata.r

```
## https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/weather_download_and_import_rawdat
##
##   1 | # **** action **** ----
##   2 | #' weather_download_and_import_rawdata (action)
##   3 | #' @param data Data
##   4 | #' @param argset Argset
##   5 | #' @param schema DB Schema
##   6 | #' @export
##   7 | weather_download_and_import_rawdata_action <- function(data, argset, schema) {
##   8 |   # tm_run_task("weather_download_and_import_rawdata")
##   9 |
##  10 |   if (plnr::is_run_directly()) {
##  11 |     # sc::tm_get_plans_argsets_as_dt("weather_download_and_import_rawdata")
##  12 |
##  13 |     index_plan <- 1
##  14 |     index_analysis <- 1
##  15 |
##  16 |     data <- sc::tm_get_data("weather_download_and_import_rawdata", index_plan = index_pl
##  17 |     argset <- sc::tm_get_argset("weather_download_and_import_rawdata", index_plan = inde
```

```
##  18 |     schema <- sc::tm_get_schema("weather_download_and_import_rawdata")
##  19 |   }
##  20 |
##  21 |   # special case that runs before everything
##  22 |   if (argset$first_analysis == TRUE) {
##  23 |
##  24 |   }
##  25 |
##  26 |   a <- data$data
##  27 |
##  28 |   baz <- xml2::xml_find_all(a, ".//maxTemperature")
##  29 |   res <- vector("list", length = length(baz))
##  30 |   for (i in seq_along(baz)) {
##  31 |     parent <- xml2::xml_parent(baz[[i]])
##  32 |     grandparent <- xml2::xml_parent(parent)
##  33 |     time_from <- xml2::xml_attr(grandparent, "from")
##  34 |     time_to <- xml2::xml_attr(grandparent, "to")
##  35 |     x <- xml2::xml_find_all(parent, ".//minTemperature")
##  36 |     temp_min <- xml2::xml_attr(x, "value")
##  37 |     x <- xml2::xml_find_all(parent, ".//maxTemperature")
##  38 |     temp_max <- xml2::xml_attr(x, "value")
##  39 |     x <- xml2::xml_find_all(parent, ".//precipitation")
##  40 |     precip <- xml2::xml_attr(x, "value")
##  41 |     res[[i]] <- data.frame(
##  42 |       time_from = as.character(time_from),
##  43 |       time_to = as.character(time_to),
##  44 |       temp_max = as.numeric(temp_max),
##  45 |       temp_min = as.numeric(temp_min),
##  46 |       precip = as.numeric(precip)
##  47 |     )
##  48 |   }
##  49 |   res <- rbindlist(res)
##  50 |   res <- res[stringr::str_sub(time_from, 12, 13) %in% c("00", "06", "12", "18"
##  51 |   res[, date := as.Date(stringr::str_sub(time_from, 1, 10))]
##  52 |   res[, N := .N, by = date]
##  53 |   res <- res[N == 4]
##  54 |   res <- res[
##  55 |     ,
##  56 |     .(
##  57 |       temp_max = max(temp_max),
##  58 |       temp_min = min(temp_min),
##  59 |       precip = sum(precip)
##  60 |     ),
##  61 |     keyby = .(date)
##  62 |   ]
##  63 |
```

```
##  64 |   # we look at the downloaded data
##  65 |   # res
##  66 |
##  67 |   # we now need to format it
##  68 |   res[, granularity_time := "day"]
##  69 |   res[, sex := "total"]
##  70 |   res[, age := "total"]
##  71 |   res[, location_code := argset$location_code]
##  72 |
##  73 |   # fill in missing structural variables
##  74 |   sc::fill_in_missing_v8(res, border = 2020)
##  75 |
##  76 |   # we look at the downloaded data
##  77 |   # res
##  78 |
##  79 |   # put data in db table
##  80 |   schema$anon_example_weather_rawdata$insert_data(res)
##  81 |
##  82 |   # special case that runs after everything
##  83 |   if (argset$last_analysis == TRUE) {
##  84 |
##  85 |   }
##  86 | }
##  87 |
##  88 | # **** data_selector **** ----
##  89 | #' weather_download_and_import_rawdata (data selector)
##  90 | #' @param argset Argset
##  91 | #' @param schema DB Schema
##  92 | #' @export
##  93 | weather_download_and_import_rawdata_data_selector <- function(argset, schema) {
##  94 |   if (plnr::is_run_directly()) {
##  95 |     # sc::tm_get_plans_argsets_as_dt("weather_download_and_import_rawdata")
##  96 |
##  97 |     index_plan <- 1
##  98 |
##  99 |     argset <- sc::tm_get_argset("weather_download_and_import_rawdata", index_plan = inde
## 100 |     schema <- sc::tm_get_schema("weather_download_and_import_rawdata")
## 101 |   }
## 102 |
## 103 |   # find the mid lat/long for the specified location_code
## 104 |   gps <- fhimaps::norway_lau2_map_b2020_default_dt[location_code == argset$location_code
## 105 |     lat = mean(lat),
## 106 |     long = mean(long)
## 107 |   )]
## 108 |
## 109 |   # download the forecast for the specified location_code
```

```
## 110 |   d <- httr::GET(glue::glue("https://api.met.no/weatherapi/locationforecast/2
## 111 |   d <- xml2::read_xml(d$content)
## 112 |
## 113 |   # The variable returned must be a named list
## 114 |   retval <- list(
## 115 |     "data" = d
## 116 |   )
## 117 |
## 118 |   retval
## 119 | }
## 120 |
## 121 | # **** functions **** ----
```

3.12.2 weather_clean_data.r

https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/weather_clean_data.r

```
## https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/weather_clean_data.r
##
##   1 | # **** action **** ----
##   2 | #' weather_clean_data (action)
##   3 | #' @param data Data
##   4 | #' @param argset Argset
##   5 | #' @param schema DB Schema
##   6 | #' @export
##   7 | weather_clean_data_action <- function(data, argset, schema) {
##   8 |   # tm_run_task("weather_clean_data")
##   9 |
##  10 |   if (plnr::is_run_directly()) {
##  11 |     # sc::tm_get_plans_argsets_as_dt("weather_clean_data")
##  12 |
##  13 |     index_plan <- 1
##  14 |     index_analysis <- 1
##  15 |
##  16 |     data <- sc::tm_get_data("weather_clean_data", index_plan = index_plan)
##  17 |     argset <- sc::tm_get_argset("weather_clean_data", index_plan = index_plan
##  18 |     schema <- sc::tm_get_schema("weather_clean_data")
##  19 |   }
##  20 |
##  21 |   # special case that runs before everything
##  22 |   if (argset$first_analysis == TRUE) {
##  23 |
##  24 |   }
##  25 |
##  26 |   # make sure there's no missing data via the creation of a skeleton
```

```
##  27 |   # https://folkehelseinstituttet.github.io/fhidata/articles/Skeletons.html
##  28 |
##  29 |   # Create a variable (possibly a list) to hold the data
##  30 |   d_agg <- list()
##  31 |   d_agg$day_municip <- copy(data$day_municip)
##  32 |
##  33 |   # Pull out important dates
##  34 |   date_min <- min(d_agg$day_municip$date, na.rm = T)
##  35 |   date_max <- max(d_agg$day_municip$date, na.rm = T)
##  36 |
##  37 |   # Create `multiskeleton`
##  38 |   # granularity_geo should have the following groups:
##  39 |   # - nodata (when no data is available, and there is no "finer" data available to aggre
##  40 |   # - all levels of granularity_geo where you have data available
##  41 |   # If you do not have data for a specific granularity_geo, but there is "finer" data av
##  42 |   # then you should not include this granularity_geo in the multiskeleton, because you w
##  43 |   # it later when you aggregate up your data (baregion)
##  44 |   multiskeleton_day <- fhidata::make_skeleton(
##  45 |     date_min = date_min,
##  46 |     date_max = date_max,
##  47 |     granularity_geo = list(
##  48 |       "nodata" = c(
##  49 |         "wardoslo",
##  50 |         "extrawardoslo",
##  51 |         "missingwardoslo",
##  52 |         "wardbergen",
##  53 |         "missingwardbergen",
##  54 |         "wardstavanger",
##  55 |         "missingwardstavanger",
##  56 |         "notmainlandmunicip",
##  57 |         "missingmunicip",
##  58 |         "notmainlandcounty",
##  59 |         "missingcounty"
##  60 |       ),
##  61 |       "municip" = c(
##  62 |         "municip"
##  63 |       )
##  64 |     )
##  65 |   )
##  66 |
##  67 |   # Merge in the information you have at different geographical granularities
##  68 |   # one level at a time
##  69 |   # municip
##  70 |   multiskeleton_day$municip[
##  71 |     d_agg$day_municip,
##  72 |     on = c("location_code", "date"),
```

```
##  73 |     c(
##  74 |       "temp_max",
##  75 |       "temp_min",
##  76 |       "precip"
##  77 |     ) := .(
##  78 |       temp_max,
##  79 |       temp_min,
##  80 |       precip
##  81 |     )
##  82 |   ]
##  83 |
##  84 |   multiskeleton_day$municip[]
##  85 |
##  86 |   # Aggregate up to higher geographical granularities (county)
##  87 |   multiskeleton_day$county <- multiskeleton_day$municip[
##  88 |     fhidata::norway_locations_hierarchy(
##  89 |       from = "municip",
##  90 |       to = "county"
##  91 |     ),
##  92 |     on = c(
##  93 |       "location_code==from_code"
##  94 |     )
##  95 |   ][,
##  96 |     .(
##  97 |       temp_max = mean(temp_max, na.rm = T),
##  98 |       temp_min = mean(temp_min, na.rm = T),
##  99 |       precip = mean(precip, na.rm = T),
## 100 |       granularity_geo = "county"
## 101 |     ),
## 102 |     by = .(
## 103 |       granularity_time,
## 104 |       date,
## 105 |       location_code = to_code
## 106 |     )
## 107 |   ]
## 108 |
## 109 |   multiskeleton_day$county[]
## 110 |
## 111 |   # Aggregate up to higher geographical granularities (nation)
## 112 |   multiskeleton_day$nation <- multiskeleton_day$municip[
## 113 |     ,
## 114 |     .(
## 115 |       temp_max = mean(temp_max, na.rm = T),
## 116 |       temp_min = mean(temp_min, na.rm = T),
## 117 |       precip = mean(precip, na.rm = T),
## 118 |       granularity_geo = "nation",
```

```
## 119 |       location_code = "norge"
## 120 |     ),
## 121 |     by = .(
## 122 |       granularity_time,
## 123 |       date
## 124 |     )
## 125 |   ]
## 126 |
## 127 |   multiskeleton_day$nation[]
## 128 |
## 129 |   # combine all the different granularity_geos
## 130 |   skeleton_day <- rbindlist(multiskeleton_day, fill = TRUE, use.names = TRUE)
## 131 |
## 132 |   skeleton_day[]
## 133 |
## 134 |   # 10. (If desirable) aggregate up to higher time granularities
## 135 |   # if necessary, it is now easy to aggregate up to weekly data from here
## 136 |   skeleton_isoweek <- copy(skeleton_day)
## 137 |   skeleton_isoweek[, isoyearweek := fhiplot::isoyearweek_c(date)]
## 138 |   skeleton_isoweek <- skeleton_isoweek[
## 139 |     ,
## 140 |     .(
## 141 |       temp_max = mean(temp_max, na.rm = T),
## 142 |       temp_min = mean(temp_min, na.rm = T),
## 143 |       precip = mean(precip, na.rm = T),
## 144 |       granularity_time = "isoweek"
## 145 |     ),
## 146 |     keyby = .(
## 147 |       isoyearweek,
## 148 |       granularity_geo,
## 149 |       location_code
## 150 |     )
## 151 |   ]
## 152 |
## 153 |   skeleton_isoweek[]
## 154 |
## 155 |   # we now need to format it and fill in missing structural variables
## 156 |   # day
## 157 |   skeleton_day[, sex := "total"]
## 158 |   skeleton_day[, age := "total"]
## 159 |   sc::fill_in_missing_v8(skeleton_day, border = config$border)
## 160 |
## 161 |   # isoweek
## 162 |   skeleton_isoweek[, sex := "total"]
## 163 |   skeleton_isoweek[, age := "total"]
## 164 |   sc::fill_in_missing_v8(skeleton_isoweek, border = config$border)
```

```
## 165 |   skeleton_isoweek[, date := as.Date(date)]
## 166 |
## 167 |   skeleton <- rbindlist(
## 168 |     list(
## 169 |       skeleton_day,
## 170 |       skeleton_isoweek
## 171 |     ),
## 172 |     use.names = T
## 173 |   )
## 174 |
## 175 |   # put data in db table
## 176 |   schema$anon_example_weather_data$drop_all_rows_and_then_insert_data(skeleto
## 177 |
## 178 |   # special case that runs after everything
## 179 |   if (argset$last_analysis == TRUE) {
## 180 |
## 181 |   }
## 182 | }
## 183 |
## 184 | # **** data_selector **** ----
## 185 | #' weather_clean_data (data selector)
## 186 | #' @param argset Argset
## 187 | #' @param schema DB Schema
## 188 | #' @export
## 189 | weather_clean_data_data_selector <- function(argset, schema) {
## 190 |   if (plnr::is_run_directly()) {
## 191 |     # sc::tm_get_plans_argsets_as_dt("weather_clean_data")
## 192 |
## 193 |     index_plan <- 1
## 194 |
## 195 |     argset <- sc::tm_get_argset("weather_clean_data", index_plan = index_plan
## 196 |     schema <- sc::tm_get_schema("weather_clean_data")
## 197 |   }
## 198 |
## 199 |   # The database schemas can be accessed here
## 200 |   d <- schema$anon_example_weather_rawdata$tbl() %>%
## 201 |     sc::mandatory_db_filter(
## 202 |       granularity_time = "day",
## 203 |       granularity_time_not = NULL,
## 204 |       granularity_geo = "municip",
## 205 |       granularity_geo_not = NULL,
## 206 |       country_iso3 = NULL,
## 207 |       location_code = NULL,
## 208 |       age = "total",
## 209 |       age_not = NULL,
## 210 |       sex = "total",
```

```
## 211 |       sex_not = NULL
## 212 |     ) %>%
## 213 |     dplyr::select(
## 214 |       granularity_time,
## 215 |       # granularity_geo,
## 216 |       # country_iso3,
## 217 |       location_code,
## 218 |       # border,
## 219 |       # age,
## 220 |       # sex,
## 221 |
## 222 |       date,
## 223 |
## 224 |       # isoyear,
## 225 |       # isoweek,
## 226 |       # isoyearweek,
## 227 |       # season,
## 228 |       # seasonweek,
## 229 |
## 230 |       # calyear,
## 231 |       # calmonth,
## 232 |       # calyearmonth,
## 233 |
## 234 |       temp_max,
## 235 |       temp_min,
## 236 |       precip
## 237 |     ) %>%
## 238 |     dplyr::collect() %>%
## 239 |     as.data.table() %>%
## 240 |     setorder(
## 241 |       location_code,
## 242 |       date
## 243 |     )
## 244 |
## 245 |   # The variable returned must be a named list
## 246 |   retval <- list(
## 247 |     "day_municip" = d
## 248 |   )
## 249 |
## 250 |   retval
## 251 | }
## 252 |
## 253 | # **** functions **** ----
```

3.12.3 weather_export_weather_plots.r

https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/weather_export_plots.r

```
## https://github.com/sykdomspulsen-org/scskeleton/blob/main/R/weather_export_plots.r
##
##   1 | # **** action **** ----
##   2 | #' weather_export_plots (action)
##   3 | #' @param data Data
##   4 | #' @param argset Argset
##   5 | #' @param schema DB Schema
##   6 | #' @export
##   7 | weather_export_plots_action <- function(data, argset, schema) {
##   8 |   # tm_run_task("weather_export_plots")
##   9 |
##  10 |   if(plnr::is_run_directly()){
##  11 |     # sc::tm_get_plans_argsets_as_dt("weather_export_plots")
##  12 |
##  13 |     index_plan <- 1
##  14 |     index_analysis <- 1
##  15 |
##  16 |     data <- sc::tm_get_data("weather_export_plots", index_plan = index_plan)
##  17 |     argset <- sc::tm_get_argset("weather_export_plots", index_plan = index_pla
##  18 |     schema <- sc::tm_get_schema("weather_export_plots")
##  19 |   }
##  20 |
##  21 |   # code goes here
##  22 |   # special case that runs before everything
##  23 |   if(argset$first_analysis == TRUE){
##  24 |
##  25 |   }
##  26 |
##  27 |   # create the output_dir (if it doesn't exist)
##  28 |   fs::dir_create(glue::glue(argset$output_dir))
##  29 |
##  30 |   q <- ggplot(data$data, aes(x = date, ymin = temp_min, ymax = temp_max))
##  31 |   q <- q + geom_ribbon(alpha = 0.5)
##  32 |
##  33 |   ggsave(
##  34 |     filename = glue::glue(argset$output_absolute_path),
##  35 |     plot = q
##  36 |   )
##  37 |
##  38 |   # special case that runs after everything
##  39 |   # copy to anon_web?
```

```
##  40 |   if(argset$last_analysis == TRUE){
##  41 |
##  42 |   }
##  43 | }
##  44 |
##  45 | # **** data_selector **** ----
##  46 | #' weather_export_plots (data selector)
##  47 | #' @param argset Argset
##  48 | #' @param schema DB Schema
##  49 | #' @export
##  50 | weather_export_plots_data_selector = function(argset, schema){
##  51 |   if(plnr::is_run_directly()){
##  52 |     # sc::tm_get_plans_argsets_as_dt("weather_export_plots")
##  53 |
##  54 |     index_plan <- 1
##  55 |
##  56 |     argset <- sc::tm_get_argset("weather_export_plots", index_plan = index_plan)
##  57 |     schema <- sc::tm_get_schema("weather_export_plots")
##  58 |   }
##  59 |
##  60 |   # The database schemas can be accessed here
##  61 |   d <- schema$anon_example_weather_data$tbl() %>%
##  62 |     sc::mandatory_db_filter(
##  63 |       granularity_time = NULL,
##  64 |       granularity_time_not = NULL,
##  65 |       granularity_geo = NULL,
##  66 |       granularity_geo_not = NULL,
##  67 |       country_iso3 = NULL,
##  68 |       location_code = argset$location_code,
##  69 |       age = NULL,
##  70 |       age_not = NULL,
##  71 |       sex = NULL,
##  72 |       sex_not = NULL
##  73 |     ) %>%
##  74 |     dplyr::select(
##  75 |       # granularity_time,
##  76 |       # granularity_geo,
##  77 |       # country_iso3,
##  78 |       # location_code,
##  79 |       # border,
##  80 |       # age,
##  81 |       # sex,
##  82 |
##  83 |       date,
##  84 |
##  85 |       # isoyear,
```

```
##  86 |       # isoweek,
##  87 |       # isoyearweek,
##  88 |       # season,
##  89 |       # seasonweek,
##  90 |       #
##  91 |       # calyear,
##  92 |       # calmonth,
##  93 |       # calyearmonth,
##  94 |
##  95 |       temp_max,
##  96 |       temp_min
##  97 |     ) %>%
##  98 |     dplyr::collect() %>%
##  99 |     as.data.table() %>%
## 100 |     setorder(
## 101 |       # location_code,
## 102 |       date
## 103 |     )
## 104 |
## 105 |   # The variable returned must be a named list
## 106 |   retval <- list(
## 107 |     "data" = d
## 108 |   )
## 109 |   retval
## 110 | }
## 111 |
## 112 | # **** functions **** ----
## 113 |
## 114 |
## 115 |
## 116 |
```

Chapter 4

Tutorial 1: Introduction

4.1 Setup

Implementing Sykdomspulsen Core requires a number of functions to be called in the correct order. To make this as simple as possible, we have provided a skeleton implementation at sykdomspulsen-org/scskeleton

For this tutorial you should clone GitHub repo to your server. This will be the package that you will be working on throughout this tutorial. You may choose to do a global find/replace on `sc-tutorial-start` with the name you want for your R package. We will refer to this R package as your "sc implementation".

You can also clone sykdomspulsen-org/sc-tutorial-end to your server. This is the end product of the tutorial, and you should refer to it in order to check your work.

For the purposes of this tutorial, we assume that the reader is either using RStudio Server Open Source or RStudio Workbench inside Docker containers that have been built according to the Sykdomspulsen specifications. We will refer to your implementation of RStudio Server Open Source/RStudio Workbench with the generic term "RStudio".

4.2 Load the code

Open`sc-tutorial-start` in RStudio project mode. Restart the R session via `Ctrl+Shift+F10`, `rstudioapi::restartSession()`, or `Session > Restart R`. This will ensure that you have a clean working environment before you begin. You may now load your sc implementation. This can be done via `Ctrl+Shift+L`, `devtools::load_all(".")`, or `Build > Load All`.

In general, we recommend cleaning your working environment every time before

running `devtools::load_all(".")`.

```
rstudioapi::restartSession()
devtools::load_all(".")
```

It you are wokring in the sykdomspulsen infrastructure you might see a warning message on the form: sh: `1: /bin/authenticate.sh: not found`. This has to do with authentication beeing done automatically on sign in. You do not have to worry about this for the purpose of this tutorial. You might also see a warnign starting with "Objects listed as exports, but not present in namespace:". This can also be ignored.

You can now see which schemas have been loaded by running `sc::tm_get_schema_names()`. These schemas were included in the skeleton. Note that schemas beginning with `config_*` are special schemas that are automatically generated by `sc`.

```
sc::tm_get_schema_names()
## [1] "config_last_updated"                                    "config_structu
## [3] "rundate"                                                "config_datetim
## [5] "anon_example_weather_rawdata"                           "anon_example_w
## [7] "anon_example_income"                                    "anon_example_h
## [9] "anon_example_house_prices_outliers_after_adjusting_for_income"
```

When you do this you will not see the schemas related to weather, income, and houseprices and you might see a Warning on the form of "In setup_ns_exports(path, export_all, export_imports) : Objects listed as exports. . . .". This is as expected.

You can also see which tasks have been loaded by running `sc::tm_get_task_names()`. These tasks were included in the skeleton. We have not yet made any tasks, hence you will see NULL.

```
sc::tm_get_task_names()
## [1] "weather_download_and_import_rawdata"                    "weather_clean_
## [3] "weather_export_plots"                                   "household_inc
## [5] "household_incomes_and_house_prices_fit_model_and_find_outliers" "household_inc
```

4.3 Weather data example

We are now going to create a weather data example. Our end goal is to plot the minimum and maximal temperature of all counties in Norway. This involves a task for downloading and importing raw data. For this we need to specify a schema which describes the data we want to store, which data identifies unique rows and who has access to the data. We also need to define the task through a task definition, i.e., task name, how many cores we want to use, the structure of the task, common arguments etc. Finally we actually implement the task by writing a data selector function, an action function and sometimes a more detailed function describing the plans and analyses of the task.

We also create a task for cleaning the raw data, again with a schema, a task definition and an implementation of a data selector function and an action function.

Finally we create a task for the creation of the plots.

All the schemas are spesified in `03_db_schemas.r`, all the task definitions are specified in `04_tasks.r`. The data selector functions and the action functions corresponding to each task have there own respective script with the name specified in the task description.

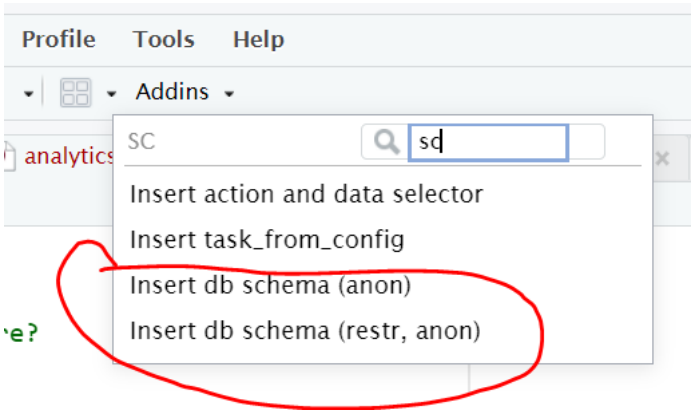
4.4 Developing weather_download_and_import_rawdata

We will walk you through the development of a task that downloads weather data from an API and imports the raw data into a database table.

4.4.1 1. Schemas

The first step when developing any task is specifying the schemas that will be used.

It is strongly recommended that you use the RStudio `Addins` menu to help you quickly insert code templates.



If you go into the script `03_db_schemas.r` you can see a function called `set_db_schemas`. All schemas are placed within this function. If you scroll down you can see that there is already a schema called `anon_example_weather_rawdata` which is commented out.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/03_db_schemas.r#L18-L64
##
## 18 |   ## > anon_example_weather_rawdata ----
## 19 |   sc::add_schema_v8(
## 20 |     name_access = c("anon"),
```

```
## 21 |     name_grouping = "example_weather",
## 22 |     name_variant = "rawdata",
## 23 |     db_configs = sc::config$db_configs,
## 24 |     field_types =  c(
## 25 |       "granularity_time" = "TEXT",
## 26 |       "granularity_geo" = "TEXT",
## 27 |       "country_iso3" = "TEXT",
## 28 |       "location_code" = "TEXT",
## 29 |       "border" = "INTEGER",
## 30 |       "age" = "TEXT",
## 31 |       "sex" = "TEXT",
## 32 |
## 33 |       "date" = "DATE",
## 34 |
## 35 |       "isoyear" = "INTEGER",
## 36 |       "isoweek" = "INTEGER",
## 37 |       "isoyearweek" = "TEXT",
## 38 |       "season" = "TEXT",
## 39 |       "seasonweek" = "DOUBLE",
## 40 |
## 41 |       "calyear" = "INTEGER",
## 42 |       "calmonth" = "INTEGER",
## 43 |       "calyearmonth" = "TEXT",
## 44 |
## 45 |       "temp_max" = "DOUBLE",
## 46 |       "temp_min" = "DOUBLE",
## 47 |       "precip" = "DOUBLE"
## 48 |     ),
## 49 |     keys = c(
## 50 |       "granularity_time",
## 51 |       "location_code",
## 52 |       "date",
## 53 |       "age",
## 54 |       "sex"
## 55 |     ),
## 56 |     censors = list(
## 57 |       anon = list(
## 58 |
## 59 |       )
## 60 |     ),
## 61 |     validator_field_types = sc::validator_field_types_sykdomspulsen,
## 62 |     validator_field_contents = sc::validator_field_contents_sykdomspulsen,
## 63 |     info = "This db table is used for..."
## 64 |   )
```

We are now going to recreate this schema. Make sure your pointer is inside of the

curly brackets. Go to the `Addins` menu and click `Insert db schema (anon)`. You have now created a boiler plate for your schema.

4.4.1.1 Schema name

Start by replacing `GROUPING_VARIANT` in `anon_GROUPING_VARIANT` with the name of your schema. For example `example_weather_rawdata`. The grouping will now be `example_weather` and the variant is `rawdata`.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/03_db_schemas.r#L20-L22
##
## 20 |     name_access = c("anon"),
## 21 |     name_grouping = "example_weather",
## 22 |     name_variant = "rawdata",
```

Fill this inn for `name_grouping` and `name_variant`. The name of the schema is then `anon_example_weather_rawdata`.

In the example we define the name of the schema to be `anon_example_weather_weather_rawdata`.

4.4.1.2 Validators

The validators are pre-made and you do not have to change anything.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/03_db_schemas.r#L61-L63
##
## 61 |     validator_field_types = sc::validator_field_types_sykdomspulsen,
## 62 |     validator_field_contents = sc::validator_field_contents_sykdomspulsen,
## 63 |     info = "This db table is used for..."
```

These are validators that check:

- Are the column names/field types in the schema definition in line with style guidelines?
- Are the values/field contents of the datasets that will be uploaded to the database correct? E.g. Does a date column actually contain dates?

When using `validator_field_types = sc::validator_field_types_sykdomspulsen` we expect that the first 16 columns are always as follows (i.e. standardized structural data):

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/03_db_schemas.r#L25-L43
##
## 25 |       "granularity_time" = "TEXT",
## 26 |       "granularity_geo" = "TEXT",
## 27 |       "country_iso3" = "TEXT",
## 28 |       "location_code" = "TEXT",
## 29 |       "border" = "INTEGER",
## 30 |       "age" = "TEXT",
## 31 |       "sex" = "TEXT",
```

```
## 32 |
## 33 |       "date" = "DATE",
## 34 |
## 35 |       "isoyear" = "INTEGER",
## 36 |       "isoweek" = "INTEGER",
## 37 |       "isoyearweek" = "TEXT",
## 38 |       "season" = "TEXT",
## 39 |       "seasonweek" = "DOUBLE",
## 40 |
## 41 |       "calyear" = "INTEGER",
## 42 |       "calmonth" = "INTEGER",
## 43 |       "calyearmonth" = "TEXT",
```

The field `info` should contain a short description of the data table.

4.4.1.3 Field types/column names

Add the spesific column names and types needed. In our case we want to store the maximum and minimum temperature and the precipitation. Call them "temp_max", "temp_min", and "precip". These are all "DOUBLE". Remove `"XXXX_n" = "INTEGER"`, and `"XXXX_pr" = "DOUBLE"`as these are dummy variables.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/03_db_schemas.r#L45
##
## 45 |       "temp_max" = "DOUBLE",
## 46 |       "temp_min" = "DOUBLE",
## 47 |       "precip" = "DOUBLE"
```

These are the extra columns that contain the context-specific data in this dataset.

4.4.1.4 Keys

The combination of these columns represents a unique row in the dataset. In this dataset the combination of "granularity_geo", "location_code", "date", "age", "sex" which are the initial suggestions are sufficient.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/03_db_schemas.r#L49
##
## 49 |     keys = c(
## 50 |       "granularity_time",
## 51 |       "location_code",
## 52 |       "date",
## 53 |       "age",
## 54 |       "sex"
## 55 |     ),
```

4.4.1.5 Censoring

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/03_db_schemas.r#L56-L60
##
## 56 |     censors = list(
## 57 |       anon = list(
## 58 |
## 59 |       )
## 60 |     ),
```

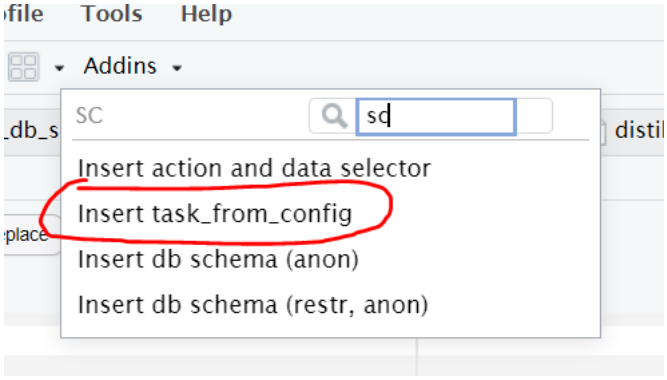
Censoring that is applied to the datasets. In this example we do not apply censoring hence remove the boiler plate suggestions.

4.4.2 2. Task definition (task_from_config)

Now we have a schema. The second step is defining the task.

Again it is strongly recommended that you use the RStudio `Addins` menu to help you quickly insert code templates.

Go to script `04_tasks.r` and place your curser inside of the curly bracket. Use the addins menu and click `Insert task_from_config`.



Now you have a boilerplate for a task definition.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/04_tasks.r#L21-L43
##
## 21 |   sc::add_task_from_config_v8(
## 22 |     name_grouping = "weather",
## 23 |     name_action = "download_and_import_rawdata",
## 24 |     name_variant = NULL,
## 25 |     cores = 1,
## 26 |     plan_analysis_fn_name = NULL,
## 27 |     for_each_plan = plnr::expand_list(
## 28 |       location_code = fhidata::norway_locations_names()[granularity_geo %in% c("municip")
## 29 |     ),
```

```
## 30 |     for_each_analysis = NULL,
## 31 |     universal_argset = NULL,
## 32 |     upsert_at_end_of_each_plan = FALSE,
## 33 |     insert_at_end_of_each_plan = FALSE,
## 34 |     action_fn_name = "scexample::weather_download_and_import_rawdata_action",
## 35 |     data_selector_fn_name = "scexample::weather_download_and_import_rawdata_dat
## 36 |     schema = list(
## 37 |       # input
## 38 |
## 39 |       # output
## 40 |       "anon_example_weather_rawdata" = sc::config$schemas$anon_example_weather_
## 41 |     ),
## 42 |     info = "This task downloads and imports the raw weather data from MET's API
## 43 |   )
```

4.4.2.1 Task name

Replace `TASK_NAME` by your task name. For example `weather_download_and_import_rawdata`. `weather` is the task/grouping and `download_and_import_rawdata`is the action name. Insert these for `name_grouping`, and `name_action`. For `name_variant` use `NULL`.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/04_tasks.r#L22-L24
##
## 22 |     name_grouping = "weather",
## 23 |     name_action = "download_and_import_rawdata",
## 24 |     name_variant = NULL,
```

Now the name of the task is defined to be `weather_download_and_import_rawdata`.

4.4.2.2 CPU cores

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/04_tasks.r#L25-L25
##
## 25 |     cores = 1,
```

We specify that the plans will run sequentially with 1 CPU core. If the number of CPU cores is 2 or higher then the first and last plans will run sequentially, and all the plans in the middle will run in parallel. The first and last plans always run sequentially because this allows us to write "special" code for the first and last plans (i.e. "do this before everything runs" and "do this after everything runs").

4.4.2.3 Plan/analysis structure

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/04_tasks.r#L26-L30
##
## 26 |     plan_analysis_fn_name = NULL,
```

```
## 27 |     for_each_plan = plnr::expand_list(
## 28 |       location_code = fhidata::norway_locations_names()[granularity_geo %in% c("municip")
## 29 |     ),
## 30 |     for_each_analysis = NULL,
```

We specify the plan/analysis structure here. You may use one of the following combinations:

- `plan_analysis_fn_name` (rarely used)
- `for_each_plan` (plan-heavy, one analysis per plan)
- `for_each_plan` + `for_each_analysis` (typically analysis-heavy)

`plan_analysis_fn_name` is a (rarely used) function that will provide a list containing the plan/analysis structure. It is generally only used when the plan/analysis structure needs to be reactive depending upon some external data (e.g. "an unknown number of data files are provided each day and need to be cleaned").

`for_each_plan` is a list, with each element corresponding to a plan defined by a named list. Within this named list, each of the named elements will be translated into argset elements that are available for the respective plans. This particular `for_each_plan` defines a task with 356 plans (one for each municipality).

`for_each_analysis` is nearly the same as `for_each_plan`. It specifies what kind of analyses you would like to perform within each plan. It is a named list, with each element corresponding to an analysis defined by a named list. Within this named list, each of the named elements will be translated into argset elements that are available for the respective analyses.

An example of a `for_each_plan` that would correspond to 11 tasks (one for each county):

```
options(width = 150)
for_each_plan = plnr::expand_list(
  location_code = fhidata::norway_locations_names()[granularity_geo %in% c("county")]$location_co
)
for_each_plan
## [[1]]
## [[1]]$location_code
## [1] "county42"
##
##
## [[2]]
## [[2]]$location_code
## [1] "county34"
##
##
## [[3]]
## [[3]]$location_code
```

```
## [1] "county15"
##
##
## [[4]]
## [[4]]$location_code
## [1] "county18"
##
##
## [[5]]
## [[5]]$location_code
## [1] "county03"
##
##
## [[6]]
## [[6]]$location_code
## [1] "county11"
##
##
## [[7]]
## [[7]]$location_code
## [1] "county54"
##
##
## [[8]]
## [[8]]$location_code
## [1] "county50"
##
##
## [[9]]
## [[9]]$location_code
## [1] "county38"
##
##
## [[10]]
## [[10]]$location_code
## [1] "county46"
##
##
## [[11]]
## [[11]]$location_code
## [1] "county30"
```

`fhidata::norway_locations_names()` gives us location codes in Norway (try and run if in your console). Implement a plan in your task which has the location codes off all municipalities (municip) in Norway.

4.4.3 Universal argset

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/04_tasks.r#L31-L31
##
## 31 |     universal_argset = NULL,
```

Here we can specify a named list, where each of the named elements will be translated into argset elements that are available for all plans/analyses.

4.4.3.1 Upsert/insert at end of each plan

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/04_tasks.r#L32-L33
##
## 32 |     upsert_at_end_of_each_plan = FALSE,
## 33 |     insert_at_end_of_each_plan = FALSE,
```

If you include a schema called `output`, then these options will let you upsert/insert the returned value from `action_fn_name` at the end of each plan. This is an important nuance, because when you write/develop your task, you can (typically) only write one function (`action_fn_name`) that is applied to all analyses. This means that if your `action_fn` wants to upsert/insert data to a schema, it (typically) will do this within *every* analysis. If you have an analysis-heavy task, then this will be a lot of frequent traffic to the databases, which may affect performance. By using these flags, you can restrict the upsert/insert to the end of the plan, which may increase performance.

4.4.3.2 action_fn_name

The action_fn_name specifies the name of the function that corresponds to the action. That is, the function that is called in every analysis. Note that:

- This is a string
- It must include the package name
- It is typically of the form `PACKAGE::TASK_action`

In our case the package is `scskeleton` and the TASK is `weather_download_and_import_rawdata`. Insert this in your task.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/04_tasks.r#L34-L34
##
## 34 |     action_fn_name = "scexample::weather_download_and_import_rawdata_action",
```

4.4.3.3 data_selector_fn_name

The data_selecotr_fn_name specifies the name of the function that corresponds to the data selector. That is, the function that is called at the start of every plan to provide data to all of the analyses inside the plan. Note that:

- This is a string
- It must include the package name

- It is typically of the form `PACKAGE::TASK_data_selector`

Try and guess what this would be in our example.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/04_tasks.r#L35-L35
##
## 35 |     data_selector_fn_name = "scexample::weather_download_and_import_rawdata_da
```

4.4.3.4 Schemas

The schemas specify a named list, where each element consists of a schema. The names will be passed through as `schema$name` in `action_fn_name` and `data_selector_fn_name`. We must include both the schemas where we get data from and the schemas we store data to.

In our example we do not yet have data so we only specify the schema we have earlier which we called `anon_example_weather_rawdata`. This meand you can remove the boiler plate input schema and replace `SCHEMA_NAME_2` with `anon_example_weather_rawdata`.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/04_tasks.r#L36-L41
##
## 36 |     schema = list(
## 37 |       # input
## 38 |
## 39 |       # output
## 40 |       "anon_example_weather_rawdata" = sc::config$schemas$anon_example_weather_
## 41 |     ),
```

4.4.3.5 Task description

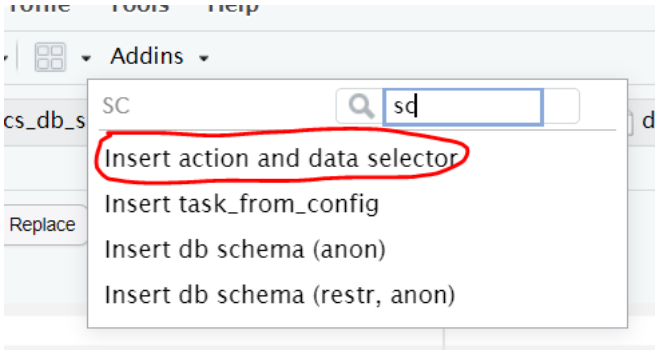
Finally create a small task description.

4.4.4 3. data_selector_fn

The third step in creating a task is defining a data selector function. This is the function that will perform the "one data-pull per plan" and subsequently provide the data to the action.

Go to script `weather_download_and_import_rawdata.r`.

Use the RStudio `Addins` menu to help you quickly insert code templates by clicking `Insert action and data selector`.



Just like that, a pre-made boilerplate is ready to go! Find the data_selector part of the script and replace `TASK_NAME` with our task name `weather_download_and_import_rawdata`.

4.4.4.1 plnr::is_run_directly()

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_download_and_import_r
##
##  94 |   if (plnr::is_run_directly()) {
##  95 |     # sc::tm_get_plans_argsets_as_dt("weather_download_and_import_rawdata")
##  96 |
##  97 |     index_plan <- 1
##  98 |
##  99 |     argset <- sc::tm_get_argset("weather_download_and_import_rawdata", index_plan = inde
## 100 |     schema <- sc::tm_get_schema("weather_download_and_import_rawdata")
## 101 |   }
```

At the top of all `data_selector_fn`s you will see a section of code wrapped inside `if (plnr::is_run_directly()) {`. This code will **only** be run if it is manually highlighted inside RStudio and then "run". This is extremely beneficial to the user, because it means that the user can easily write small pieces of code that are only used during development, which will not be run when the code is run "properly".

Sykdomspulsen core uses these sections to let the user "jump" directly into the function. Look at the arguments for `weather_download_and_import_rawdata_data_selector` and you will see that it needs `argset` and `schema`.

The code inside `if (plnr::is_run_directly()) {` loads `argset` and `schema` for `index_plan = 1`. **By running these lines, you can treat the inside of `weather_download_and_import_rawdata_data_selector` as an interactive script!**

This makes the development of the code extremely easy as "everything is an interactive script".

Check that you have an argset and a schema by running the lines within `if (plnr::is_run_directly()) {}`.

4.4.5 Getting data

The majority of the `data_selector_fn` is concerned with selecting data (obviously). Remember that the data should be selected to meet the needs of the plan. If you have 11 plans (one for each county), then your `data_selector_fn` should only extract data for the county of interest.

Take a look at your argset for plan = 1. Since we do not have input data from a schema we can remove the premade schema. Instead we are going to get data from `fhimaps::norway_lau2_map_b2020_default_dt` which provides latitudes (lat) and longitudes (long). Explore the available data by running `fhimaps::norway_lau2_map_b2020_default_dt` in your console. It returns a data table. We only want the mean latitude and longitude of the specific location_code for this particular plan and analysis. Therefor try and select only this data and call it `gps`.

Now we download the weather forcast for this specific location from an api by using httr::GET and glue::glue to get the right address.

httr::GET(glue::glue("https://api.met.no/weatherapi/locationforecast/2.0/classic?lat=%7Bgps$lat%7D&lon=%7Bgps$long%7D"), httr::content_type_xml()). Use xml2::read_xml() to read the content. Take a peak below and implement it yourself.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_download_ar
##
## 103 |   # find the mid lat/long for the specified location_code
## 104 |   gps <- fhimaps::norway_lau2_map_b2020_default_dt[location_code == argset$loc
## 105 |     lat = mean(lat),
## 106 |     long = mean(long)
## 107 |   )]
## 108 |
## 109 |   # download the forecast for the specified location_code
## 110 |   d <- httr::GET(glue::glue("https://api.met.no/weatherapi/locationforecast/2
## 111 |   d <- xml2::read_xml(d$content)
```

4.4.6 Returning data

`data_selector_fn` needs to return a named list. This will be made available to the user in `action_fn` (`weather_download_and_import_rawdata_action`) via the argument `data`.

In your task replace "NAME" by the name for your data for example "data".

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_download_ar
##
## 113 |   # The variable returned must be a named list
## 114 |   retval <- list(
## 115 |     "data" = d
```

```
## 116 |   )
```

The entire data selector function should now look like this.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_download_and_import_r
##
##  88 | # **** data_selector **** ----
##  89 | #' weather_download_and_import_rawdata (data selector)
##  90 | #' @param argset Argset
##  91 | #' @param schema DB Schema
##  92 | #' @export
##  93 | weather_download_and_import_rawdata_data_selector <- function(argset, schema) {
##  94 |   if (plnr::is_run_directly()) {
##  95 |     # sc::tm_get_plans_argsets_as_dt("weather_download_and_import_rawdata")
##  96 |
##  97 |     index_plan <- 1
##  98 |
##  99 |     argset <- sc::tm_get_argset("weather_download_and_import_rawdata", index_plan = inde
## 100 |     schema <- sc::tm_get_schema("weather_download_and_import_rawdata")
## 101 |   }
## 102 |
## 103 |   # find the mid lat/long for the specified location_code
## 104 |   gps <- fhimaps::norway_lau2_map_b2020_default_dt[location_code == argset$location_code
## 105 |     lat = mean(lat),
## 106 |     long = mean(long)
## 107 |   )]
## 108 |
## 109 |   # download the forecast for the specified location_code
## 110 |   d <- httr::GET(glue::glue("https://api.met.no/weatherapi/locationforecast/2.0/classic?
## 111 |   d <- xml2::read_xml(d$content)
## 112 |
## 113 |   # The variable returned must be a named list
## 114 |   retval <- list(
## 115 |     "data" = d
## 116 |   )
## 117 |
## 118 |   retval
## 119 | }
```

Check that the data selector function works by restarting R (`ctrl + shift + F10`) and loading the packages (`ctrl + shift + L`) before running through the data selector function line by line.

4.4.7 4. action_fn

The fourth step is defining an action function. This is the function that will perform the "action" within the the analysis. That is, given:

- data
- argset
- schema

What do you actually want to *do* with them? Find the action part in your script and replace `TASK_NAME` with our task name `weather_download_and_import_rawdata`.

4.4.7.1 plnr::is_run_directly()

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_download_a
##
##  95 |     # sc::tm_get_plans_argsets_as_dt("weather_download_and_import_rawdata")
##  96 |
##  97 |     index_plan <- 1
##  98 |
##  99 |     argset <- sc::tm_get_argset("weather_download_and_import_rawdata", index_
## 100 |     schema <- sc::tm_get_schema("weather_download_and_import_rawdata")
## 101 |   }
## 102 |
```

At the top of all `action_fn`s you will again see a section of code wrapped inside `if (plnr::is_run_directly()) {`. This works exactly the same as for the data_selector_fn.

Look at the arguments for `weather_download_and_import_rawdata_data_selector` and you will see that it needs `data`, `argset` and `schema`. The code inside `if (plnr::is_run_directly()) {` loads `data`, `argset` and `schema` for `index_plan = 1` and `index_analysis = 1`. **By running these lines, you can treat the inside of `weather_download_and_import_rawdata_action` as an interactive script!**

Check out the data, argset and schema you have by running these lines.

4.4.7.2 argset$first_analysis

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_download_a
##
## 21 |   # special case that runs before everything
## 22 |   if (argset$first_analysis == TRUE) {
## 23 |
## 24 |   }
```

This code is only run if it is the first analysis. It is typically used to drop rows in a database, so that the following code may `insert` data (faster) instead of using `upsert` data (slower). If you ran the full task at the beginning of this tutorial you can insert `schema$anon_example_weather_rawdata$drop_all_rows()` inside here to delete the stored data.

4.4.7.3 Doing things

In this tutorial we do not go in to much detail about how the data is collected so for now copy the content of the action function into your action function. (You find it commented out in your file.)

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_download_and_import_r
##
## 26 |   a <- data$data
## 27 |
## 28 |   baz <- xml2::xml_find_all(a, ".//maxTemperature")
## 29 |   res <- vector("list", length = length(baz))
## 30 |   for (i in seq_along(baz)) {
## 31 |     parent <- xml2::xml_parent(baz[[i]])
## 32 |     grandparent <- xml2::xml_parent(parent)
## 33 |     time_from <- xml2::xml_attr(grandparent, "from")
## 34 |     time_to <- xml2::xml_attr(grandparent, "to")
## 35 |     x <- xml2::xml_find_all(parent, ".//minTemperature")
## 36 |     temp_min <- xml2::xml_attr(x, "value")
## 37 |     x <- xml2::xml_find_all(parent, ".//maxTemperature")
## 38 |     temp_max <- xml2::xml_attr(x, "value")
## 39 |     x <- xml2::xml_find_all(parent, ".//precipitation")
## 40 |     precip <- xml2::xml_attr(x, "value")
## 41 |     res[[i]] <- data.frame(
## 42 |       time_from = as.character(time_from),
## 43 |       time_to = as.character(time_to),
## 44 |       temp_max = as.numeric(temp_max),
## 45 |       temp_min = as.numeric(temp_min),
## 46 |       precip = as.numeric(precip)
## 47 |     )
## 48 |   }
## 49 |   res <- rbindlist(res)
## 50 |   res <- res[stringr::str_sub(time_from, 12, 13) %in% c("00", "06", "12", "18")]
## 51 |   res[, date := as.Date(stringr::str_sub(time_from, 1, 10))]
## 52 |   res[, N := .N, by = date]
## 53 |   res <- res[N == 4]
## 54 |   res <- res[
## 55 |     ,
## 56 |     .(
## 57 |       temp_max = max(temp_max),
## 58 |       temp_min = min(temp_min),
## 59 |       precip = sum(precip)
## 60 |     ),
## 61 |     keyby = .(date)
## 62 |   ]
## 63 |
```

```
## 64 |   # we look at the downloaded data
## 65 |   # res
## 66 |
## 67 |   # we now need to format it
## 68 |   res[, granularity_time := "day"]
## 69 |   res[, sex := "total"]
## 70 |   res[, age := "total"]
## 71 |   res[, location_code := argset$location_code]
## 72 |
## 73 |   # fill in missing structural variables
## 74 |   sc::fill_in_missing_v8(res, border = 2020)
## 75 |
## 76 |   # we look at the downloaded data
## 77 |   # res
## 78 |
## 79 |   # put data in db table
## 80 |   schema$anon_example_weather_rawdata$insert_data(res)
```

Every analysis will perform this code.

Run through it line by line and pay special attention to how the data from the data_selector_fn is accesed, the last part where the data is formatted and we use `sc::fill_in_missing_v8(res, border = 2020)` to fill inn the mandatory data columns and the end where the data is inserted to the database.

4.4.7.4 Accessing data from data_selector_fn

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_download_a
##
## 26 |   a <- data$data
```

Here you see that we access the data that was passed to us from `data_selector_fn`

4.4.7.5 Structural data/sc::fill_in_missing_v8

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_download_a
##
## 68 |   res[, granularity_time := "day"]
## 69 |   res[, sex := "total"]
## 70 |   res[, age := "total"]
## 71 |   res[, location_code := argset$location_code]
## 72 |
## 73 |   # fill in missing structural variables
## 74 |   sc::fill_in_missing_v8(res, border = 2020)
```

We have 16 structural data columns that we expect. These columns typically have a lot of redundancy (e.g. date, isoyear, isoyearweek). To make things easier,

we provide a function called `sc::fill_in_missing_v8` that uses the information present in the dataset to try and impute the missing structural data.

4.4.7.6 Insert/upsert to databases

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_download_and_import_r
##
## 80 |   schema$anon_example_weather_rawdata$insert_data(res)
```

Here we insert the data to the database table.

Insert is an append (so the data cannot already exist in the database table), while upsert is "update (overwrite) if already exists, insert (append) if it doesn't".

If you want to deleate the data use `schema$NAME_DATABASE$drop_all_rows()` in our case `schema$anon_example_weather_rawdata$drop_all_rows()`.

4.4.7.7 argset$last_analysis

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_download_and_import_r
##
## 21 |   # special case that runs before everything
## 22 |   if (argset$first_analysis == TRUE) {
## 23 |
## 24 |   }
```

This code is only run if it is the last analysis. It is typically used to copy an internal database table (i.e. one that the public is not directly viewing) to an external database (i.e. one that the public is directly viewing).

By distinguishing between internal database tables (e.g. anon_webkhtint_test) and external database tables (e.g. anon_webkht_test) we can do whatever we want to anon_webkhtint_test while anon_webkht_test remains in place and untouched. This makes it less likely that any mistakes will affect any APIs or websites that the public uses.

4.4.8 Test the code

The action function should look like this.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_download_and_import_r
##
##  1 | # **** action **** ----
##  2 | #' weather_download_and_import_rawdata (action)
##  3 | #' @param data Data
##  4 | #' @param argset Argset
##  5 | #' @param schema DB Schema
##  6 | #' @export
##  7 | weather_download_and_import_rawdata_action <- function(data, argset, schema) {
```

```
##  8 |   # tm_run_task("weather_download_and_import_rawdata")
##  9 |
## 10 |   if (plnr::is_run_directly()) {
## 11 |     # sc::tm_get_plans_argsets_as_dt("weather_download_and_import_rawdata")
## 12 |
## 13 |     index_plan <- 1
## 14 |     index_analysis <- 1
## 15 |
## 16 |     data <- sc::tm_get_data("weather_download_and_import_rawdata", index_plan =
## 17 |     argset <- sc::tm_get_argset("weather_download_and_import_rawdata", index_pl
## 18 |     schema <- sc::tm_get_schema("weather_download_and_import_rawdata")
## 19 |   }
## 20 |
## 21 |   # special case that runs before everything
## 22 |   if (argset$first_analysis == TRUE) {
## 23 |
## 24 |   }
## 25 |
## 26 |   a <- data$data
## 27 |
## 28 |   baz <- xml2::xml_find_all(a, ".//maxTemperature")
## 29 |   res <- vector("list", length = length(baz))
## 30 |   for (i in seq_along(baz)) {
## 31 |     parent <- xml2::xml_parent(baz[[i]])
## 32 |     grandparent <- xml2::xml_parent(parent)
## 33 |     time_from <- xml2::xml_attr(grandparent, "from")
## 34 |     time_to <- xml2::xml_attr(grandparent, "to")
## 35 |     x <- xml2::xml_find_all(parent, ".//minTemperature")
## 36 |     temp_min <- xml2::xml_attr(x, "value")
## 37 |     x <- xml2::xml_find_all(parent, ".//maxTemperature")
## 38 |     temp_max <- xml2::xml_attr(x, "value")
## 39 |     x <- xml2::xml_find_all(parent, ".//precipitation")
## 40 |     precip <- xml2::xml_attr(x, "value")
## 41 |     res[[i]] <- data.frame(
## 42 |       time_from = as.character(time_from),
## 43 |       time_to = as.character(time_to),
## 44 |       temp_max = as.numeric(temp_max),
## 45 |       temp_min = as.numeric(temp_min),
## 46 |       precip = as.numeric(precip)
## 47 |     )
## 48 |   }
## 49 |   res <- rbindlist(res)
## 50 |   res <- res[stringr::str_sub(time_from, 12, 13) %in% c("00", "06", "12", "18")
## 51 |   res[, date := as.Date(stringr::str_sub(time_from, 1, 10))]
## 52 |   res[, N := .N, by = date]
## 53 |   res <- res[N == 4]
```

```
## 54 |   res <- res[
## 55 |     ,
## 56 |     .(
## 57 |       temp_max = max(temp_max),
## 58 |       temp_min = min(temp_min),
## 59 |       precip = sum(precip)
## 60 |     ),
## 61 |     keyby = .(date)
## 62 |   ]
## 63 |
## 64 |   # we look at the downloaded data
## 65 |   # res
## 66 |
## 67 |   # we now need to format it
## 68 |   res[, granularity_time := "day"]
## 69 |   res[, sex := "total"]
## 70 |   res[, age := "total"]
## 71 |   res[, location_code := argset$location_code]
## 72 |
## 73 |   # fill in missing structural variables
## 74 |   sc::fill_in_missing_v8(res, border = 2020)
## 75 |
## 76 |   # we look at the downloaded data
## 77 |   # res
## 78 |
## 79 |   # put data in db table
## 80 |   schema$anon_example_weather_rawdata$insert_data(res)
## 81 |
## 82 |   # special case that runs after everything
## 83 |   if (argset$last_analysis == TRUE) {
## 84 |
## 85 |   }
## 86 | }
```

Try and restart, load all and run the code line by line.

4.4.9 Which plan/analysis is which?

Inside the `if (plnr::is_run_directly()) {` sections, you specify `index_plan` and `index_analysis`. However, these are just numbers. If you want to specifically look at the plan for Oslo municipality, how do you know which `index_plan` this corresponds to?

```
options(width = 150)
sc::tm_get_plans_argsets_as_dt("weather_download_and_import_rawdata")
##      index_plan index_analysis **universal** **plan** location_code **analysis** **automatic**
##   1:          1              1             *        *   municip1820            *             *
```

```
##   2:          2             1             *        *   municip5403            *
##   3:          3             1             *        *   municip3428            *
##   4:          4             1             *        *   municip4631            *
##   5:          5             1             *        *   municip1871            *
##  ---
## 352:        352             1             *        *   municip3442            *
## 353:        353             1             *        *   municip3048            *
## 354:        354             1             *        *   municip3440            *
## 355:        355             1             *        *   municip4626            *
## 356:        356             1             *        *   municip3453            *
##      first_argset last_analysis last_argset
##   1:         TRUE         FALSE       FALSE
##   2:        FALSE         FALSE       FALSE
##   3:        FALSE         FALSE       FALSE
##   4:        FALSE         FALSE       FALSE
##   5:        FALSE         FALSE       FALSE
##  ---
## 352:        FALSE         FALSE       FALSE
## 353:        FALSE         FALSE       FALSE
## 354:        FALSE         FALSE       FALSE
## 355:        FALSE         FALSE       FALSE
## 356:        FALSE          TRUE        TRUE
```

Try and change the plan number and run the script again.

Now you have implemented your first task by creating a schema, a task description a data selector function and an action function! Congratulations!

Run the entire task by running `tm_run_task("weather_download_and_import_rawdata")`.

4.5 Developing weather_clean_data

The previous task (weather_download_and_import_rawdata) focused on downloading raw data from an API and inserting it into a database table.

The task weather_clean_data focuses on cleaning the raw data and inserting it in another database table. That is, the data source is a Sykdomspulsen Core database table, and the output is also a Sykdomspulsen Core database table.

We will walk you through the development of weather_clean_data, however, the description of this task will be less comprehensive than the previous task, and will focus primarily on parts that are novel.

We already mentioned that weather_clean_data cleans the raw data. We want this task to take the raw data we obtained in the previous task and aggregate it to obtain weather data on different geographical regions than municipalities. To do so we use some pre-made FHI functions such as

`fhidata::make_skeleton` which makes a data table skeleton for the regions of interest and `fhidata::norway_locations_hierarchy` which converts location codes from one location code level to another.

4.5.1 1. Schemas

First we start by creating a schema for the data we want this task to store. The structure is exactly the same as for the previous task with name access anon and temp_max, temp_min and precip as additional colums. Try and create this schema in `03_db_schemas.r`.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/03_db_schemas.r#L66-L113
##
##  66 |   ## > anon_example_weather_data ----
##  67 |   sc::add_schema_v8(
##  68 |     name_access = c("anon"),
##  69 |     name_grouping = "example_weather",
##  70 |     name_variant = "data",
##  71 |     db_configs = sc::config$db_configs,
##  72 |     field_types =  c(
##  73 |       "granularity_time" = "TEXT",
##  74 |       "granularity_geo" = "TEXT",
##  75 |       "country_iso3" = "TEXT",
##  76 |       "location_code" = "TEXT",
##  77 |       "border" = "INTEGER",
##  78 |       "age" = "TEXT",
##  79 |       "sex" = "TEXT",
##  80 |
##  81 |       "date" = "DATE",
##  82 |
##  83 |       "isoyear" = "INTEGER",
##  84 |       "isoweek" = "INTEGER",
##  85 |       "isoyearweek" = "TEXT",
##  86 |       "season" = "TEXT",
##  87 |       "seasonweek" = "DOUBLE",
##  88 |
##  89 |       "calyear" = "INTEGER",
##  90 |       "calmonth" = "INTEGER",
##  91 |       "calyearmonth" = "TEXT",
##  92 |
##  93 |       "temp_max" = "DOUBLE",
##  94 |       "temp_min" = "DOUBLE",
##  95 |       "precip" = "DOUBLE"
##  96 |     ),
##  97 |     keys = c(
##  98 |       "granularity_time",
```

```
##  99 |       "location_code",
## 100 |       "date",
## 101 |       "age",
## 102 |       "sex"
## 103 |     ),
## 104 |     censors = list(
## 105 |       anon = list(
## 106 |
## 107 |       )
## 108 |     ),
## 109 |     validator_field_types = sc::validator_field_types_sykdomspulsen,
## 110 |     validator_field_contents = sc::validator_field_contents_sykdomspulsen,
## 111 |     info = "This db table is used for..."
## 112 |   )
## 113 |
```

4.5.2 2. Task definition (task_from_config)

The next step is to define the task in `04_tasks.r` . For this task the aim is to aggregate data to higher levels meaning we need all data available at the same time and we preform the entire task in one analysis. Hence we need a task with only one plan (x = 1). We need the data from the database created in the previous task as input schema and the schema we just implemented as output schema. Try and create this task definition! (Remember to use the addins menu to get a boiler plate task definition.)

4.5.2.1 Plan/analysis structure

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/04_tasks.r#L52-L56
##
## 52 |     plan_analysis_fn_name = NULL,
## 53 |     for_each_plan = plnr::expand_list(
## 54 |       x = 1
## 55 |     ),
## 56 |     for_each_analysis = NULL,
```

For this particular task, we have decided to only implement one plan containing one analysis, which will process all of the data at once.

If we were only aggregating municipality data to the county level, we could have implemented 11 plans (one for each county). However, because we are also aggregating to the national level, we need all the data available at once.

4.5.2.2 Schemas

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/04_tasks.r#L62-L68
##
```

```
## 62 |     schema = list(
## 63 |       # input
## 64 |       "anon_example_weather_rawdata" = sc::config$schemas$anon_example_weather_rawdata,
## 65 |
## 66 |       # output
## 67 |       "anon_example_weather_data" = sc::config$schemas$anon_example_weather_data
## 68 |     ),
```

We need to specify the schemas that are used for both input and output.

4.5.2.3 Full task description

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/04_tasks.r#L45-L70
##
## 45 |   ## > weather_clean_data ----
## 46 |   # tm_run_task("weather_clean_data")
## 47 |   sc::add_task_from_config_v8(
## 48 |     name_grouping = "weather",
## 49 |     name_action = "clean_data",
## 50 |     name_variant = NULL,
## 51 |     cores = 1,
## 52 |     plan_analysis_fn_name = NULL,
## 53 |     for_each_plan = plnr::expand_list(
## 54 |       x = 1
## 55 |     ),
## 56 |     for_each_analysis = NULL,
## 57 |     universal_argset = NULL,
## 58 |     upsert_at_end_of_each_plan = FALSE,
## 59 |     insert_at_end_of_each_plan = FALSE,
## 60 |     action_fn_name = "scexample::weather_clean_data_action",
## 61 |     data_selector_fn_name = "scexample::weather_clean_data_data_selector",
## 62 |     schema = list(
## 63 |       # input
## 64 |       "anon_example_weather_rawdata" = sc::config$schemas$anon_example_weather_rawdata,
## 65 |
## 66 |       # output
## 67 |       "anon_example_weather_data" = sc::config$schemas$anon_example_weather_data
## 68 |     ),
## 69 |     info = "This task cleans the raw data and aggregates it to county and national level"
## 70 |   )
```

4.5.3 3. data_selector_fn

Now we are ready to create the data selector function. Go to script `weather_clean_data`. Use the addins menu as before to get a boiler plate for the action function and the data selector function and scroll down to the data selector part. Start by inserting your task name instead of `TASK_NAME`.

4.5.3.1 Getting data (specify the schema)

Next fill inn the name of the input schema instead of `SCHEMA_NAME`, connecting to the database table linked to the schema.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_clean_data
##
## 200 |   d <- schema$anon_example_weather_rawdata$tbl() %>%
```

4.5.3.2 Getting data (sc::mandatory_db_filter)

We then introduce the sc::mandatory_db_filter. This is a filter on the most common structural variables. We say this is "mandatory" because we want the user to always keep in mind:

- The minimal amount of data needed to do the job
- To be as explicit as possible with what data is needed to do the job

Fill inn the mandatory filters as best you can and take a peak below if you are not sure.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_clean_data
##
## 201 |     sc::mandatory_db_filter(
## 202 |       granularity_time = "day",
## 203 |       granularity_time_not = NULL,
## 204 |       granularity_geo = "municip",
## 205 |       granularity_geo_not = NULL,
## 206 |       country_iso3 = NULL,
## 207 |       location_code = NULL,
## 208 |       age = "total",
## 209 |       age_not = NULL,
## 210 |       sex = "total",
## 211 |       sex_not = NULL
## 212 |     ) %>%
```

You will notice that we don't use all of the arguments passed into the function, but we use as many as we can.

4.5.3.3 Getting data (dplyr::select)

We always want to be as explicit as possible with what data is needed to do the job. To achieve this, we use `dplyr::select` to select the columns that we are interested in.

If you want to quickly generate a `dplyr::select` boilerplate for your schema that you can copy/paste, you can do this via either of the following:

```
schema$anon_example_weather_rawdata$print_dplyr_select()
```

```
## dplyr::select(
##   granularity_time,
##   granularity_geo,
##   country_iso3,
##   location_code,
##   border,
##   age,
##   sex,
##   date,
##   isoyear,
##   isoweek,
##   isoyearweek,
##   season,
##   seasonweek,
##   calyear,
##   calmonth,
##   calyearmonth,
##   temp_max,
##   temp_min,
##   precip
## ) %>%
```

Use one of these functions and replace the dplyr::select part in your data selector function. To aggregate data we need location_code, date, temp_max, temp_min, precip, and the granularity_time (dayly, weekly, etc). Comment out the other variables.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_clean_data.r#L213-L23
##
## 213 |     dplyr::select(
## 214 |       granularity_time,
## 215 |       # granularity_geo,
## 216 |       # country_iso3,
## 217 |       location_code,
## 218 |       # border,
## 219 |       # age,
## 220 |       # sex,
## 221 |
## 222 |       date,
## 223 |
## 224 |       # isoyear,
## 225 |       # isoweek,
## 226 |       # isoyearweek,
## 227 |       # season,
## 228 |       # seasonweek,
## 229 |
## 230 |       # calyear,
```

```
## 231 |       # calmonth,
## 232 |       # calyearmonth,
## 233 |
## 234 |       temp_max,
## 235 |       temp_min,
## 236 |       precip
## 237 |     ) %>%
```

4.5.3.4 Getting data (dplyr::collect)

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_clean_data
##
## 238 |     dplyr::collect() %>%
```

This executes the SQL call to the database.

4.5.3.5 Getting data (data.table and setorder)

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_clean_data
##
## 239 |     as.data.table() %>%
## 240 |     setorder(
## 241 |       location_code,
## 242 |       date
## 243 |     )
```

Firstly, as a general rule we prefer to use data.table. So we would like to convert our data.frame to a data.table.

Secondly, we are not guaranteed to receive our data in any particular order. Because of this, it is very important that we sort our data on arrival (if this is relevant to the action_fn, e.g. if cumulative sums are created).

4.5.3.6 Set a name

Finally give the dataset you return a suitable name for example `day_municip`.

Check that you data selector function works by saving, restarting, and loading all packages. Then run through the function line by line.

4.5.3.7 Example of the data_selector function

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_clean_data
##
## 184 | # **** data_selector **** ----
## 185 | #' weather_clean_data (data selector)
## 186 | #' @param argset Argset
## 187 | #' @param schema DB Schema
## 188 | #' @export
```

```
## 189 | weather_clean_data_data_selector <- function(argset, schema) {
## 190 |   if (plnr::is_run_directly()) {
## 191 |     # sc::tm_get_plans_argsets_as_dt("weather_clean_data")
## 192 |
## 193 |     index_plan <- 1
## 194 |
## 195 |     argset <- sc::tm_get_argset("weather_clean_data", index_plan = index_plan)
## 196 |     schema <- sc::tm_get_schema("weather_clean_data")
## 197 |   }
## 198 |
## 199 |   # The database schemas can be accessed here
## 200 |   d <- schema$anon_example_weather_rawdata$tbl() %>%
## 201 |     sc::mandatory_db_filter(
## 202 |       granularity_time = "day",
## 203 |       granularity_time_not = NULL,
## 204 |       granularity_geo = "municip",
## 205 |       granularity_geo_not = NULL,
## 206 |       country_iso3 = NULL,
## 207 |       location_code = NULL,
## 208 |       age = "total",
## 209 |       age_not = NULL,
## 210 |       sex = "total",
## 211 |       sex_not = NULL
## 212 |     ) %>%
## 213 |     dplyr::select(
## 214 |       granularity_time,
## 215 |       # granularity_geo,
## 216 |       # country_iso3,
## 217 |       location_code,
## 218 |       # border,
## 219 |       # age,
## 220 |       # sex,
## 221 |
## 222 |       date,
## 223 |
## 224 |       # isoyear,
## 225 |       # isoweek,
## 226 |       # isoyearweek,
## 227 |       # season,
## 228 |       # seasonweek,
## 229 |
## 230 |       # calyear,
## 231 |       # calmonth,
## 232 |       # calyearmonth,
## 233 |
## 234 |       temp_max,
```

```
## 235 |       temp_min,
## 236 |       precip
## 237 |     ) %>%
## 238 |     dplyr::collect() %>%
## 239 |     as.data.table() %>%
## 240 |     setorder(
## 241 |       location_code,
## 242 |       date
## 243 |     )
## 244 |
## 245 |   # The variable returned must be a named list
## 246 |   retval <- list(
## 247 |     "day_municip" = d
## 248 |   )
## 249 |
## 250 |   retval
## 251 | }
```

4.5.4 4. action_fn

The final step in the process is creating the action function. Replace `TASK_NAME` with your task name.

4.5.4.1 Skeleton

In this action function we use fhi skeletons to create bases for our data tables. Read here about the concept of skeletons.

Start by creating a variable (for example d_agg) for an empty list and copy the data collected in the data selector function into this (`d_agg$day_municip <- copy(data$day_municip)`). Extract the first and last date from this dataset.

Now we are going to create a skeleton for where we separate between regions where we have data (municipalities) and regions where we do not have data (bo og arbeigs regioner). The skeleton function takes min and max dates and in this case we will pass it granularity_geo consisting of a list with

```
list(
          "nodata" = c(
          "wardoslo",
          "extrawardoslo",
          "missingwardoslo",
          "wardbergen",
          "missingwardbergen",
          "wardstavanger",
          "missingwardstavanger",
          "notmainlandmunicip",
```

```
          "missingmunicip",
          "notmainlandcounty",
          "missingcounty"
        ),
        "municip" = c(
          "municip"
        )
)
## $nodata
##  [1] "wardoslo"             "extrawardoslo"        "missingwardoslo"      "wardbergen"
##  [7] "missingwardstavanger" "notmainlandmunicip"   "missingmunicip"       "notmainlandcounty"
##
## $municip
## [1] "municip"
```

4.5.4.2 Merge in weather data

Next we want to merge the information we have on weather data for the municipalities into this data.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_clean_data.r#L67-L84
##
## 67 |   # Merge in the information you have at different geographical granularities
## 68 |   # one level at a time
## 69 |   # municip
## 70 |   multiskeleton_day$municip[
## 71 |     d_agg$day_municip,
## 72 |     on = c("location_code", "date"),
## 73 |     c(
## 74 |       "temp_max",
## 75 |       "temp_min",
## 76 |       "precip"
## 77 |     ) := .(
## 78 |       temp_max,
## 79 |       temp_min,
## 80 |       precip
## 81 |     )
## 82 |   ]
## 83 |
## 84 |   multiskeleton_day$municip[]
```

4.5.4.3 Aggregate to a county level

Now aggregate the data to a county level with the help of `fhidata::norway_locations_hierarchy`

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_clean_data.r#L86-L109
##
```

```
##  86 |   # Aggregate up to higher geographical granularities (county)
##  87 |   multiskeleton_day$county <- multiskeleton_day$municip[
##  88 |     fhidata::norway_locations_hierarchy(
##  89 |       from = "municip",
##  90 |       to = "county"
##  91 |     ),
##  92 |     on = c(
##  93 |       "location_code==from_code"
##  94 |     )
##  95 |   ][,
##  96 |     .(
##  97 |       temp_max = mean(temp_max, na.rm = T),
##  98 |       temp_min = mean(temp_min, na.rm = T),
##  99 |       precip = mean(precip, na.rm = T),
## 100 |       granularity_geo = "county"
## 101 |     ),
## 102 |     by = .(
## 103 |       granularity_time,
## 104 |       date,
## 105 |       location_code = to_code
## 106 |     )
## 107 |   ]
## 108 |
## 109 |   multiskeleton_day$county[]
```

4.5.4.4 Aggregate to national level

There is no overlap in municipalities, hence aggregating to a national level can be done without the help of `fhidata::norway_locations_hierarchy`.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_clean_data
##
## 111 |   # Aggregate up to higher geographical granularities (nation)
## 112 |   multiskeleton_day$nation <- multiskeleton_day$municip[
## 113 |     ,
## 114 |     .(
## 115 |       temp_max = mean(temp_max, na.rm = T),
## 116 |       temp_min = mean(temp_min, na.rm = T),
## 117 |       precip = mean(precip, na.rm = T),
## 118 |       granularity_geo = "nation",
## 119 |       location_code = "norge"
## 120 |     ),
## 121 |     by = .(
## 122 |       granularity_time,
## 123 |       date
## 124 |     )
```

```
## 125 |   ]
## 126 |
## 127 |   multiskeleton_day$nation[]
```

4.5.4.5 Combine data

Combine all the different granularity geos by using rbindlist and storing it to a new name f.eks skeleton_day.

4.5.4.6 Weekly data.

As a challenge try and aggregate the daily data to weekly data! You can use `fhiplot::isoyearweek_c(date)` to get the isoweek of a date.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_clean_data.r#L134-L15
##
## 134 |   # 10. (If desirable) aggregate up to higher time granularities
## 135 |   # if necessary, it is now easy to aggregate up to weekly data from here
## 136 |   skeleton_isoweek <- copy(skeleton_day)
## 137 |   skeleton_isoweek[, isoyearweek := fhiplot::isoyearweek_c(date)]
## 138 |   skeleton_isoweek <- skeleton_isoweek[
## 139 |     ,
## 140 |     .(
## 141 |       temp_max = mean(temp_max, na.rm = T),
## 142 |       temp_min = mean(temp_min, na.rm = T),
## 143 |       precip = mean(precip, na.rm = T),
## 144 |       granularity_time = "isoweek"
## 145 |     ),
## 146 |     keyby = .(
## 147 |       isoyearweek,
## 148 |       granularity_geo,
## 149 |       location_code
## 150 |     )
## 151 |   ]
## 152 |
## 153 |   skeleton_isoweek[]
```

4.5.4.7 Structural data

The next step is to fill in all missing structural data. Fill in sex = "total" and age= "total" manually then you can use `sc::fill_in_missing_v8(skeleton_day, border = config$border)`. For the weekly data make sure to also convert the date by using `as.Date(date)` to ensure it is on the right format.

Rbindlist binds the two data tables together.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_clean_data.r#L155-L17
##
```

```
## 155 |   # we now need to format it and fill in missing structural variables
## 156 |   # day
## 157 |   skeleton_day[, sex := "total"]
## 158 |   skeleton_day[, age := "total"]
## 159 |   sc::fill_in_missing_v8(skeleton_day, border = config$border)
## 160 |
## 161 |   # isoweek
## 162 |   skeleton_isoweek[, sex := "total"]
## 163 |   skeleton_isoweek[, age := "total"]
## 164 |   sc::fill_in_missing_v8(skeleton_isoweek, border = config$border)
## 165 |   skeleton_isoweek[, date := as.Date(date)]
## 166 |
## 167 |   skeleton <- rbindlist(
## 168 |     list(
## 169 |       skeleton_day,
## 170 |       skeleton_isoweek
## 171 |     ),
## 172 |     use.names = T
## 173 |   )
```

4.5.4.8 Store the data

Insert the final data table into the database specified in the task description

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_clean_data
##
## 175 |   # put data in db table
## 176 |   schema$anon_example_weather_data$drop_all_rows_and_then_insert_data(skeleto
```

Restart R, load all packages and try and run the task.

Run the entire task by running `tm_run_task("weather_clean_data")`. If you ran the tasks at the beginning of the script you might need to run `schema$anon_example_weather_data$drop_all_rows()` first.

4.5.4.9 Full example

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_clean_data
##
##   1 | # **** action **** ----
##   2 | #' weather_clean_data (action)
##   3 | #' @param data Data
##   4 | #' @param argset Argset
##   5 | #' @param schema DB Schema
##   6 | #' @export
##   7 | weather_clean_data_action <- function(data, argset, schema) {
##   8 |   # tm_run_task("weather_clean_data")
##   9 |
```

```
##  10 |   if (plnr::is_run_directly()) {
##  11 |     # sc::tm_get_plans_argsets_as_dt("weather_clean_data")
##  12 |
##  13 |     index_plan <- 1
##  14 |     index_analysis <- 1
##  15 |
##  16 |     data <- sc::tm_get_data("weather_clean_data", index_plan = index_plan)
##  17 |     argset <- sc::tm_get_argset("weather_clean_data", index_plan = index_plan, index_ana
##  18 |     schema <- sc::tm_get_schema("weather_clean_data")
##  19 |   }
##  20 |
##  21 |   # special case that runs before everything
##  22 |   if (argset$first_analysis == TRUE) {
##  23 |
##  24 |   }
##  25 |
##  26 |   # make sure there's no missing data via the creation of a skeleton
##  27 |   # https://folkehelseinstituttet.github.io/fhidata/articles/Skeletons.html
##  28 |
##  29 |   # Create a variable (possibly a list) to hold the data
##  30 |   d_agg <- list()
##  31 |   d_agg$day_municip <- copy(data$day_municip)
##  32 |
##  33 |   # Pull out important dates
##  34 |   date_min <- min(d_agg$day_municip$date, na.rm = T)
##  35 |   date_max <- max(d_agg$day_municip$date, na.rm = T)
##  36 |
##  37 |   # Create `multiskeleton`
##  38 |   # granularity_geo should have the following groups:
##  39 |   # - nodata (when no data is available, and there is no "finer" data available to aggre
##  40 |   # - all levels of granularity_geo where you have data available
##  41 |   # If you do not have data for a specific granularity_geo, but there is "finer" data av
##  42 |   # then you should not include this granularity_geo in the multiskeleton, because you w
##  43 |   # it later when you aggregate up your data (baregion)
##  44 |   multiskeleton_day <- fhidata::make_skeleton(
##  45 |     date_min = date_min,
##  46 |     date_max = date_max,
##  47 |     granularity_geo = list(
##  48 |       "nodata" = c(
##  49 |         "wardoslo",
##  50 |         "extrawardoslo",
##  51 |         "missingwardoslo",
##  52 |         "wardbergen",
##  53 |         "missingwardbergen",
##  54 |         "wardstavanger",
##  55 |         "missingwardstavanger",
```

```
##  56 |         "notmainlandmunicip",
##  57 |         "missingmunicip",
##  58 |         "notmainlandcounty",
##  59 |         "missingcounty"
##  60 |       ),
##  61 |       "municip" = c(
##  62 |         "municip"
##  63 |       )
##  64 |     )
##  65 |   )
##  66 |
##  67 |   # Merge in the information you have at different geographical granularities
##  68 |   # one level at a time
##  69 |   # municip
##  70 |   multiskeleton_day$municip[
##  71 |     d_agg$day_municip,
##  72 |     on = c("location_code", "date"),
##  73 |     c(
##  74 |       "temp_max",
##  75 |       "temp_min",
##  76 |       "precip"
##  77 |     ) := .(
##  78 |       temp_max,
##  79 |       temp_min,
##  80 |       precip
##  81 |     )
##  82 |   ]
##  83 |
##  84 |   multiskeleton_day$municip[]
##  85 |
##  86 |   # Aggregate up to higher geographical granularities (county)
##  87 |   multiskeleton_day$county <- multiskeleton_day$municip[
##  88 |     fhidata::norway_locations_hierarchy(
##  89 |       from = "municip",
##  90 |       to = "county"
##  91 |     ),
##  92 |     on = c(
##  93 |       "location_code==from_code"
##  94 |     )
##  95 |   ][,
##  96 |     .(
##  97 |       temp_max = mean(temp_max, na.rm = T),
##  98 |       temp_min = mean(temp_min, na.rm = T),
##  99 |       precip = mean(precip, na.rm = T),
## 100 |       granularity_geo = "county"
## 101 |     ),
```

```
## 102 |     by = .(
## 103 |       granularity_time,
## 104 |       date,
## 105 |       location_code = to_code
## 106 |     )
## 107 |   ]
## 108 |
## 109 |   multiskeleton_day$county[]
## 110 |
## 111 |   # Aggregate up to higher geographical granularities (nation)
## 112 |   multiskeleton_day$nation <- multiskeleton_day$municip[
## 113 |     ,
## 114 |     .(
## 115 |       temp_max = mean(temp_max, na.rm = T),
## 116 |       temp_min = mean(temp_min, na.rm = T),
## 117 |       precip = mean(precip, na.rm = T),
## 118 |       granularity_geo = "nation",
## 119 |       location_code = "norge"
## 120 |     ),
## 121 |     by = .(
## 122 |       granularity_time,
## 123 |       date
## 124 |     )
## 125 |   ]
## 126 |
## 127 |   multiskeleton_day$nation[]
## 128 |
## 129 |   # combine all the different granularity_geos
## 130 |   skeleton_day <- rbindlist(multiskeleton_day, fill = TRUE, use.names = TRUE)
## 131 |
## 132 |   skeleton_day[]
## 133 |
## 134 |   # 10. (If desirable) aggregate up to higher time granularities
## 135 |   # if necessary, it is now easy to aggregate up to weekly data from here
## 136 |   skeleton_isoweek <- copy(skeleton_day)
## 137 |   skeleton_isoweek[, isoyearweek := fhiplot::isoyearweek_c(date)]
## 138 |   skeleton_isoweek <- skeleton_isoweek[
## 139 |     ,
## 140 |     .(
## 141 |       temp_max = mean(temp_max, na.rm = T),
## 142 |       temp_min = mean(temp_min, na.rm = T),
## 143 |       precip = mean(precip, na.rm = T),
## 144 |       granularity_time = "isoweek"
## 145 |     ),
## 146 |     keyby = .(
## 147 |       isoyearweek,
```

```
## 148 |       granularity_geo,
## 149 |       location_code
## 150 |     )
## 151 |   ]
## 152 |
## 153 |   skeleton_isoweek[]
## 154 |
## 155 |   # we now need to format it and fill in missing structural variables
## 156 |   # day
## 157 |   skeleton_day[, sex := "total"]
## 158 |   skeleton_day[, age := "total"]
## 159 |   sc::fill_in_missing_v8(skeleton_day, border = config$border)
## 160 |
## 161 |   # isoweek
## 162 |   skeleton_isoweek[, sex := "total"]
## 163 |   skeleton_isoweek[, age := "total"]
## 164 |   sc::fill_in_missing_v8(skeleton_isoweek, border = config$border)
## 165 |   skeleton_isoweek[, date := as.Date(date)]
## 166 |
## 167 |   skeleton <- rbindlist(
## 168 |     list(
## 169 |       skeleton_day,
## 170 |       skeleton_isoweek
## 171 |     ),
## 172 |     use.names = T
## 173 |   )
## 174 |
## 175 |   # put data in db table
## 176 |   schema$anon_example_weather_data$drop_all_rows_and_then_insert_data(skeleto
## 177 |
## 178 |   # special case that runs after everything
## 179 |   if (argset$last_analysis == TRUE) {
## 180 |
## 181 |   }
## 182 | }
```

4.6 Developing weather_export_plots

The final task of this tutorial, weather_export_plots, takes the cleaned data and plots 11 graphs (one for each county) of min and max temperatures. This means we need 11 plans, one for each county. We use input data generated by data clean_weather_data. Hence, we do not need to create a new schema. We are going to pass a few universal argset through the task definition to define the location to store the figures.

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_clean_data.r#L84-L88
##
## 84 |   multiskeleton_day$municip[]
## 85 |
## 86 |   # Aggregate up to higher geographical granularities (county)
## 87 |   multiskeleton_day$county <- multiskeleton_day$municip[
## 88 |     fhidata::norway_locations_hierarchy(
```

The benefits of placing the output directories and filenames in the task declaration are:

- It makes your action_fn more generic, and can be reused by multiple tasks
- It is easier to get an overview of where the output is being sent
- "More decisions" in the task config and "fewer decisions" in the action_fn makes the system easier for everyone to understand, because decisions become more explicit

Everything inside the curly brackets get passed through the action function.

Each plan only need the data for that specific location_code. This can be implemented in the manditory filters in the data selector function.

`fs::dir_create(glue::glue(argset$output_dir))` can be used to create the output directory.

Try putting everything you have learned so fare together and create this task by yourself. If you get stuck you can always peak below. Good luck!

4.6.1 1. Schemas

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/03_db_schemas.r#L66-L112
##
##  66 |   ## > anon_example_weather_data ----
##  67 |   sc::add_schema_v8(
##  68 |     name_access = c("anon"),
##  69 |     name_grouping = "example_weather",
##  70 |     name_variant = "data",
##  71 |     db_configs = sc::config$db_configs,
##  72 |     field_types =  c(
##  73 |       "granularity_time" = "TEXT",
##  74 |       "granularity_geo" = "TEXT",
##  75 |       "country_iso3" = "TEXT",
##  76 |       "location_code" = "TEXT",
##  77 |       "border" = "INTEGER",
##  78 |       "age" = "TEXT",
##  79 |       "sex" = "TEXT",
##  80 |
##  81 |       "date" = "DATE",
##  82 |
```

```
##  83 |       "isoyear" = "INTEGER",
##  84 |       "isoweek" = "INTEGER",
##  85 |       "isoyearweek" = "TEXT",
##  86 |       "season" = "TEXT",
##  87 |       "seasonweek" = "DOUBLE",
##  88 |
##  89 |       "calyear" = "INTEGER",
##  90 |       "calmonth" = "INTEGER",
##  91 |       "calyearmonth" = "TEXT",
##  92 |
##  93 |       "temp_max" = "DOUBLE",
##  94 |       "temp_min" = "DOUBLE",
##  95 |       "precip" = "DOUBLE"
##  96 |     ),
##  97 |     keys = c(
##  98 |       "granularity_time",
##  99 |       "location_code",
## 100 |       "date",
## 101 |       "age",
## 102 |       "sex"
## 103 |     ),
## 104 |     censors = list(
## 105 |       anon = list(
## 106 |
## 107 |       )
## 108 |     ),
## 109 |     validator_field_types = sc::validator_field_types_sykdomspulsen,
## 110 |     validator_field_contents = sc::validator_field_contents_sykdomspulsen,
## 111 |     info = "This db table is used for..."
## 112 |   )
```

This schema has already been created by the previous task `weather_clean_data`.

4.6.2 2. Task definition (task_from_config)

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/04_tasks.r#L72-L100
##
##  72 |   ## > weather_clean_data ----
##  73 |   # tm_run_task("weather_export_plots")
##  74 |   sc::add_task_from_config_v8(
##  75 |     name_grouping = "weather",
##  76 |     name_action = "export_plots",
##  77 |     name_variant = NULL,
##  78 |     cores = 1,
##  79 |     plan_analysis_fn_name = NULL,
##  80 |     for_each_plan = plnr::expand_list(
```

```
##  81 |       location_code = fhidata::norway_locations_names()[granularity_geo %in% c("county")
##  82 |     ),
##  83 |     for_each_analysis = NULL,
##  84 |     universal_argset = list(
##  85 |       output_dir = tempdir(),
##  86 |       output_filename = "weather_{argset$location_code}.png",
##  87 |       output_absolute_path = fs::path("{argset$output_dir}", "{argset$output_filename}")
##  88 |     ),
##  89 |     upsert_at_end_of_each_plan = FALSE,
##  90 |     insert_at_end_of_each_plan = FALSE,
##  91 |     action_fn_name = "scexample::weather_export_plots_action",
##  92 |     data_selector_fn_name = "scexample::weather_export_plots_data_selector",
##  93 |     schema = list(
##  94 |       # input
##  95 |       "anon_example_weather_data" = sc::config$schemas$anon_example_weather_data
##  96 |
##  97 |       # output
##  98 |     ),
##  99 |     info = "This task ploduces plots"
## 100 |   )
```

4.6.2.1 Plan/analysis structure

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/04_tasks.r#L79-L83
##
## 79 |     plan_analysis_fn_name = NULL,
## 80 |     for_each_plan = plnr::expand_list(
## 81 |       location_code = fhidata::norway_locations_names()[granularity_geo %in% c("county")]
## 82 |     ),
## 83 |     for_each_analysis = NULL,
```

Here we choose a plan-heavy approach (11 plans, 1 analysis per plan) to minimize the amount of data loaded into RAM at any point in time.

4.6.2.2 Universal argset

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/04_tasks.r#L84-L88
##
## 84 |     universal_argset = list(
## 85 |       output_dir = tempdir(),
## 86 |       output_filename = "weather_{argset$location_code}.png",
## 87 |       output_absolute_path = fs::path("{argset$output_dir}", "{argset$output_filename}")
## 88 |     ),
```

4.6.3 3. data_selector_fn

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_export_plots.r#L45-L1
```

```
##
##  45 | # **** data_selector **** ----
##  46 | #' weather_export_plots (data selector)
##  47 | #' @param argset Argset
##  48 | #' @param schema DB Schema
##  49 | #' @export
##  50 | weather_export_plots_data_selector = function(argset, schema){
##  51 |   if(plnr::is_run_directly()){
##  52 |     # sc::tm_get_plans_argsets_as_dt("weather_export_plots")
##  53 |
##  54 |     index_plan <- 1
##  55 |
##  56 |     argset <- sc::tm_get_argset("weather_export_plots", index_plan = index_pla
##  57 |     schema <- sc::tm_get_schema("weather_export_plots")
##  58 |   }
##  59 |
##  60 |   # The database schemas can be accessed here
##  61 |   d <- schema$anon_example_weather_data$tbl() %>%
##  62 |     sc::mandatory_db_filter(
##  63 |       granularity_time = NULL,
##  64 |       granularity_time_not = NULL,
##  65 |       granularity_geo = NULL,
##  66 |       granularity_geo_not = NULL,
##  67 |       country_iso3 = NULL,
##  68 |       location_code = argset$location_code,
##  69 |       age = NULL,
##  70 |       age_not = NULL,
##  71 |       sex = NULL,
##  72 |       sex_not = NULL
##  73 |     ) %>%
##  74 |     dplyr::select(
##  75 |       # granularity_time,
##  76 |       # granularity_geo,
##  77 |       # country_iso3,
##  78 |       # location_code,
##  79 |       # border,
##  80 |       # age,
##  81 |       # sex,
##  82 |
##  83 |       date,
##  84 |
##  85 |       # isoyear,
##  86 |       # isoweek,
##  87 |       # isoyearweek,
##  88 |       # season,
##  89 |       # seasonweek,
```

```
##  90 |       #
##  91 |       # calyear,
##  92 |       # calmonth,
##  93 |       # calyearmonth,
##  94 |
##  95 |       temp_max,
##  96 |       temp_min
##  97 |     ) %>%
##  98 |     dplyr::collect() %>%
##  99 |     as.data.table() %>%
## 100 |     setorder(
## 101 |       # location_code,
## 102 |       date
## 103 |     )
## 104 |
## 105 |   # The variable returned must be a named list
## 106 |   retval <- list(
## 107 |     "data" = d
## 108 |   )
## 109 |   retval
## 110 | }
```

4.6.4 4. action_fn

```
## https://github.com/sykdomspulsen-org/sc-tutorial-end/blob/main/R/weather_export_plots.r#L1-L43
##
##  1 | # **** action **** ----
##  2 | #' weather_export_plots (action)
##  3 | #' @param data Data
##  4 | #' @param argset Argset
##  5 | #' @param schema DB Schema
##  6 | #' @export
##  7 | weather_export_plots_action <- function(data, argset, schema) {
##  8 |   # tm_run_task("weather_export_plots")
##  9 |
## 10 |   if(plnr::is_run_directly()){
## 11 |     # sc::tm_get_plans_argsets_as_dt("weather_export_plots")
## 12 |
## 13 |     index_plan <- 1
## 14 |     index_analysis <- 1
## 15 |
## 16 |     data <- sc::tm_get_data("weather_export_plots", index_plan = index_plan)
## 17 |     argset <- sc::tm_get_argset("weather_export_plots", index_plan = index_plan, index_an
## 18 |     schema <- sc::tm_get_schema("weather_export_plots")
## 19 |   }
## 20 |
```

```
## 21 |   # code goes here
## 22 |   # special case that runs before everything
## 23 |   if(argset$first_analysis == TRUE){
## 24 |
## 25 |   }
## 26 |
## 27 |   # create the output_dir (if it doesn't exist)
## 28 |   fs::dir_create(glue::glue(argset$output_dir))
## 29 |
## 30 |   q <- ggplot(data$data, aes(x = date, ymin = temp_min, ymax = temp_max))
## 31 |   q <- q + geom_ribbon(alpha = 0.5)
## 32 |
## 33 |   ggsave(
## 34 |     filename = glue::glue(argset$output_absolute_path),
## 35 |     plot = q
## 36 |   )
## 37 |
## 38 |   # special case that runs after everything
## 39 |   # copy to anon_web?
## 40 |   if(argset$last_analysis == TRUE){
## 41 |
## 42 |   }
## 43 | }
```

4.7 Final package

If you save, restart and load all all packages you can now see which schemas have been loaded by running `sc::tm_get_schema_names()`. You can now see that the schemas you made are included.

```
sc::tm_get_schema_names()
## [1] "config_last_updated"                                    "config_structu
## [3] "rundate"                                                "config_datetim
## [5] "anon_example_weather_rawdata"                           "anon_example_w
## [7] "anon_example_income"                                    "anon_example_h
## [9] "anon_example_house_prices_outliers_after_adjusting_for_income"
```

You can alsp see which tasks have been loaded by running `sc::tm_get_task_names()`. These tasks are included in the skeleton.

```
sc::tm_get_task_names()
## [1] "weather_download_and_import_rawdata"                     "weather_clean
## [3] "weather_export_plots"                                   "household_inc
## [5] "household_incomes_and_house_prices_fit_model_and_find_outliers" "household_inc
```

4.7.1 Running

You can now run these tasks in your console if you want. Note that we use `scskeleton::tm_run_task` instead of `sc::tm_run_task`. This is because we want to ensure that `scexample::.onLoad` has been called which authenticates you.

```
scskeleton::tm_run_task("weather_download_and_import_rawdata")
scskeleton::tm_run_task("weather_clean_data")
scskeleton::tm_run_task("weather_export_weather_plots")
```

Congratulations! You have now successfully finnished your first tutorial on Sykdomspulsen Core.

4.8 What now?

After Tutorial 1, we expect that you understand the four fundamental parts of developing a task:

1. Schemas
2. Task definition (task_from_config)
3. data_selector_fn
4. action_fn

We also expect that you can:

1. Run a task using `tm_run_task`
2. Use `sc::tm_get_plans_argsets_as_dt` to identify which `index_plan` and `index_analysis` corresponds to the plan/analysis you are interested in (e.g. Oslo)
3. Run the inside code of a `data_selector_fn` for different `index_plan`s as if it were an interactive script
4. Run the inside code of an `action_fn` for different `index_plan`s and `index_analysis`s as if it were an interactive script