User guide to UHERO's forecast processes

Peter Fuleky

2025-01-03

Table of contents

1 About

This document describes some useful practices for using R for applied research, especially in the time series and forecasting domain. It also serves as a guide for contributors to the `forecastr` R project. The focus of the project is forecasting using multi-equation behavioral models. The project encompasses data preparation, model selection (work in progress), external forecast generation, local forecast generation (planned), simulations (planned), and forecast distribution to a more granular scale.

1.1 Contents

Chapters 2 - 4 discuss the general setup of a collaborative project environment under version control. Chapter 5 deals with the setup file that configures the most general aspects of the `forecastr` project. Chapter 6 describes useful functions for time series manipulation and forecasting. Chapter 7 gives examples of best practices for time series manipulation. Chapter 8 describes a forecasting workflow from model selection to simulation.

2 Projects

Projects are useful for organizing work, especially when working on multiple pieces of research simultaneously. They help to keep the workspace clean and avoid conflicts between tasks. Projects also make it easier to share work with others. A project consists of the files associated with a given project (input data, R scripts, analytical results, and figures) kept together in a folder.

2.1 Project setup and conventions

Create a new project locally in RStudio under the File menu or using `usethis::create_project("proj_dir")`. The *.Rproj* file contains the project settings. Open the project by double clicking this file in Finder. The minimum structure of a project includes an *R* folder for scripts, a *data* folder for data, and an *output* folder for reports and plots. If present, the *data/raw* folder contains data external to the project and the *data/processed* folder contains intermediate processed data. Although local projects are sometimes useful to explore an idea, whenever you consider version tracking or collaboration, the project should be initiated from GitHub (see Chapter 3 for details).

Use the renv package to store information about the packages used in the project. The renv package facilitates sharing a project and maintaining the same behavior on different machines. It creates a local package directory for the project. This means that it keeps track of all the packages and package versions that are used in the project, and collaborators can restore the exact same package environment and reproduce the results (see Chapter 4 for details).

Use the here package to create paths relative to the project root. For example, `here::here("data", "raw/file.csv")` returns the path to the file *file.csv* in the *data/raw* folder. Load libraries and put hard coded lines at the top of the script. Use the conflicted package to detect conflicts across packages and assign preferences. For example, `conflict_prefer("filter", "dplyr")` assigns preference to the `filter` function in the dplyr package over the `filter` function in the stats package. Don't save the workspace on exit (Tools > Global Options > General > Save workspace to .RData on exit > Never or `usethis::use_blank_slate()`).

Start each pipe with a comment, and if necessary add comments to each line. Enable Github Copilot for RStudio; it is free for higher education users. Github Copilot will suggest code based on comments, which you can accept with the tab key. Use sectioning comments (#

comments followed by at least four dashes —-) to separate different parts of the script (they show up in the outline section of the editor pane). Use the addin provided by styler package to format the code. Follow the `tidyverse` "dialect" and syntax.

Use R scripts for coding; don't put the analysis into chunks in markdown documents. Only render important results in code chunks of quarto (*qmd*) or Rmarkdown (*Rmd*) documents. Within a *qmd* or *Rmd* document source the R script containing the analysis. Alternatively, save the entire workspace or individual objects from the R script, and then load these in the appropriate code chunks of the markdown document. Make sure the code chunks are looking for objects in the correct working directory.

Store secrets, passwords, and keys with the keyring package. For example, set the UDAMAN token with `keyring::key_set_with_value(service = "udaman_token", password = "-ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890=")` and retrieve it with `keyring::key_get("udaman_token")`. To avoid disclosing your secrets, do not store/assign the retrieved credentials to a variable. If security is not a concern, environment variables, such as API keys, can also be stored in a project specific *.Renviron* file; it must end with `\n`. In addition to *.Renviron*, the *.Rprofile* file is also executed each time R starts up. The latter typically contains a script with options and startup tasks. Both files are located in the project root folder.

2.2 Additional resources

Overview of R setup:
https://rstats.wtf

Best practices:
https://kdestasio.github.io/post/r_best_practices/

Considerations for structuring projects:
https://www.r-bloggers.com/2018/08/structuring-r-projects/

Set up your work in projects:
https://r4ds.hadley.nz/workflow-scripts#projects

Efficient data management in R:
https://www.r-bloggers.com/2020/02/efficient-data-management-in-r/

Efficient R programming:
https://csgillespie.github.io/efficientR/

Data science workflow:
http://dcl-workflow.stanford.edu

3 Version control

Version control tracks changes in files over time, allows you to revert files back to a previous state and compare changes over time. It is essential for the collaborative development of a project, but is also useful for individual projects. Git is a popular version control system and GitHub is a web-based platform that hosts Git repositories and provides tools for collaboration.

3.1 Version control terms in Git(Hub)

- **.gitignore**: A file that specifies which files and directories should be excluded from version control.
- **Branch**: A separate line of development or parallel version of the repository. Branches are used to develop features or fix bugs without affecting the main branch.
- **Clone**: A copy of a repository on your local machine. Cloning a repository allows you to work on the project locally.
- **Commit**: A snapshot of the project at a specific point in time. Commits are used to save changes to the repository.
- **Conflict**: When two branches have made changes to the same line in a file, a conflict occurs. Conflicts need to be resolved before the changes can be merged.
- **Fork**: A copy of a repository on GitHub. This copy is independent of the original and you can make changes to it without affecting the original project.
- **HEAD**: A reference to the most recent commit in the repository, tip of the branch that is currently checked out.
- **Main or Master**: The default branch in a Git repository.
- **Merge**: Combining changes from one branch into another branch.
- **Origin**: The default name of primary version of the repository on GitHub.
- **Pull**: Getting changes from GitHub to your local repository.

- **Pull request**: A request to merge proposed changes from one branch into another branch. Pull requests are used to review and discuss changes before merging them into the main branch.
- **Push**: Sending changes from your local repository to GitHub.
- **Remote**: A remote repository on GitHub that you can pull from and push to.
- **Repository**: A folder that contains all the files and history of a project. There can be a local repository (on your computer) and a remote repository (hosted on GitHub).
- **Stage**: Staging allows you to select which changes should be included in the next commit.
- **Upstream**: The original repository that you forked from on GitHub.

3.2 Version control setup

RStudio has built in support for Git-based version control for a project. Check if Git is installed by running `git --version` in the terminal. Git is included in Xcode command line tools, which can be installed by running `xcode-select --install` in the terminal. Provide your GitHub user name and email to Git via `usethis::use_git_config(user.name = "Jane Doe", user.email = "jane@example.org")`. Next, generate and store a personal access token for GitHub via `usethis::create_github_token()` and `gitcreds::gitcreds_set()`, respectively. Finally, make sure version control is enabled in RStudio (RStudio > Global Options > Git/SVN > Enable version control interface for RStudio projects > check box or `usethis::use_blank_slate()`). This concludes the setup for Git and GitHub, making it possible to establish a connection between RStudio and GitHub.

Always start setting up version control on GitHub. Even if you already have a project on your computer, begin by setting up a repository on GitHub. This can be done by clicking on the "+" in the top right corner of the GitHub website and selecting "New repository". Give the repository a name and description, make it public or private, add a readme file, and choose the R template for .gitignore. After creating the repository, copy the URL and open RStudio. The next paragraph describes a robust local setup method with the `usethis` package, but there is also a menu based option: create a new project from version control by selecting File > New Project > Version Control > Git and pasting the URL in the "Repository URL" field. Use the repository name as the project directory name. Choose a folder on your computer where the project will be stored locally and click "Create". This will clone the repository to your computer and create a new RStudio project. The project is now connected to the repository on GitHub.

The local setup described above can be automated with `usethis::create_from_github("repo_url", "proj_dir")`. If you have permission to push to the remote GitHub repository because you are an owner or collaborator on the project, then `create_from_github` will clone it. If you do not have parmission to push to the repository, `create_from_github` will fork and clone

it. In either case, do not work in the main branch. Instead, create a "dev" branch in RStudio and work in that branch: in the Git pane, click on the "New Branch", enter "dev" as the branch name, keep the remote origin, and chek the sync with remote box. This will create a new branch and switch to it.

While in the dev branch, make changes to scripts or other files and save them. When you are ready to commit the changes, stage the files in the Git pane ("Command a" selects all files), click "Commit" at the top of the Git pane. In the pop-up window, the "Diff" button allows you to browse the changes made to the files in the repository, while the "History" button shows the commit history of the repository. These can be analyzed more conveniently on GitHub as described below in Section 3.3. Enter a commit message in the text box, and click the "Commit" button. Then push the changes to the remote repository. Go to the GitHub page of the dev branch and create a pull request to merge the dev branch into the main branch of the remote repository. After the pull request is merged, delete the dev branch on GitHub. Delete the local dev branch by executing `git branch -d dev` in the terminal. Finally, switch back to the main branch in RStudio and pull the changes from the remote repository.

3.3 Time travel on GitHub

From your repo's landing page, access commit history by clicking on "123 Commits" under the green Code button. Once you're viewing the history, notice three ways to access more info for each commit:

- The clipboard icon copies the SHA of the commit. This can be handy if you need to refer to this commit elsewhere, e.g. in an issue thread or a commit message or in a Git command you're forming for local execution.
- Click on the abbreviated SHA itself in order to the view the diff associated with the commit.
- Click on the double angle brackets <> to browse the state of the entire repo at that point in history.

Back out of any drilled down view by clicking on YOU/REPO to return to your repo's landing page. This brings you back to the present state and top-level of your repo.

Once you've identified a relevant commit, diff, or file state, you can copy the current URL from your browser and use it to enhance online discussion elsewhere, i.e. to bring other people to this exact view of the repo. The hyperlink-iness of repos hosted on GitHub can make online discussion of a project much more precise and efficient.

What if you're interested in how a specific file came to be the way it is? First navigate to the file in the repo, then notice "Blame" on the left and "History" in the upper right.

- The "blame" view of a file reveals who last touched each line of the file, how long ago, and the associated commit message. Click on the commit message to visit that commit. Or click the "stacked rectangles" icon to move further back in time, but staying in blame view. This is handy when doing forensics on a specific and small set of lines.
- The "history" view for a file is very much like the overall commit history described above, except it only includes commits that affect the file of interest. This can be handy when your inquiry is rather diffuse and you're trying to digest the general story arc for a file.

When viewing a file on GitHub, you can click on a line number to highlight it. Use "click … shift-click" to select a range of lines. Notice your browser's URL shows something of this form: https://github.com/OWNER/REPO/blob/SHA/path/to/file.R#L27-L31 If the URL does not contain the SHA, type "y" to toggle into that form. These file- and SHA-specific URLs are a great way to point people at particular lines of code in online conversations. It's best practice to use the uglier links that contain the SHA, as they will stand the test of time.

Search is always available in the upper-righthand corner of GitHub. Once you enter some text in the search box, a dropdown provides the choice to search in the current repo (the default) or all of GitHub. GitHub searches the contents of files (described as "Code"), commit messages, and issues. Take advantage of the search hits across these different domains. Again, this is a powerful way to zoom in on specific lines of code, revisit an interesting time in project history, or re-discover a conversation thread.

3.4 Version control workflow

After the initial setup, the workflow should always follow the following sequence:

1. in the local/main branch, click on the "Pull" button in the Git pane to pull changes from the main branch of the remote repository, which is

   - upstream if the repository was forked,
   - origin if the repository was cloned,

   in the case of forked repo, after pulling from upstream, push to origin,
2. create a new dev branch and switch to it,
3. make changes in the dev branch,
4. commit changes in the dev branch,
5. push changes to the dev branch of the remote origin repository,
6. create a pull request on GitHub to merge the origin dev branch a) into upstream main if the repository was forked, b) into origin main if the repository was cloned,

7. merge the pull request (or wait for it to be merged by the owner),
8. after merging, delete the dev branch on the remote and locally,
9. repeat step 1. (and then repeat it again before you resume your work on the project).

This workflow is recommended to avoid conflicts with other collaborators.

3.5 Dealing with conflicts

If a push is rejected, pull the changes from the remote repository. If there are conflicts, resolve them by editing the files and committing the changes. Every merge conflict inserts three delimiters:

`<<<<<<` feature branch name, the start of the merge conflict
`======` the separator between the content of both branches
`>>>>>>` base branch name, the end of the merge conflict

Fix the merge conflict by directly editing the script at the indicated locations. Often you can fix it by simply deleting the content of one of the branches within the conflict. Potentially you need to keep a mix of both. Don't forget to also delete the three delimiters when you're ready.

3.6 Additional resources

Visual Git guide:
https://inbo.github.io/git-course/index.html

Exhaustive discussion of Git for R users:
https://happygitwithr.com

A research workflow based on Github:
https://www.carlboettiger.info/2012/05/06/research-workflow.html

For more advanced tasks, use GitHub Desktop:
https://desktop.github.com

4 Package management

The renv package helps create **r**eproducible **env**ironments for your R projects. Use renv to make R projects more isolated, portable and reproducible.

- **Isolated**: Installing a new or updated package for one project won't break other projects, and vice versa. That's because renv gives each project its own private library.
- **Portable**: Easily transport projects from one computer to another, even across different platforms. renv makes it easy to install the packages the project depends on.
- **Reproducible**: renv records the exact package versions the project depends on, and ensures those exact versions get installed by others who work on the project.

4.1 Getting started

To convert a project to use renv, call `renv::init()`. This adds three new files and directories to the project:

- The project library, *renv/library*, is a library that contains all packages currently used by the project[1]. This is the key magic that makes renv work: instead of having one library containing the packages used in every project, renv gives you a separate library for each project. This provides the benefits of isolation: different projects can use different versions of packages, and installing, updating, or removing packages in one project doesn't affect any other project.
- The lockfile, *renv.lock*, records enough metadata about every package that it can be re-installed on a new machine.
- renv uses *.Rprofile* to configure the R session to use the project library. This ensures that once renv is turned on for a project, it stays on, until it is deliberately turned off.

The next important pair of tools is `renv::snapshot()` and `renv::restore()`. `snapshot()` updates the lockfile with metadata about the currently-used packages in the project library. Sharing the lockfile allows other people or other computers to reproduce the current project environment by running `restore()`, which uses the metadata from the lockfile to install exactly

[1] If you'd like to skip dependency discovery, you can call `renv::init(bare = TRUE)` to initialize a project with an empty project library.

the same version of every package. This pair of functions provides the benefits of reproducibility and portability: you are now tracking exactly which package versions you have installed so you can recreate them on other machines.

4.2 Collaboration

One of the reasons to use renv is to make it easier to share code in such a way that everyone gets exactly the same package versions. As above, start by calling `renv::init()`. You'll then need to commit *renv.lock*, *.Rprofile*, *renv/settings.json* and *renv/activate.R* to version control, ensuring that others can recreate your project environment. If you're using git, this is particularly simple because renv will create a *.gitignore*, and you can just commit all suggested files.

Now when one of your collaborators opens this project, renv will automatically bootstrap itself, downloading and installing the appropriate version of renv. It will also ask them if they want to download and install all the packages it needs by running `renv::restore()`.

4.3 Installing packages

If you use renv for multiple projects, you'll have multiple libraries, meaning that you'll often need to install the same package in multiple places. It would be annoying if you had to download (or worse, compile) the package repeatedly, so renv uses a package cache. That means you only ever have to download and install a package once, and for each subsequent install, renv will just add a link from the project library to the global cache. You can learn more about the cache in `vignette("package-install")`.

After installing the package and checking that the code works, you should call `renv::snapshot()` to record the latest package versions in your lockfile. If you're collaborating with others, you'll need to commit those changes to git, and let them know that you've updated the lockfile and they should call `renv::restore()` when they're next working on a project.

4.4 Updating packages

Regularly (at least once a year) update the packages in your project to get the latest versions of all dependencies. Similarly, if you're making major changes to a project that you haven't worked on for a while, it's often a good idea to start with an `renv::update()`[2] before making any changes to the code.

[2] You can also use `update.packages()`, but `renv::update()` works with the same sources that `renv::install()` supports.

After calling `renv::update()`, you should run the code in your project and verify that it still works (or make any changes needed to get it working). Then call `renv::snapshot()` to record the new versions in the lockfile. If you get stuck, and can't get the project to work with the new versions, you can call `renv::restore()` to roll back changes to the project library and revert to the known good state recorded in your lockfile. If you need to roll back to an even older version, take a look at `renv::history()` and `renv::revert()`. `renv::update()` will also update renv itself, ensuring that you get all the latest features.

4.5 Workflow for setting up a project with version control and renv

The combination of version control and package management can facilitate reproducibility and collaboration.

1. Set up a repository on GitHub
2. Clone the repo with `usethis::create_from_github()`
3. Initialize renv with `renv::init()`
4. Edit a script, install packages, run the script
5. Store info about installed packages with `renv::snapshot()`
6. Sync the project state with GitHub by committing and pushing up the changes
7. Collaborators (fork and) clone the repo, and install the necessary packages with `renv::restore()`
8. In subsequent work,
    a. pull the current state of the repo from GitHub,
    b. run `renv::restore()` to install the necessary packages,
    c. edit the script, install new packages, run the script,
    d. run `renv::snapshot()`,
    e. push changes to GitHub and place a pull request if necessary.

This process ensures that you will always work with the most recent version of your project.

4.6 Additional resources

Overview of `renv`:
https://rstudio.github.io/renv/articles/renv.html

5 Setup of the forecastr project

The `setup.R` file contains general information used throughout the project. The contents are listed below.

5.1 Start with a clean slate

First remove all objects from global environment:
`rm(list = ls())`

If only some objects need to be removed, search for them via wildcards:
`rm(list = ls(pattern = glob2rx("*__*")))`

Detach all loaded packages:

```
if (!is.null(names(sessionInfo()$otherPkgs))) {
invisible(
  suppressMessages(
    suppressWarnings(
      lapply(
        paste("package:", names(sessionInfo()$otherPkgs), sep=""),
        detach,
        character.only = TRUE,
        unload = TRUE
        )
      )
    )
  )
}
```

5.2 Packages

The setup file clarifies its own location relative to the project root and loads the necessary packages.

Navigate within a project using the `here()` package. Start by specifying:
`here::i_am("R/setup.R")`

Then load necessary packages

```
# load necessary packages
library(here) # navigation within the project
library(conflicted) # detect conflicts across packages
library(tidyverse) # a set of frequently used data-wrangling tools
library(magrittr) # more than just pipes
library(lubridate) # dealing with dates
library(tsbox) # dealing with time series
library("pins")
# library(bimets)
# install.packages("devtools")
# devtools::install_github("UHERO/fcutils")
# renv::purge("fcutils")
library("fcutils")
```

Detect conflicts across packages and assign preferences

```
conflicted::conflict_scout()
conflicted::conflict_prefer("filter", "dplyr") # dplyr v stats
conflicted::conflict_prefer("first", "dplyr") # dplyr v xts
conflicted::conflict_prefer("lag", "dplyr") # dplyr v stats
conflicted::conflict_prefer("last", "dplyr") # dplyr v xts
conflicted::conflict_prefer("extract", "magrittr") # magrittr vs tidyr
conflicted::conflict_prefer("set_names", "magrittr") # magrittr vs purrr
```

Verify top level project directory `here::here()`.

5.3 Package descriptions

Only load essential packages with many useful functions (don't load a whole package to access a single function). Refer to individual functions in packages that are not loaded by `namespace::function()`.

- Core tidyverse packages.

- Non-core tidyverse packages (need to be loaded separately):
  - magrittr
  - lubridate
- Time series tools in tsbox (learn them and use them, very useful). All start with `ts_`.
- Load the fcutils package for utility functions.
- Forecasting with multi-equation behavioral models: only the load bimets pakage if actually doing forecasts, no need for data manipulation.
- bimets depends on the xts package (if not loaded, can access necessary functions via `xts::function()`). Prefer using tsbox and tidyverse functions whenever possible, but understand the components and behavior of xts objects.

6 Utility functions

This section describes utility functions that facilitate common operations on time series data. These functions perform frequency conversion, growth rate conversion, time series arithmetic and combination, plot generation, etc.

The R ecosystem knows a vast number of time series classes: ts, xts, zoo, tsibble, tibbletime, tis, or timeSeries. The plethora of standards causes confusion. As different packages rely on different classes, it is hard to use them in the same analysis. The tsbox package provides a set of tools that make it easy to switch between these classes. It also allows the user to treat time series as plain data frames, facilitating the use with tools that assume rectangular data. In each section below functions from the tsbox package are listed first followed by functions from the fcutils package. The uherotheme package contains functions to apply the UHERO style guide to plots made with ggplot. These functions are listed under Section 6.7.

6.1 Input-output and data generation

6.1.1 fcutils::get_series()

6.1.1.1 Description

Download a set of series from udaman using series names

6.1.1.2 Usage

```
get_series(
  ser_id_vec,
  format = "wide",
  expand,
  raw = TRUE,
  rename = "compact",
  freq = NULL,
  descr = FALSE,
```

```
  public = FALSE
)
```

6.1.1.3 Arguments

ser_id_vec	vector of series names (character)
format	"wide" (default) or "long" or "xts"
expand	DEPRECATED, USE raw INSTEAD "true" (default) or "raw" ("true" downloads formatted da
raw	TRUE (default) or FALSE (TRUE downloads raw data, FALSE downloads scaled and rounded
rename	"compact" (default), "full", "no". "compact": @ replaced by _ and no frequency; "full": @ repla
freq	if frequency is missing from series names (or want to modify freq in existing names) specify freq
descr	if TRUE add to the udaman series name the series description in parentheses (default: FALSE)
public	if TRUE use the public API interface - does not require VPN (default: FALSE)

6.1.1.4 Details

This function requires permission to access UDAMAN. Store the udaman token in the .Renvi-
ron file using the following format: udaman_token = "-ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890="
Or using fcutils::set_udaman_token("-ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890=")
Or store the udaman token among your credentials (e.g. keychain) using keyring:
keyring::key_set_with_value(service = "udaman_token", password = "-ABCDEFGHIJKLMNOPQRSTUVWX

6.1.1.5 Value

time and data for all series combined in an object specified by the format option

6.1.1.6 Examples

```
get_series(c("VISNS@HI.M", "VAPNS@HI.M"), raw = TRUE)
get_series(c("VEXP_RB@HI.M"))
get_series(c("VISNS@HI.M", "VAPNS@HI.M"), public = TRUE)
get_series(c("VISNS@HI.M", "VISUSNS@HI.M"), freq = "Q")
get_series(c("VISNS@HI.M", "VAPNS@HI.M"), format = "xts")
get_series(c("VISNS@HI.M"), format = "xts")
get_series(c("VISNS@HI.M"), format = "xts", descr = TRUE)
get_series(c("E_NF_HI", "ECT_HI", "E_TU_HAW"), freq = "M")
get_series(c("E_NF__HI_M", "ECT__HI_M", "VAP__HI_W"))
get_series(c("E_NF_AT_HI_M", "ECT_AT_HI_M", "VAP_AT_HI_W"))
```

6.1.2 fcutils::get_series_exp()

6.1.2.1 Description

Download series listed in an export table from udaman

6.1.2.2 Usage

```
get_series_exp(
  exp_id,
  format = "wide",
  expand,
  raw = TRUE,
  rename = "compact",
  descr = FALSE,
  public = FALSE,
  save_loc = NULL
)
```

6.1.2.3 Arguments

exp_id	export id (character or numeric)
format	”wide” (default) or ”long” or ”xts”
expand	DEPRECATED, USE raw INSTEAD ”true” or ”raw” (”true” downloads formatted data, ”raw” do
raw	TRUE (default) or FALSE (TRUE downloads raw data, FALSE downloads scaled and rounded da
rename	”compact” (default), ”full”, ”no”. ”compact”: @ replaced by _ and no frequency; ”full”: @ replace
descr	if TRUE add to the udaman series name the series description in parentheses (default: FALSE)
public	if TRUE use the public API interface - does not require VPN (default: FALSE)
save_loc	file path for saving data incl. extension (”html” or ”csv”) (default NULL)

6.1.2.4 Details

This function requires permission to access UDAMAN. Store the udaman token in the .Renvi-
ron file using the following format: udaman_token = ”-ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890=”
Or using fcutils::set_udaman_token(”-ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890=”)
Or store the udaman token among your credentials (e.g. keychain) using keyring:
keyring::key_set_with_value(service = ”udaman_token”, password = ”-ABCDEFGHIJKLMNOPQRSTUVWX

6.1.2.5 Value

time and data for all series combined in a tibble

6.1.2.6 Examples

```
get_series_exp(exp_id = 74)
get_series_exp(74, format = "xts")
```

6.1.3 fcutils::set_udaman_token()

6.1.3.1 Description

Set udaman token for API access

6.1.3.2 Usage

```
set_udaman_token(key)
```

6.1.3.3 Arguments

key	a string containing 44 characters

6.1.3.4 Details

Save the token in .Renviron as udaman_token = key.

6.1.3.5 Value

true if setting the token in .Renviron succeeded

6.1.3.6 Examples

```
set_udaman_token("-ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890=")
```

6.1.4 fcutils::make_xts()

6.1.4.1 Description

Create xts and fill with values

6.1.4.2 Usage

```
make_xts(start = bnk_start, end = NULL, per = "year", val = NA_real_)
```

6.1.4.3 Arguments

`start`	date of series start (character: ”yyyy-mm-dd”, ”yyyyqq”, ”yyyy”)
`end`	date of series end (character: ”yyyy-mm-dd”, ”yyyyqq”, ”yyyy”)
`per`	periodicity of series (character: ”year” - default) if date format of start is quarterly, automatically set
`val`	values to fill in (numeric scalar, vector, or tibble)

6.1.4.4 Details

when end is missing, but val is a vector of more than one element, the end date is automatically determined by the length of the val vector. if end is missing and val is a scalar, the end date is set to bnk_end. if end is missing the remaining arguments have to be named. if val is a tibble, the end date is automatically determined by the number of rows in the tibble.

6.1.4.5 Value

an xts series

6.1.4.6 Examples

```
make_xts()
make_xts(val = 0, per = "m")
make_xts(start = 20100101, per = "quarter", val = 0)
make_xts(start = 2010.1, per = "q", val = 1:10)
make_xts(2010.1, val = 1:10) ## automatically set per = "quarter"
make_xts(start = "2010-01-01", per = "m", val = 0)
make_xts(start = 201001, per = "q",
         val = tibble::tibble(E_NF_HON = c(1:10), ECT_HI = c(11:20)))
```

6.1.5 fcutils::addf()

6.1.5.1 Description

Create xts addfactor

6.1.5.2 Usage

```
addf(
  start = bnk_start,
  end = bnk_end,
  from = 0,
  to = 0,
  ser_name = "value",
  per = "year"
)
```

6.1.5.3 Arguments

start	start date of linear interpolation (character: ”yyyy-mm-dd”, ”yyyyqq”, ”yyyy”)
end	end date of linear interpolation (character: ”yyyy-mm-dd”, ”yyyyqq”, ”yyyy”)
from	first value for linear interpolation (numeric)
to	last value for linear interpolation (numeric)
ser_name	name of the xts series (string)
per	periodicity of series (character: ”year” - default) if date format of start is quarterly, automatically

6.1.5.4 Details

this is a wrapper around make_xts with some additional functionality. the start and end dates specify the span of the non-zero add-factor value. the remaining period between start and end is filled with zeros.

6.1.5.5 Value

a single xts series spanning bnk_start-bnk_end

6.1.5.6 Examples

```
addf()
addf(201002, 201504, 1, 2)
addf(20100101, 20601201, 1, 2, per = "month")
addf(20100101, from = 1, to = 2, per = "quarter")
addf(2010.2, 2015.4, 1, 2, "ECT_HI")
```

6.1.6 fcutils::copy_tbl()

6.1.6.1 Description

Copy a data frame to clipboard (only works on MacOS)

6.1.6.2 Usage

```
copy_tbl(x, dec = 2)
```

6.1.6.3 Arguments

x	tibble (or data frame) to be copied
dec	number of decimals to round numeric columns to (default: 2)

6.1.6.4 Value

copy_tbl() returns the input x invisibly

6.1.6.5 Examples

```
monthly_data_example |> copy_tbl()
monthly_data_example |> copy_tbl(1)
```

6.1.7 fcutils::gen_table()

6.1.7.1 Description

Generate a table with time series

6.1.7.2 Usage

```
gen_table(
  x,
  tbl_start = as.character(Sys.Date() - lubridate::years(10)),
  tbl_end = as.character(Sys.Date() + lubridate::years(2)),
  percent = "pc",
  time_across = TRUE,
  tbl_height = 800,
  save_loc = NULL
)
```

6.1.7.3 Arguments

`x`	a ts-boxable object
`tbl_start`	start period for table
`tbl_end`	end period for table
`percent`	what type of percent should be added ("none", "pc" (default), "pcy", "pca")
`time_across`	should time be in column headers and variable names in first column (default TRUE)
`tbl_height`	the height of the table in px (default 800)
`save_loc`	file path for saving table incl. extension ("html" or "csv") (default NULL)

6.1.7.4 Value

table formatted for output

6.1.7.5 Examples

```
quarterly_data_example %>%
  tsbox::ts_long() %>%
  tsbox::ts_tslist() %>%
  gen_table()
gen_table(quarterly_data_example)
gen_table(quarterly_data_example, percent = "none")
gen_table(quarterly_data_example, percent = "pcy", time_across = FALSE)

gen_table(quarterly_data_example,
  percent = "pcy",
  time_across = FALSE, save_loc = "~/Downloads/temp.csv"
)
gen_table(quarterly_data_example,
  percent = "pcy", time_across = TRUE,
  save_loc = "~/Downloads/temp.html"
)
```

6.1.8 fcutils::write_tsd()

6.1.8.1 Description

Save a ts-boxable object in tsd format

6.1.8.2 Usage

```
write_tsd(x, file)
```

6.1.8.3 Arguments

`x`	a ts-boxable object (only M, Q, A frequency)
`file`	character string denoting the location and name of the output file

6.1.8.4 Value

nothing (silently save the contents of the tsd file to a user defined location)

6.1.8.5 Examples

```
quarterly_data_example |> write_tsd("out.tsd")
```

6.2 Time series info and date manipulation

6.2.1 tsbox::ts_summary()

6.2.1.1 Description

Extract time series properties, such as the number of observations (`obs`), the time differences between observations (`obs`), the number of observations per year (`freq`), and the start time stamp (`start`) and the end time stamp (`end`) of the series.

6.2.1.2 Usage

```
ts_summary(x, spark = FALSE)
```

6.2.1.3 Arguments

`x`	ts-boxable time series, an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`
`spark`	logical should an additional column with a spark-line added to the data frame (experimental, ASCII o

6.2.1.4 Value

`ts_summary` returns a `data.frame`. Individual column can be accessed through the `$` notation (see examples).

6.2.1.5 Examples

```r
ts_summary(ts_c(mdeaths, austres))
ts_summary(ts_c(mdeaths, austres), spark = TRUE)
## Extracting specific properties
ts_summary(AirPassengers)$start
ts_summary(AirPassengers)$freq
ts_summary(AirPassengers)$obs
```

6.2.2 tsbox::ts_first_of_period()

6.2.2.1 Description

Replace date or time values by the first of the period. tsbox usually relies on timestamps being the first value of a period.

6.2.2.2 Usage

```r
ts_first_of_period(x)
```

6.2.2.3 Arguments

x	ts-boxable time series, an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `tbl`

6.2.2.4 Value

a ts-boxable object of the same class as `x`, i.e., an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `tbl_time`, `tis`, `irts` or `timeSeries`.

6.2.2.5 Examples

```r
x <- ts_c(
  a = ts_lag(ts_df(mdeaths), "14 days"),
  b = ts_lag(ts_df(mdeaths), "-2 days")
)
ts_first_of_period(x)
ts_first_of_period(ts_lag(ts_df(austres), "14 days"))
```

6.2.3 tsbox::ts_regular()

6.2.3.1 Description

Enforces regularity in data frame and `xts` objects, by turning implicit `NA`s into explicit `NA`s. In `ts` objects, regularity is automatically enforced.

6.2.3.2 Usage

```
ts_regular(x, fill = NA)
```

6.2.3.3 Arguments

`x`	ts-boxable time series, an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`,
`fill`	numeric, instead of `NA`, an alternative value can be specified. E.g., 0, -99.

6.2.3.4 Value

a ts-boxable object of the same class as `x`, i.e., an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `tbl_time`, `tis`, `irts` or `timeSeries`.

6.2.3.5 Examples

```
x0 <- AirPassengers
x0[c(10, 15)] <- NA
x <- ts_na_omit(ts_dts(x0))
ts_regular(x)
ts_regular(x, fill = 0)

m <- mdeaths
m[c(10, 69)] <- NA
f <- fdeaths
f[c(1, 3, 15)] <- NA

ts_regular(ts_na_omit(ts_dts(ts_c(f, m))))
```

6.2.4 tsbox::ts_na_omit()

6.2.4.1 Description

Remove NA values in ts-boxable objects, turning explicit into implicit missing values.

6.2.4.2 Usage

```
ts_na_omit(x)
```

6.2.4.3 Arguments

x	ts-boxable time series, an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `tbl`

6.2.4.4 Details

Note that internal NAs in `ts` time series will not be removed, as this conflicts with the regular structure.

6.2.4.5 Value

a ts-boxable object of the same class as `x`, i.e., an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `tbl_time`, `tis`, `irts` or `timeSeries`.

6.2.4.6 See Also

ts_regular, for the opposite, turning implicit into explicit missing values.

6.2.4.7 Examples

```r
x <- AirPassengers
x[c(2, 4)] <- NA

## A ts object does only know explicit NAs
ts_na_omit(x)

## by default, NAs are implicit in data frames
ts_df(x)

## make NAs explicit
ts_regular(ts_df(x))

## and implicit again
ts_na_omit(ts_regular(ts_df(x)))
```

6.2.5 tsbox::ts_span()

6.2.5.1 Description

Filter time series for a time span.

6.2.5.2 Usage

```r
ts_span(x, start = NULL, end = NULL, template = NULL, extend = FALSE)
```

6.2.5.3 Arguments

`x`	ts-boxable time series, an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl`
`start`	start date, character string of length 1, `Date` or `POSIXct`
`end`	end date, character string of length 1, `Date` or `POSIXct`.
`template`	ts-boxable time series, an object of class `ts`, `xts`, `data.frame`, `data.table`, or `tibble`. If provided
`extend`	logical. If true, the start and end values are allowed to extend the series (by adding `NA` values).

6.2.5.4 Details

All date and times, when entered as character strings, are processed by `anytime::anydate()` or `anytime::anytime()`. Thus a wide range of inputs are possible. See examples.

`start` and `end` can be specified relative to each other, using one of `"sec"`, `"min"`, `"hour"`, `"day"`, `"week"`, `"month"`, `"quarter"` or "year", or an abbreviation. If the series are of the same frequency, the shift can be specified in periods. See examples.

6.2.5.5 Value

a ts-boxable object of the same class as `x`, i.e., an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `tbl_time`, `tis`, `irts` or `timeSeries`.

6.2.5.6 Examples

```
## use 'anytime' shortcuts
ts_span(mdeaths, start = "1979")       ## shortcut for 1979-01-01
ts_span(mdeaths, start = "1979-4")     ## shortcut for 1979-04-01
ts_span(mdeaths, start = "197904")     ## shortcut for 1979-04-01

## it's fine to use an to date outside of series span
ts_span(mdeaths, end = "2001-01-01")

## use strings to set start or end relative to each other

ts_span(mdeaths, start = "-7 month")   ## last 7 months
ts_span(mdeaths, start = -7)           ## last 7 periods
ts_span(mdeaths, start = -1)           ## last single value
ts_span(mdeaths, end = "1e4 hours")    ## first 10000 hours


ts_plot(
  ts_span(mdeaths, start = "-3 years"),
  title = "Three years ago",
  subtitle = "The last three years of available data"
)

ts_ggplot(
  ts_span(mdeaths, end = "28 weeks"),
  title = "28 weeks later",
```

```
  subtitle = "The first 28 weeks of available data"
) + theme_tsbox() + scale_color_tsbox()


## Limit span of 'discoveries' to the same span as 'AirPassengers'
ts_span(discoveries, template = AirPassengers)
ts_span(mdeaths, end = "19801201", extend = TRUE)
```

6.2.6 fcutils::span()

6.2.6.1 Description

Specify span of time series (wrapper around tsbox::ts_span())

6.2.6.2 Usage

```
span(x, start = NULL, end = NULL, template = NULL, extend = FALSE)
```

6.2.6.3 Arguments

`x`	ts-boxable object to filter by span
`start`	start date (see examples)
`end`	end date (see examples)
`template`	ts-boxable time series (see tsbox::ts_span)
`extend`	logical. If true, the start and end values are allowed to extend the series (by adding NA values).

6.2.6.4 Value

filtered object of the same type as the input

6.2.6.5 Examples

```
quarterly_data_example |>
  span(2010.1)
quarterly_data_example |>
  span(2010.1, 2010.4)
quarterly_data_example |>
  span("2010-01-01", "2010-12-31")
```

6.2.7 fcutils::find_start()

6.2.7.1 Description

Find the date of the first observation (NAs are dropped)

6.2.7.2 Usage

```
find_start(x)
```

6.2.7.3 Arguments

x	ts-boxable object

6.2.7.4 Value

dates associated with first observation

6.2.7.5 Examples

```
quarterly_data_example |>
  dplyr::mutate(E_NF_HI = dplyr::if_else(time < "2000-01-01", NA_real_, E_NF_HI)) |>
  find_start()
```

6.2.8 fcutils::find_end()

6.2.8.1 Description

Find the date of the last observation (NAs are dropped)

6.2.8.2 Usage

```
find_end(x, last_day = FALSE)
```

6.2.8.3 Arguments

x	ts-boxable object
last_day	should the last day of period be returned (default: FALSE)

6.2.8.4 Value

date associated with last observation

6.2.8.5 Examples

```
quarterly_data_example |>
  dplyr::mutate(E_NF_HI = dplyr::if_else(time > "2022-01-01", NA_real_, E_NF_HI)) |>
  find_end()
quarterly_data_example |>
  dplyr::mutate(E_NF_HI = dplyr::if_else(time > "2022-01-01", NA_real_, E_NF_HI)) |>
  find_end(TRUE)
```

6.2.9 fcutils::nmons()

6.2.9.1 Description

Calculate number of months between two dates yyyyMm, yyyy.m or yyyy-mm-dd

6.2.9.2 Usage

```
nmons(dat1 = "", dat2 = "")
```

6.2.9.3 Arguments

`dat1`	date of period start (string: yyyyMm, yyyy.m, or yyyy-mm-dd)
`dat2`	date of period end (string: yyyyMm, yyyy.m, or yyyy-mm-dd)

6.2.9.4 Details

The endpoints are included in the result so subtract one for time difference. Also, the result is rounded down so partial months are not counted. See examples.

6.2.9.5 Value

numeric length of date range in months

6.2.9.6 Examples

```
nmons("2010M1", "2010M2")
nmons(2010.1, 2010.4)
nmons("2010-01-15", "2010-04-15")
nmons("2010-01-15", "2010-04-18")
nmons("2010-01-15", "2010-04-12")
```

6.2.10 fcutils::nqtrs()

6.2.10.1 Description

Calculate number of quarters between two dates yyyyQq, yyyy.q or yyyy-mm-dd

6.2.10.2 Usage

```
nqtrs(dat1 = "", dat2 = "")
```

6.2.10.3 Arguments

dat1	date of period start (string: yyyyQq, yyyy.q, or yyyy-mm-dd)
dat2	date of period end (string: yyyyQq, yyyy.q, or yyyy-mm-dd)

6.2.10.4 Details

The endpoints are included in the result so subtract one for time difference. Also, the result is rounded down so partial quarters are not counted. See examples.

6.2.10.5 Value

numeric length of date range in quarters

6.2.10.6 Examples

```
nqtrs("2010Q1", "2020Q4")
nqtrs(2010.1, 2020.4)
nqtrs("2010-01-01", "2020-10-01")
nqtrs("2010-02-01", "2020-11-01")
nqtrs("2010-02-01", "2020-10-01")
nqtrs("2010-01-01", "2020-11-01")
```

6.2.11 fcutils::qtrs()

6.2.11.1 Description

Convert period in quarters to period in months

6.2.11.2 Usage

```
qtrs(nr_quarters)
```

6.2.11.3 Arguments

nr_quarters	number of quarters in period (integer)

6.2.11.4 Value

number of months in period

6.2.11.5 Examples

```
qtrs(3)
lubridate::ymd("2020-01-01") + qtrs(3)
```

6.2.12 fcutils::p()

6.2.12.1 Description

Concatenate dates to obtain period

6.2.12.2 Usage

```
p(dat1 = "", dat2 = "")
```

6.2.12.3 Arguments

dat1	date of period start (string: see examples)
dat2	date of period end (string: see examples)

6.2.12.4 Value

string containing date range

6.2.12.5 Examples

```
p("2010-01-01", "2020-01-01")
p(20100101, 20200101)
p(2010.1, 2020.4)
p(,2020.4)
p("2010Q1", "2020Q4")
p(2010, 2020) ## for annual period only
```

6.2.13 fcutils::pm()

6.2.13.1 Description

Concatenate dates formatted as yyyyMm or yyyy.m to obtain period

6.2.13.2 Usage

```
pm(dat1 = "", dat2 = "")
```

6.2.13.3 Arguments

`dat1`	date of period start (string: yyyyMm or yyyy.m)
`dat2`	date of period end (string: yyyyMm or yyyy.m)

6.2.13.4 Value

string containing date range

6.2.13.5 Examples

```
pm("2010M1", "2020M4")
pm(2010.1, 2020.4)
pm(2010.1, )
pm(, 2010.1)
```

6.2.14 fcutils::pq()

6.2.14.1 Description

Concatenate dates formatted as yyyyQq or yyyy.q to obtain period

6.2.14.2 Usage

```
pq(dat1 = "", dat2 = "")
```

6.2.14.3 Arguments

dat1	date of period start (string: yyyyQq or yyyy.q)
dat2	date of period end (string: yyyyQq or yyyy.q)

6.2.14.4 Value

string containing date range

6.2.14.5 Examples

```
pq("2010Q1", "2020Q4")
pq(2010.1, 2020.4)
pq(2010.1, )
pq(, 2010.1)
```

6.2.15 fcutils::py()

6.2.15.1 Description

Concatenate dates formatted as yyyy to obtain period

6.2.15.2 Usage

```
py(dat1 = "", dat2 = "")
```

6.2.15.3 Arguments

dat1	year of period start (string or numeric: yyyy)
dat2	year of period end (string or numeric: yyyy)

6.2.15.4 Value

string containing date range

6.2.15.5 Examples

```
py("2010", "2020")
py(2010, 2020)
py(2010, )
py(, 2010)
```

6.2.16 fcutils::to_ymd()

6.2.16.1 Description

Parse strings into dates in yyyy-mm-dd format

6.2.16.2 Usage

```
to_ymd(x)
```

6.2.16.3 Arguments

x	string (string: yyyymmdd, yyyyqq, yyyy.q, yyyy)

6.2.16.4 Value

formatted dates (yyyy-mm-dd)

6.2.16.5 Examples

```
to_ymd(c("2010.0211", 202002, 2020.2, "2020"))
```

6.2.17 fcutils::ymd_to_yQq()

6.2.17.1 Description

Convert dates from yyyy-mm-dd to yyyyQqq format

6.2.17.2 Usage

```
ymd_to_yQq(x)
```

6.2.17.3 Arguments

x	dates (string: yyyy-mm-dd)

6.2.17.4 Value

formatted dates (string: yyyyQqq)

6.2.17.5 Examples

```
ymd_to_yQq(c("2010-01-01", "2020-10-01"))
ymd_to_yQq(c("2010-01-01", "2020-10-01")) |> lubridate::yq()
```

6.3 Frequency conversion

6.3.1 tsbox::ts_frequency()

6.3.1.1 Description

Changes the frequency of a time series. By default, incomplete periods of regular series are omitted.

6.3.1.2 Usage

```
ts_frequency(
  x,
  to = c("year", "quarter", "month", "week", "day", "hour", "min", "sec"),
  aggregate = "mean",
  na.rm = FALSE
)
```

6.3.1.3 Arguments

x	ts-boxable time series, an object of class ts, xts, zoo, zooreg, data.frame, data.table, tbl, tb
to	desired frequency, either a character string ("year", "quarter", "month") or an integer (1, 4, 12
aggregate	character string, or function. Either "mean", "sum", "first", or "last", or any aggregate functi
na.rm	logical, if TRUE, incomplete periods are aggregated as well. For irregular series, incomplete period

6.3.1.4 Details

The tempdisagg package can convert low frequency to high frequency data and has support for ts-boxable objects. See `vignette("hf-disagg", package = "tempdisagg")`.

6.3.1.5 Value

a ts-boxable time series, with the same class as the input.

6.3.1.6 Examples

```
ts_frequency(cbind(mdeaths, fdeaths), "year", "sum")
ts_frequency(cbind(mdeaths, fdeaths), "quarter", "last")

ts_frequency(AirPassengers, 4, "sum")

## Note that incomplete years are omited by default
ts_frequency(EuStockMarkets, "year")
ts_frequency(EuStockMarkets, "year", na.rm = TRUE)
```

6.3.2 fcutils::disagg()

6.3.2.1 Description

Interpolate univariate or multivariate time series from low to high frequency

6.3.2.2 Usage

```
disagg(x, conv_type = "mean", target_freq = "quarter", pattern = NULL)
```

6.3.2.3 Arguments

`x`	a tx-boxable object at a low frequency (e.g. annual or quarterly)
`conv_type`	match the quarterly value via ”first”, ”last”, ”sum”, ”mean”
`target_freq`	target frequency ”quarter” or ”month”
`pattern`	a single high-frequency pattern that the interpolation should follow

6.3.2.4 Details

the time-span of the high-frequency pattern has to match or be larger than the time-span of the low frequency series. NA values are not allowed.

6.3.2.5 Value

interpolated object of the same type as the input

6.3.2.6 Examples

```
quarterly_data_example |>
  disagg(conv_type = "mean", target_freq = "month")
quarterly_data_example |>
  disagg(conv_type = "mean", target_freq = "month") |>
  tsbox::ts_long() |>
  tsbox::ts_frequency(to = "quarter", aggregate = "mean") |>
  tsbox::ts_wide() ## this matches original data
## works with a single series too
quarterly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_pick("E_NF_HI") |>
  disagg(conv_type = "mean", target_freq = "month") |>
  tsbox::ts_plot()
## using a high-frequency pattern
quarterly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_span("2005-01-01", "2020-01-01") |>
  disagg(
    conv_type = "mean", target_freq = "month", pattern = monthly_data_example |>
      tsbox::ts_long() |>
      tsbox::ts_pick("VISNS_HI")
  )
## multiple low-frequency series, same number of high-frequency patterns
purrr::map2(
  quarterly_data_example |>
    tsbox::ts_long() |>
    tsbox::ts_pick("E_NF_HI", "ECT_HI") |>
    tsbox::ts_span("2005-01-01", "2020-01-01") |>
    tsbox::ts_tslist(),
  monthly_data_example |>
    tsbox::ts_long() |>
    tsbox::ts_tslist(),
  ~ disagg(.x, conv_type = "mean", target_freq = "month", pattern = .y)
)
```

6.3.3 fcutils::QtoA()

6.3.3.1 Description

Aggregate from quarterly to annual frequency (superseded by tsbox::ts_frequency())

6.3.3.2 Usage

```
QtoA(ser_in, aggr = "mean")
```

6.3.3.3 Arguments

ser_in	the xts series to be converted (freq = q)
aggr	aggregate via mean (default) or sum

6.3.3.4 Value

converted xts series (freq = a)

6.3.3.5 Examples

```
quarterly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_xts() |>
  tsbox::ts_pick("E_NF_HI") |>
  QtoA() |> ## this matches with below
  AtoQ() |>
  QtoA() |> ## this matches with above
  tsbox::ts_plot()
```

6.3.4 fcutils::AtoQ()

6.3.4.1 Description

Linear interpolation based on AREMOS command reference page 292 (superseded by disagg())

6.3.4.2 Usage

```
AtoQ(ser_in, aggr = "mean")
```

6.3.4.3 Arguments

ser_in	the xts series to be interpolated (freq = a)
aggr	interpolation method: aggregate via mean (default) or sum

6.3.4.4 Value

interpolated xts series (freq = q)

6.3.4.5 Examples

```
quarterly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_xts() |>
  tsbox::ts_pick("E_NF_HI") |>
  QtoA() |> ## this matches with below
  AtoQ() |>
  QtoA() |> ## this matches with above
  tsbox::ts_plot()
```

6.4 Growth rates and level operations

6.4.1 tsbox::ts_pc()

6.4.1.1 Description

`ts_pcy` and `ts_diffy` calculate the percentage change rate and the difference compared to the previous period, `ts_pcy` and `ts_diffy` calculate the percentage change rate compared to the same period of the previous year. `ts_pca` calculates annualized percentage change rates compared to the previous period.

6.4.1.2 Usage

```
ts_pc(x)

ts_diff(x)

ts_pca(x)

ts_pcy(x)

ts_diffy(x)
```

6.4.1.3 Arguments

x	ts-boxable time series, an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `tbl`

6.4.1.4 Value

a ts-boxable object of the same class as `x`, i.e., an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `tbl_time`, `tis`, `irts` or `timeSeries`.

6.4.1.5 Examples

```
x <- ts_c(fdeaths, mdeaths)
ts_diff(x)
ts_pc(x)
ts_pca(x)
ts_pcy(x)
ts_diffy(x)
```

6.4.2 tsbox::ts_lag()

6.4.2.1 Description

Shift time stamps in ts-boxable time series, either by a number of periods or by a fixed amount of time.

6.4.2.2 Usage

```
ts_lag(x, by = 1)
```

6.4.2.3 Arguments

x	ts-boxable time series, an object of class ts, xts, zoo, zooreg, data.frame, data.table, tbl, tbl_ts, tb
by	integer or character, either the number of shifting periods (integer), or an absolute amount of time (chara

6.4.2.4 Details

The lag order, by, is defined the opposite way as in R base. Thus, -1 is a lead and +1 a lag.

If by is integer, the time stamp is shifted by the number of periods. This requires the series to be regular.

If by is character, the time stamp is shifted by a specific amount of time. This can be one of one of "sec", "min", "hour", "day", "week", "month", "quarter" or "year", optionally preceded by a (positive or negative) integer and a space, or followed by plural "s". This is passed to base::seq.Date(). This does not require the series to be regular.

6.4.2.5 Value

a ts-boxable object of the same class as x, i.e., an object of class ts, xts, zoo, zooreg, data.frame, data.table, tbl, tbl_ts, tbl_time, tis, irts or timeSeries.

6.4.2.6 Examples

```
ts_plot(AirPassengers, ts_lag(AirPassengers), title = "The need for glasses")

ts_lag(fdeaths, "1 month")
ts_lag(fdeaths, "1 year")
x <- ts_df(fdeaths)
ts_lag(x, "2 day")
ts_lag(x, "2 min")
ts_lag(x, "-1 day")
```

6.4.3 tsbox::ts_arithmetic()

6.4.3.1 Description

Arithmetic Operators for ts-boxable objects

6.4.3.2 Usage

```
e1 %ts+% e2

e1 %ts-% e2

e1 %ts*% e2

e1 %ts/% e2
```

6.4.3.3 Arguments

e1	ts-boxable time series, an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `tb`
e2	ts-boxable time series, an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `tb`

6.4.3.4 Value

a ts-boxable time series, with the same class as the left input.

6.4.3.5 Examples

```
head(fdeaths - mdeaths)
head(fdeaths %ts-% mdeaths)
head(ts_df(fdeaths) %ts-% mdeaths)
```

6.4.4 tsbox::ts_scale()

6.4.4.1 Description

Subtract mean (*sum(x)/n*) and divide by standard deviation (*sqrt(sum(x^2)/(n-1))*). Based on `base::scale()`.

6.4.4.2 Usage

```
ts_scale(x, center = TRUE, scale = TRUE)
```

6.4.4.3 Arguments

x	ts-boxable time series, an object of class ts, xts, zoo, zooreg, data.frame, data.table, tbl, tbl_t
center	logical
scale	logical

6.4.4.4 Value

a ts-boxable object of the same class as x, i.e., an object of class ts, xts, zoo, zooreg, data.frame, data.table, tbl, tbl_ts, tbl_time, tis, irts or timeSeries.

6.4.4.5 Examples

```
ts_plot(ts_scale((ts_c(airmiles, co2, JohnsonJohnson, discoveries))))
ts_plot(ts_scale(ts_c(AirPassengers, DAX = EuStockMarkets[, "DAX"])))
```

6.4.5 tsbox::ts_trend()

6.4.5.1 Description

Trend estimation that uses stats::loess().

6.4.5.2 Usage

```
ts_trend(x, ...)
```

6.4.5.3 Arguments

x	ts-boxable time series, an object of class ts, xts, zoo, zooreg, data.frame, data.table, tbl, tbl_ts, tbl_time, tis, irts or timeSeries.
...	arguments, passed to stats::loess(): • degree degree of Loess smoothing • span smoothing parameter, if NULL, an automated search performed (see Details)

6.4.5.4 Value

a ts-boxable object of the same class as x, i.e., an object of class ts, xts, zoo, zooreg, data.frame, data.table, tbl, tbl_ts, tbl_time, tis, irts or timeSeries.

6.4.5.5 References

Cleveland, William S., Eric Grosse, and William M. Shyu. "Local regression models." Statistical models in S. Routledge, 2017. 309-376.

6.4.5.6 Examples

```
ts_plot(
  `Raw series` = fdeaths,
  `Loess trend` = ts_trend(fdeaths),
  title = "Deaths from Lung Diseases",
  subtitle = "per month"
)
```

6.4.6 tsbox::ts_bind()

6.4.6.1 Description

Combine time series to a new, single time series. ts_bind combines time series as they are, ts_chain chains them together, using percentage change rates.

6.4.6.2 Usage

```
ts_bind(...)

ts_chain(...)
```

6.4.6.3 Arguments

...	ts-boxable time series, an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `t`

6.4.6.4 Details

In data frame objects, multiple time series are stored in a long data frame. In `ts` and `xts` objects, time series are combined horizontally.

6.4.6.5 Value

a ts-boxable object of the same class as the input, i.e., an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `tbl_time`, `tis`, `irts` or `timeSeries`. If series of different classes are combined, the class of the first series is used (if possible).

6.4.6.6 See Also

ts_c to collect multiple time series

6.4.6.7 Examples

```
ts_bind(ts_span(mdeaths, end = "1975-12-01"), fdeaths)
ts_bind(mdeaths, c(2, 2))
ts_bind(mdeaths, 3, ts_bind(fdeaths, c(99, 2)))
ts_bind(ts_dt(mdeaths), AirPassengers)

## numeric vectors
ts_bind(12, AirPassengers, c(2, 3))
ts_chain(ts_span(mdeaths, end = "1975-12-01"), fdeaths)
```

```
ts_plot(ts_pc(ts_c(
  comb = ts_chain(ts_span(mdeaths, end = "1975-12-01"), fdeaths),
  fdeaths
)))
```

6.4.7 tsbox::ts_index()

6.4.7.1 Description

`ts_index` returns an indexed series, with value of 1 at the `base` date or range. `ts_compound` builds an index from percentage change rates, starting with 1 and compounding the rates.

6.4.7.2 Usage

```
ts_compound(x, denominator = 100)

ts_index(x, base = NULL)
```

6.4.7.3 Arguments

`x`	ts-boxable time series, an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`,
`denominator`	positive number. Set equal to 1 if percentage change rate is given a decimal fraction
`base`	base date, character string, `Date` or `POSIXct`, at which the index is set to 1. If two dates are pr

6.4.7.4 Value

a ts-boxable object of the same class as `x`, i.e., an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `tbl_time`, `tis`, `irts` or `timeSeries`.

6.4.7.5 Examples

```
x <- ts_pc(ts_c(fdeaths, mdeaths))
ts_compound(x)
y <- ts_df(ts_c(fdeaths, mdeaths))
ts_index(y, "1974-02-01")

ts_plot(
  `My Expert Knowledge` = ts_chain(
    mdeaths,
    ts_compound(ts_bind(ts_pc(mdeaths), 15, 23, 33))
  ),
  `So Far` = mdeaths,
  title = "A Very Manual Forecast"
)

## mean of 1974 = 1
ts_index(mdeaths, c("1974-01-01", "1974-12-31"))
```

6.4.8 fcutils::index()

6.4.8.1 Description

Get indexed series (wrapper around tsbox::ts_index())

6.4.8.2 Usage

```
index(x, base_per = as.character(Sys.Date()), base_value = 100)
```

6.4.8.3 Arguments

x	ts-boxable object to be indexed
base_per	base date when the index is set to base_value (see examples). If two dates are provided, the me
base_value	numeric value of the index at base_per (e.g. 1 or 100)

6.4.8.4 Value

indexed object of the same type as the input

6.4.8.5 Examples

```
quarterly_data_example |>
  index(2010.1)
quarterly_data_example |>
  index(c(2010.1, 2010.4))
quarterly_data_example |>
  index(c("2010-01-01", "2010-12-31"), 1)
```

6.4.9 fcutils::ma()

6.4.9.1 Description

Backward looking moving average

6.4.9.2 Usage

```
ma(x, order)
```

6.4.9.3 Arguments

`x`	ts-boxable object
`order`	numeric order (window length) of moving average, includes contemporaneous observation

6.4.9.4 Value

object of the same type as the input containing moving average

6.4.9.5 Examples

```
quarterly_data_example |>
  ma(4) |>
  head()
```

6.4.10 fcutils::yoy_to_lev()

6.4.10.1 Description

Extend a series using year over year growth

6.4.10.2 Usage

```
yoy_to_lev(yoy_gr, hist_lev, smooth_span = 0)
```

6.4.10.3 Arguments

yoy_gr	ts-boxable object containing year over year growth rates
hist_lev	ts-boxable object containing the history in levels for forecast and at least one year of history (i
smooth_span	extent of smoothing between 0-1 (default: 0, no smoothing)

6.4.10.4 Details

This function only works for univariate time series and requires that the growth rates are available for at least the last year of history. Year-over-year growth rates propagate the fluctuations of the base period into the extension period. This can be mitigated by smoothing the extension.

6.4.10.5 Value

object of the same type as hist_lev extended with year over year growth

6.4.10.6 Examples

```
gr <- quarterly_data_example |>
  tsbox::ts_long() |>
  dplyr::filter(id == "E_NF_HI") |>
  tsbox::ts_pcy()
lev <- quarterly_data_example |>
  tsbox::ts_long() |>
  dplyr::filter(id == "ECT_HI")
```

```
res1 <- yoy_to_lev(gr, lev |> dplyr::filter(time <= "2010-01-01"))
res2 <- yoy_to_lev(gr, lev |> dplyr::filter(time <= "2010-01-01"), 1/8)
tsbox::ts_plot(lev, res1, res2)
```

6.4.11 fcutils::mtd_cum()

6.4.11.1 Description

Month to date sum or average

6.4.11.2 Usage

```
mtd_cum(x, avg = TRUE)
```

6.4.11.3 Arguments

x	a ts-boxable object
avg	if TRUE (default), return month to date average, if FALSE, return month to date sum

6.4.11.4 Value

object of the same type as the input containing year to date sum or average

6.4.11.5 Examples

```
daily_data_example |>
  mtd_cum()
test <- daily_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_pick("VAPNS_HI") |>
  mtd_cum()
tsbox::`%ts/%`(test, tsbox::ts_lag(test, "6 months")) |> tail()
```

6.4.12 fcutils::mtd_gr()

6.4.12.1 Description

Month to date growth rate

6.4.12.2 Usage

```
mtd_gr(x)
```

6.4.12.3 Arguments

x	a ts-boxable object

6.4.12.4 Value

object of the same type as the input containing month to date growth rate

6.4.12.5 Examples

```
daily_data_example |>
  mtd_gr() |>
  tail()
```

6.4.13 fcutils::ytd_cum()

6.4.13.1 Description

Year to date sum or average

6.4.13.2 Usage

```
ytd_cum(x, avg = TRUE)
```

6.4.13.3 Arguments

x	a ts-boxable object
avg	if TRUE (default), return year to date average, if FALSE, return year to date sum

6.4.13.4 Value

object of the same type as the input containing year to date sum or average

6.4.13.5 Examples

```
monthly_data_example |>
  ytd_cum()
monthly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_pick("VISNS_HI") |>
  tsbox::ts_xts() |>
  ytd_cum(avg = FALSE) |>
  tsbox::ts_plot()
```

6.4.14 fcutils::ytd_gr()

6.4.14.1 Description

Year to date growth rate

6.4.14.2 Usage

```
ytd_gr(x)
```

6.4.14.3 Arguments

x	a ts-boxable object

6.4.14.4 Value

object of the same type as the input containing year to date growth rate

6.4.14.5 Examples

```
monthly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_pick("VISNS_HI") |>
  tsbox::ts_xts() |>
  ytd_gr() |>
  tail()
```

6.4.15 fcutils::ptd_cum()

6.4.15.1 Description

Period to date sum or average

6.4.15.2 Usage

```
ptd_cum(x, per = "year", avg = TRUE)
```

6.4.15.3 Arguments

x	a ts-boxable object
per	unit of time supplied to floor_date() (for ytd per = ”year” (default), for mtd per = ”month”)
avg	if TRUE (default), retorn period to date average, if FALSE, return period to date sum

6.4.15.4 Value

object of the same type as the input containing period to date sum or average

6.4.15.5 Examples

```
daily_data_example |>
  ptd_cum("week")
test <- daily_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_pick("VAPNS_HI") |>
  ptd_cum("week")
tsbox::`%ts/%`(test, tsbox::ts_lag(test, "4 weeks")) |>
  tsbox::`%ts-%`(1) |>
  tsbox::`%ts*%`(100) |>
  tail()
```

6.4.16 fcutils::ptd_gr()

6.4.16.1 Description

Period to date growth rate

6.4.16.2 Usage

```
ptd_gr(x, per = "year", lag_length = "1 year")
```

6.4.16.3 Arguments

x	a ts-boxable object
per	unit of time supplied to floor_date() (for ytd per = "year" (default), for mtd per = "month")
lag_length	period over which growth is calculated (e.g. "1 year" (default), "3 years", etc. See ?ts_lag() for

6.4.16.4 Value

object of the same type as the input containing period to date growth rate

6.4.16.5 Examples

```
monthly_data_example |>
  ptd_gr() |>
  tail()
monthly_data_example |>
  dplyr::select(time, "VAPNS_HI") |>
  ptd_gr(per = "month", lag_length = "3 years") |>
  tail()
## don't use lag_length = "1 year" with weekly data
daily_data_example |>
  ptd_gr("week")
## lag_length = "52 weeks" instead
daily_data_example |>
  ptd_gr("week", "52 weeks")
## and use lag_length = "364 days" with daily data
daily_data_example |>
  ptd_gr("day", "364 days")
daily_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_pick("VAPNS_HI") |>
  ptd_gr("week", "4 weeks") %>%
  tail()
```

6.4.17 fcutils::pca_to_pc()

6.4.17.1 Description

Convert annualized growth to quarterly growth

6.4.17.2 Usage

```
pca_to_pc(x, freq = 4)
```

6.4.17.3 Arguments

x	ts-boxable object containing annualized growth (in percent)

`freq`	numeric frequency of the time series e.g. 4 for quarterly

6.4.17.4 Value

object of the same type as the input containing quarterly growth (in percent)

6.4.17.5 Examples

```
quarterly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_pca() |>
  pca_to_pc() |>
  tail()
tsbox::ts_c(
  quarterly_data_example |>
    tsbox::ts_long() |>
    tsbox::ts_pca() |>
    pca_to_pc(),
  quarterly_data_example |>
    tsbox::ts_long() |>
    tsbox::ts_pc()
) |>
  dplyr::arrange(id, time) |>
  tsbox::ts_wide()
```

6.4.18 fcutils::pc_to_pca()

6.4.18.1 Description

Convert quarterly growth to annualized growth

6.4.18.2 Usage

```
pc_to_pca(x, freq = 4)
```

6.4.18.3 Arguments

`x`	ts-boxable object containing quarterly growth (in percent)
`freq`	numeric frequency of the time series e.g. 4 for quarterly

6.4.18.4 Value

object of the same type as the input containing annualized growth (in percent)

6.4.18.5 Examples

```
quarterly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_pc() |>
  pc_to_pca() |>
  tail()
tsbox::ts_c(
  quarterly_data_example |>
    tsbox::ts_long() |>
    tsbox::ts_pc() |>
    pc_to_pca(),
  quarterly_data_example |>
    tsbox::ts_long() |>
    tsbox::ts_pca()
) |>
  dplyr::arrange(id, time) |>
  tsbox::ts_wide()
```

6.4.19 fcutils::cagr()

6.4.19.1 Description

Calculate compund annual growth

6.4.19.2 Usage

```
cagr(x)
```

6.4.19.3 Arguments

x	ts-boxable object for which growth is calculated between first and last period

6.4.19.4 Value

a tibble with a single row containing the compound annual growth between the first and last period of x (in percent)

6.4.19.5 Examples

```
quarterly_data_example |>
  cagr()
quarterly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_xts() |>
  cagr()
quarterly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_xts() |>
  tsbox::ts_span("2000-01-01", "2020-01-01") |>
  tsbox::ts_pick("E_NF_HI") |>
  cagr()
```

6.4.20 fcutils::pcmp()

6.4.20.1 Description

Calculate multi-period average growth

6.4.20.2 Usage

```
pcmp(x, lag = 4, comp_freq = 1)
```

6.4.20.3 Arguments

x	ts-boxable object for which growth is calculated (in levels)
lag	number of periods over which growth is calculated
comp_freq	compounding frequency (1 if period by period, 4 if annualized for quarterly data, etc.)

6.4.20.4 Value

object of the same type as the input ontaining the average growth of x (in percent)

6.4.20.5 Examples

```
quarterly_data_example |>
  pcmp(20) |>
  tail()
quarterly_data_example |>
  pcmp(4, 4) |>
  tail()
quarterly_data_example |>
  pcmp(1, 4) |>
  tail()
```

6.4.21 fcutils::%+=%()

6.4.21.1 Description

Warning: Typing x %+=% y/2 returns x <- (x + y)/2. Adding parentheses, i.e. x %+=% (y/2) solves the problem.

6.4.21.2 Usage

```
e1 %+=% e2
```

6.4.21.3 Arguments

e1	first addend (and returned sum)
e2	second addend

6.4.21.4 Value

sum of the two addends replacing the values in the first addend

6.4.21.5 Examples

```
add_QMOD.xts$VISUS_HI[pq(2022.3, 2023.4)] <- add_QMOD.xts$VISUS_HI[pq(2022.3, 2023.4)] +
  c(0.01, -0.04, rep(-0.025, 4))
add_QMOD.xts$VISUS_HI[pq(2022.3, 2023.4)] %+=% c(0.01, -0.04, rep(-0.025, 4)) ## easier on t
```

6.5 Data type conversion

6.5.1 tsbox::ts_ts()

6.5.1.1 Description

tsbox is built around a set of converters, which convert time series stored as `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `tbl_time`, `tis`, `irts` or `timeSeries` to each other.

6.5.1.2 Usage

```
ts_data.frame(x)

ts_df(x)

ts_data.table(x)

ts_dt(x)

ts_tbl(x)
```

```
ts_tibbletime(x)

ts_timeSeries(x)

ts_tis(x)

ts_ts(x)

ts_irts(x)

ts_tsibble(x)

ts_tslist(x)

ts_xts(x)

ts_zoo(x)

ts_zooreg(x)
```

6.5.1.3 Arguments

x	ts-boxable time series, an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `tbl`

6.5.1.4 Details

In data frames, multiple time series will be stored in a 'long' format. tsbox detects a *value*, a *time* and zero to several *id* columns. Column detection is done in the following order:

1. Starting **on the right**, the first first `numeric` or `integer` column is used as **value column**.
2. Using the remaining columns, and starting on the right again, the first `Date`, `POSIXct`, `numeric` or `character` column is used as **time column**. `character` strings are parsed by `anytime::anytime()`. The time stamp, `time`, indicates the beginning of a period.
3. **All remaining** columns are **id columns**. Each unique combination of id columns points to a time series.

Alternatively, the **time** column and the **value** column to be explicitly named as `time` and `value`. If explicit names are used, the column order will be ignored.

Whenever possible, tsbox relies on **heuristic time conversion**. When a monthly `"ts"` time series, e.g., `AirPassengers`, is converted to a data frame, each time stamp (of class `"Date"`) is the first day of the month. In most circumstances, this reflects the actual meaning of the data stored in a `"ts"` object. Technically, of course, this is not correct: `"ts"` objects divide time in period of equal length, while in reality, February is shorter than January. Heuristic conversion is done for frequencies of 0.1 (decades), 1 (years), 4 (quarters) and 12 (month).

For other frequencies, e.g. 260, of `EuStockMarkets`, tsbox uses **exact time conversion**. The year is divided into 260 equally long units, and time stamp of a period will be a point in time (of class `"POSIXct"`).

6.5.1.5 Value

ts-boxable time series of the desired class, i.e., an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `tbl_time`, `tis`, `irts` or `timeSeries`.

6.5.1.6 Examples

```
x.ts <- ts_c(mdeaths, fdeaths)
x.ts
ts_df(x.ts)

suppressMessages(library(dplyr))
ts_tbl(x.ts)

suppressMessages(library(data.table))
ts_dt(x.ts)

suppressMessages(library(xts))
ts_xts(x.ts)

## heuristic time conversion
## 1 month: approx. 1/12 year
ts_df(AirPassengers)

## exact time conversion
## 1 trading day: exactly 1/260 year
ts_df(EuStockMarkets)
```

```
## multiple ids
a <- ts_df(ts_c(fdeaths, mdeaths))
a$type <- "level"
b <- ts_pc(a)
b$type <- "pc"
multi.id.df <- rbind(a, b)

ts_ts(multi.id.df)
ts_plot(multi.id.df)
```

6.5.2 tsbox::ts_long()

6.5.2.1 Description

Functions to reshape multiple time series from 'wide' to 'long' and vice versa. Note that long format data frames are ts-boxable objects, where wide format data frames are not. `ts_long` automatically identifies a **time** column, and uses columns on the left as id columns.

6.5.2.2 Usage

```
ts_long(x)

ts_wide(x)
```

6.5.2.3 Arguments

`x`	a ts-boxable time series, or a wide `data.frame`, `data.table`, or `tibble`.

6.5.2.4 Value

a ts-boxable object of the same class as `x`, i.e., an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `tbl_time`, `tis`, `irts` or `timeSeries`.

6.5.2.5 Examples

```
x <- ts_df(ts_c(mdeaths, fdeaths))
df.wide <- ts_wide(x)
df.wide
ts_long(df.wide)
```

6.5.3 fcutils::conv_long()

6.5.3.1 Description

Convert "ts-boxable" objects into long format

6.5.3.2 Usage

```
conv_long(x, ser_info = FALSE)
```

6.5.3.3 Arguments

`x`	a "tx-boxable" object to be converted
`ser_info`	should additional details be returned (TRUE) or only the long format of x (default: FALSE)

6.5.3.4 Details

This function converts wide data frames and other ts-boxable objects to the long format (wide data frames are not ts-boxable). In addition, it ensures that objects containing a single time series have an id column.

6.5.3.5 Value

returns a ts-boxable object in long format with `id`, `time` and `value` columns. if `ser_info = TRUE`, also returns the following attributes: `was_wide` is `TRUE` if x is a wide data frame, `FALSE` otherwise, and `ser_names` are the names of the series in x.

6.5.3.6 Examples

```
quarterly_data_example |>
  conv_long()
quarterly_data_example |>
  conv_long() |>
  tsbox::ts_tslist() |>
  conv_long()
quarterly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_xts() |>
  conv_long(ser_info = TRUE)
quarterly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_pick("E_NF_HI") |>
  tsbox::ts_xts() |>
  conv_long()
quarterly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_xts() |>
  tsbox::ts_pick("E_NF_HI") |>
  conv_long()
```

6.5.4 fcutils::is_wide()

6.5.4.1 Description

Check if a data frame is in wide format

6.5.4.2 Usage

```
is_wide(x)
```

6.5.4.3 Arguments

x	tibble or data frame

6.5.4.4 Value

returns TRUE for wide format data frame (time and value columns), FALSE otherwise

6.5.4.5 Examples

```
monthly_data_example |> is_wide()
monthly_data_example |>
  tsbox::ts_long() |>
  is_wide()
dat_in <- monthly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_tslist()
wide_df <- is_wide(dat_in)
x_mod <- if (wide_df) tsbox::ts_long(dat_in) else tsbox::ts_tbl(dat_in)
ans <- if (wide_df) tsbox::ts_wide(x_mod) else tsbox::copy_class(x_mod, dat_in)
```

6.5.5 fcutils::set_attr_tslist()

6.5.5.1 Description

Set class attribute to tslist

6.5.5.2 Usage

```
set_attr_tslist(x)
```

6.5.5.3 Arguments

x	list, typically a result of purrr::map() applied to a tslist

6.5.5.4 Details

A purrr::map() function applied to a tslist (obtained by tsbox::ts_tslist()) drops the tslist class attribute. This function resets that attribute.

6.5.5.5 Value

list with class attributes set to list and tslist

6.5.5.6 Examples

```
monthly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_tslist() |>
  purrr::map(~ .x / 1000) |>
  set_attr_tslist() |>
  tsbox::ts_tbl() |>
  tsbox::ts_wide()
```

6.6 Data field operations

6.6.1 tsbox::ts_c()

6.6.1.1 Description

Collect time series as multiple time series.

6.6.1.2 Usage

```
ts_c(...)
```

6.6.1.3 Arguments

`...`	ts-boxable time series, an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `t`

6.6.1.4 Details

In data frame objects, multiple time series are stored in a long data frame. In `ts` and `xts` objects, time series are combined horizontally.

6.6.1.5 Value

a ts-boxable object of the same class as the input, i.e., an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `tbl_time`, `tis`, `irts` or `timeSeries`. If series of different classes are combined, the class of the first series is used (if possible).

6.6.1.6 See Also

ts_bind, to bind multiple time series to a single series.

6.6.1.7 Examples

```
ts_c(mdeaths, fdeaths)

ts_c(ts_df(EuStockMarkets), AirPassengers)

## labeling
x1 <- ts_c(
  `International Airline Passengers` = ts_xts(AirPassengers),
  `Deaths from Lung Diseases` = ldeaths
)
head(x1)
```

6.6.2 tsbox::ts_default()

6.6.2.1 Description

In data frame objects (`data.frame`, `tibble`, `data.table`), tsbox automatically detects the time and the value column. This function changes the column names to the defaults (`time`, `value`), so that auto-detection can be avoided in future operations.

6.6.2.2 Usage

```
ts_default(x)
```

6.6.2.3 Arguments

x	ts-boxable time series, an object of class ts, xts, zoo, zooreg, data.frame, data.table, tbl, tbl_ts, tbl

6.6.2.4 Value

a ts-boxable object of the same class as x, i.e., an object of class ts, xts, zoo, zooreg, data.frame, data.table, tbl, tbl_ts, tbl_time, tis, irts or timeSeries.

6.6.2.5 Examples

```
df <- ts_df(ts_c(mdeaths, fdeaths))
## non-default colnames
colnames(df) <- c("id", "date", "count")
## switch back to default colnames
ts_default(df)
```

6.6.3 tsbox::ts_pick()

6.6.3.1 Description

Pick (and optionally rename) series from multiple time series.

6.6.3.2 Usage

```
ts_pick(x, ...)
```

6.6.3.3 Arguments

x	ts-boxable time series, an object of class ts, xts, zoo, zooreg, data.frame, data.table, tbl, tbl_ts, t
...	character string(s), names of the series to be picked, or integer, with positions. If arguments are named,

6.6.3.4 Value

a ts-boxable object of the same class as `x`, i.e., an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl_ts`, `tbl_time`, `tis`, `irts` or `timeSeries`.

6.6.3.5 Examples

```
## Interactive use

ts_plot(ts_pick(
  EuStockMarkets,
  `My Dax` = "DAX",
  `My Smi` = "SMI"
))
ts_pick(EuStockMarkets, c(1, 2))
ts_pick(EuStockMarkets, `My Dax` = "DAX", `My Smi` = "SMI")

## Programming use
to.be.picked.and.renamed <- c(`My Dax` = "DAX", `My Smi` = "SMI")
ts_pick(EuStockMarkets, to.be.picked.and.renamed)
```

6.6.4 fcutils::get_var()

6.6.4.1 Description

Construct a series name from variable components and retrieve the series

6.6.4.2 Usage

```
get_var(ser_in, env = parent.frame())
```

6.6.4.3 Arguments

`ser_in`	a variable name (character string with substituted expressions)
`env`	environment where the expression should be evaluated

6.6.4.4 Value

variable

6.6.4.5 Examples

```
ser_i <- "_NF"
cnty_i <- "HI"
quarterly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_xts() %$% get_var("E{ser_i}_{cnty_i}")
```

6.6.5 fcutils::rename_udaman()

6.6.5.1 Description

Format series names to udaman format (mnemonic@loc.freq)

6.6.5.2 Usage

```
rename_udaman(ser_in, freq = NULL)
```

6.6.5.3 Arguments

`ser_in`	series names (character "mnemonic_loc", "mnemonic_AT_loc_freq", "mnemonic___loc_freq", mne
`freq`	frequency of the series, required if not contained in the series name (character "D", "W", "M", "Q",

6.6.5.4 Value

series names following udaman convention "mnemonic@loc.freq"

6.6.5.5 Examples

```
rename_udaman(c("E_NF_HI", "ECT_HI", "E_TU_HAW"), freq = "M")
rename_udaman(c("E_NF__HI_M", "ECT__HI_M", "VAP__HAW_W"))
rename_udaman(c("E_NF_AT_HI_M", "ECT_AT_HI_M", "VAP_AT_HAW_W"))
rename_udaman(c("E_NF@HI.M", "ECT@HI.M", "VAP@HAW.W"))
rename_udaman(c("SH_US@HI.M", "SH_JP__HON_M"))
quarterly_data_example |> dplyr::rename_with(~ rename_udaman(., freq = "M"), .cols = -1)
quarterly_data_example |> dplyr::rename_with(rename_udaman, freq = "M", .cols = !time)
quarterly_data_example |>
  tsbox::ts_long() |>
  dplyr::mutate(id = rename_udaman(id, freq = "M")) |>
  tsbox::ts_xts()
```

6.7 Plots

6.7.1 tsbox::ts_ggplot()

6.7.1.1 Description

`ts_ggplot()` has the same syntax and produces a similar plot as `ts_plot()`, but uses the ggplot2 graphic system, and can be customized. With `theme_tsbox()` and `scale_color_tsbox()`, the output of `ts_ggplot` has a similar look and feel.

6.7.1.2 Usage

```
ts_ggplot(..., title, subtitle, ylab = "")

theme_tsbox(base_family = getOption("ts_font", ""), base_size = 12)

colors_tsbox()

scale_color_tsbox(...)

scale_fill_tsbox(...)
```

6.7.1.3 Arguments

...	ts-boxable time series, objects of class ts, xts, data.frame, data.table, or tibble. For scal
title	title (optional)
subtitle	subtitle (optional)
ylab	ylab (optional)
base_family	base font family (can also be set via options)
base_size	base font size

6.7.1.4 Details

Both ts_plot() and ts_ggplot() combine multiple ID dimensions into a single dimension. To plot multiple dimensions in different shapes, facets, etc., use standard ggplot (see examples).

6.7.1.5 See Also

ts_plot(), for a simpler and faster plotting function. ts_dygraphs(), for interactive time series plots.

6.7.1.6 Examples

```
## using the ggplot2 graphic system
p <- ts_ggplot(total = ldeaths, female = fdeaths, male = mdeaths)
p

## with themes for the look and feel of ts_plot()
p + theme_tsbox() + scale_color_tsbox()

## also use themes with standard ggplot
suppressMessages(library(ggplot2))
df <- ts_df(ts_c(total = ldeaths, female = fdeaths, male = mdeaths))
ggplot(df, aes(x = time, y = value)) +
  facet_wrap("id") +
  geom_line() +
  theme_tsbox() +
  scale_color_tsbox()


### tsbox::ts_ggplot()
library(dataseries)
```

```r
dta <- ds(c("GDP.PBRTT.A.R", "CCI.CCIIR"), "xts")
ts_ggplot(ts_scale(ts_span(
  ts_c(
    `GDP Growth` = ts_pc(dta[, "GDP.PBRTT.A.R"]),
    `Consumer Sentiment Index` = dta[, "CCI.CCIIR"]
  ),
  start = "1995-01-01"
))) +
  ggplot2::ggtitle("GDP and Consumer Sentiment", subtitle = "normalized") +
  theme_tsbox() +
  scale_color_tsbox()

### tsbox::ts_ggplot()
```

6.7.2 tsbox::ts_plot()

6.7.2.1 Description

`ts_plot()` is a fast and simple plotting function for ts-boxable time series, with limited customizability. For more theme options, use `ts_ggplot()`.

6.7.2.2 Usage

```r
ts_plot(..., title, subtitle, ylab = "", family = getOption("ts_font", "sans"))
```

6.7.2.3 Arguments

`...`	ts-boxable time series, an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tbl`
`title`	title (optional)
`subtitle`	subtitle (optional)
`ylab`	ylab (optional)
`family`	font family (optional, can also be set via `options`)

6.7.2.4 Details

Both `ts_plot()` and `ts_ggplot()` combine multiple ID dimensions into a single dimension. To plot multiple dimensions in different shapes, facets, etc., use standard ggplot.

Limited customizability of `ts_plot` is available via options. See examples.

6.7.2.5 See Also

`ts_ggplot()`, for a plotting function based on ggplot2. `ts_dygraphs()`, for interactive time series plots. `ts_save()` to save a plot to the file system.

6.7.2.6 Examples

```
ts_plot(
  AirPassengers,
  title = "Airline passengers",
  subtitle = "The classic Box & Jenkins airline data"
)

## naming arguments
ts_plot(total = ldeaths, female = fdeaths, male = mdeaths)

## using different ts-boxable objects
ts_plot(ts_scale(ts_c(
  ts_xts(airmiles),
  ts_tbl(co2),
  JohnsonJohnson,
  ts_df(discoveries)
)))

## customize ts_plot
op <- options(
  tsbox.lwd = 3,
  tsbox.col = c("gray51", "gray11"),
  tsbox.lty = "dashed"
)
ts_plot(
  "Female" = fdeaths,
  "Male" = mdeaths
```

```
)
options(op) ## restore defaults
```

6.7.3 tsbox::ts_save()

6.7.3.1 Description

Save Previous Plot

6.7.3.2 Usage

```
ts_save(
  filename = tempfile(fileext = ".pdf"),
  width = 10,
  height = 5,
  device = NULL,
  open = TRUE
)
```

6.7.3.3 Arguments

`filename`	filename
`width`	width
`height`	height
`device`	device
`open`	logical, should the saved plot be opened?

6.7.3.4 Value

invisible `TRUE`, if successful

6.7.3.5 Examples

```
ts_plot(AirPassengers)
tf <- tempfile(fileext = ".pdf")
ts_save(tf)
unlink(tf)
```

6.7.4 fcutils::plot_1()

6.7.4.1 Description

Interactive plot with level and growth rate

6.7.4.2 Usage

```
plot_1(
  x,
  rng_start = as.character(Sys.Date() - lubridate::years(10)),
  rng_end = as.character(Sys.Date() + lubridate::years(2)),
  height = 300,
  width = 900,
  yoy_gr = TRUE,
  gr_1 = TRUE
)
```

6.7.4.3 Arguments

x	ts-boxable object to plot (e.g. time series of history, oldsol, sol)
rng_start	start of zoom range ("YYYY-MM-DD")
rng_end	end of the zoom range ("YYYY-MM-DD")
height	height of a single panel (px)
width	width of a single panel (px)
yoy_gr	year-over-year (default) or annualized growth
gr_1	only show growth for the first series (default)

6.7.4.4 Value

a dygraph plot

6.7.4.5 Examples

```
monthly_data_example |>
  plot_1()
quarterly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_pick("E_TU_HI", "ECT_HI", "EMN_HI") |>
  plot_1()
```

6.7.5 fcutils::plot_2ax()

6.7.5.1 Description

Interactive lineplot with two axes

6.7.5.2 Usage

```
plot_2ax(
  x,
  rng_start = as.character(Sys.Date() - lubridate::years(10)),
  rng_end = as.character(Sys.Date() + lubridate::years(2)),
  height = 300,
  width = 900
)
```

6.7.5.3 Arguments

x	ts-boxable object to plot (e.g. time series of history, oldsol, sol)
rng_start	start of zoom range ("YYYY-MM-DD")
rng_end	end of the zoom range ("YYYY-MM-DD")
height	height of a single panel (px)
width	width of a single panel (px)

6.7.5.4 Value

a dygraph plot

6.7.5.5 Examples

```
monthly_data_example |>
  plot_2ax()
quarterly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_pick("E_TU_HI", "ECT_HI", "EMN_HI") |>
  plot_2ax()
```

6.7.6 fcutils::plot_comp_2()

6.7.6.1 Description

Two-panel plot of levels and growth rates

6.7.6.2 Usage

```
plot_comp_2(
  x,
  rng_start = as.character(Sys.Date() - lubridate::years(10)),
  rng_end = as.character(Sys.Date() + lubridate::years(2)),
  height = 300,
  width = 900,
  yoy_gr = TRUE,
  gr_bar = FALSE
)
```

6.7.6.3 Arguments

x	ts-boxable object to plot
rng_start	start of the zoom range ("YYYY-MM-DD")
rng_end	end of the zoom range ("YYYY-MM-DD")
height	height of a single panel (px)
width	width of a single panel (px)
yoy_gr	year-over-year (default) or annualized growth
gr_bar	show bars or line (default) for the growth series

6.7.6.4 Value

a list with two dygraph plots (level, growth)

6.7.6.5 Examples

```
monthly_data_example |>
  plot_comp_2()
quarterly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_pick("E_TU_HI", "ECT_HI", "EMN_HI") |>
  plot_comp_2()
```

6.7.7 fcutils::plot_comp_3()

6.7.7.1 Description

Three-panel plot of levels, index, and growth rates

6.7.7.2 Usage

```
plot_comp_3(
  x,
  base_date = as.character(Sys.Date() - lubridate::years(10)),
  rng_start = as.character(Sys.Date() - lubridate::years(10)),
  rng_end = as.character(Sys.Date() + lubridate::years(2)),
  height = 300,
  width = 900,
  yoy_gr = TRUE,
  gr_bar = FALSE
)
```

6.7.7.3 Arguments

x	ts-boxable object to plot
base_date	base period for the indexed series ("YYYY-MM-DD")
rng_start	start of the zoom range ("YYYY-MM-DD")

rng_end	end of the zoom range ("YYYY-MM-DD")
height	height of a single panel (px)
width	width of a single panel (px)
yoy_gr	year-over-year (default) or annualized growth
gr_bar	show bars or line (default) for the growth series

6.7.7.4 Value

a list with three dygraph plots (level, index, growth)

6.7.7.5 Examples

```
monthly_data_example |>
  plot_comp_3()
quarterly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_pick("E_TU_HI", "ECT_HI", "EMN_HI") |>
  plot_comp_3()
```

6.7.8 fcutils::plot_fc()

6.7.8.1 Description

Interactive plot with level and growth rate for forecast series

6.7.8.2 Usage

```
plot_fc(
  x,
  rng_start = as.character(Sys.Date() - lubridate::years(10)),
  rng_end = as.character(Sys.Date() + lubridate::years(2)),
  add_table = TRUE,
  table_start = rng_start,
  table_end = rng_end,
  height = 300,
  width = 900,
```

```
  yoy_gr = TRUE
)
```

6.7.8.3 Arguments

x	ts-boxable object to plot (min 1, max 3 time series) (e.g. current fcst, old fcst, history)
rng_start	start of zoom range ("YYYY-MM-DD")
rng_end	end of the zoom range ("YYYY-MM-DD")
add_table	should a data table be appended to the plot? (default = TRUE)
table_start	start of table range ("YYYY-MM-DD") (all data = NULL, default = rng_start)
table_end	end of table range ("YYYY-MM-DD") (all data = NULL, default = rng_end)
height	height of a single panel (px)
width	width of a single panel (px)
yoy_gr	year-over-year (default) or annualized growth

6.7.8.4 Value

a dygraph plot

6.7.8.5 Examples

```
monthly_data_example |>
  plot_fc()
quarterly_data_example |>
  tsbox::ts_long() |>
  tsbox::ts_pick("E_TU_HI", "ECT_HI", "EMN_HI") |>
  plot_fc()
```

6.7.9 fcutils::save_plot_list()

6.7.9.1 Description

Save a list of interactive plots to html

6.7.9.2 Usage

```
save_plot_list(plot_list, save_loc)
```

6.7.9.3 Arguments

plot_list	a list of plots generated by fcutils::plot_xxxx() functions
save_loc	location to save the plots to, including file name

6.7.9.4 Value

nothing (silently save the html to a user defined location)

6.7.9.5 Examples

```
## hold the plots in a list
plot_out <- list()
for (i in monthly_data_example[2:3] |> names()) {
  plot_out[[i]] <- plot_1(
    monthly_data_example |> tsbox::ts_long() |>
      dplyr::filter(stringr::str_detect(id, i)),
    rng_start = as.character(Sys.Date() - lubridate::years(5)),
    rng_end = as.character(Sys.Date() + lubridate::years(7)),
    width = 1500, height = 650, yoy_gr = TRUE
  )
}
## specify location of the output
save_loc <- stringr::str_c("~/Downloads/plots_", Sys.Date(), ".html")
## combine a list of plots into a single html

plot_out |> save_plot_list(save_loc)
```

6.7.10 uherotheme::uhero_theme()

6.7.10.1 Description

Modifies the ggplot minimal theme to fit the style used for UHERO reports/presentations.

6.7.10.2 Usage

```
uhero_theme(layout = FALSE)
```

6.7.10.3 Arguments

layout	A boolean to indicate whether or not the theme is being applied to a plot that is to be used in a UH

6.7.10.4 Examples

```
plot <- ggplot2::ggplot(ggplot2::mpg) + uhero_theme()
```

6.7.11 uherotheme::uhero_scale_nums()

6.7.11.1 Description

This can be passed in to the labels parameter of ggplot scales like scale_x_continous() or scale_y_continuous() If the max value of the scale is larger than 1000, the tick labels will be scaled down with a suffix added to the maximum value. For max values of at least 10^3 but less than 10^6, tick labels will be divided by 10^3 with "K" added to the label for the maximum value. For max values of at least 10^6 but less than 10^9, tick labels will be divided by 10^6 with "M" added to the label for the maximum value. For max values of at least 10^9 but less than 10^12, tick labels will be divided by 10^9 with "B" added to the label for the maximum value.

6.7.11.2 Usage

```
uhero_scale_nums(
  x,
  scale_limit = max(x, na.rm = TRUE),
  prefix = "",
  percent = FALSE,
  ...
)
```

6.7.11.3 Arguments

x	Passed in from labels function.
scale_limit	Defaults to the max value of the scale.
prefix	This is optional and defaults to an empty string. This can be used to add things like currency
percent	Boolean that defaults to false. Set the value to true to add a "%" to the maximum value tick l
...	Additional arguments that can be passed to R's format().

6.7.11.4 Value

Returns a formatted string for the tick label.

6.7.11.5 Examples

```
set.seed(1)
df <- data.frame(
  x = rnorm(10) * 100000,
  y = seq(0, 1, length.out = 10)
)
ggplot2::ggplot(df, ggplot2::aes(x, y)) + ggplot2::geom_point() +
ggplot2::scale_y_continuous(labels = function(x) uhero_scale_nums(x, percent = TRUE)) +
ggplot2::scale_x_continuous(labels = function(x) uhero_scale_nums(x))
```

6.7.12 uherotheme::uhero_colors()

6.7.12.1 Description

Hex codes for a given color in the UHERO colors

6.7.12.2 Usage

```
uhero_colors(...)
```

6.7.12.3 Arguments

...	Names of the UHERO colors. There are 14 total: "blue", "orange", "green", "purple", "cyan", "gray", "r

6.7.12.4 Value

A list of colors with their HEX codes.

6.7.12.5 Examples

```
uhero_colors("blue")
uhero_colors("blue", "light orange")
```

6.7.13 uherotheme::uhero_pal()

6.7.13.1 Description

A palette generator for UHERO colors.

6.7.13.2 Usage

```
uhero_pal(palette = "primary", discrete = TRUE, reverse = FALSE, ...)
```

6.7.13.3 Arguments

`palette`	A string that defaults to "primary". This can also be set to either "secondary" or "all".
`discrete`	A boolean that defaults to TRUE. Set to FALSE for a continuous scale.
`reverse`	A boolean that defaults to FALSE. Set to TRUE to reverse the color scale.
`...`	Additional parameters that can be passed to `colorRampPalette`

6.7.13.4 Examples

```
uhero_pal()
uhero_pal("secondary", discrete = TRUE, reverse = TRUE)
```

6.7.14 uherotheme::uhero_scale_colour()

6.7.14.1 Description

Uses `ggplot2::discrete_scale` for the color aesthetic for discrete scales and `ggplot2::scale_colour_gradien` for the color aesthetic on continuous scales. `uhero_scale_color` is available as an alias.

6.7.14.2 Usage

```
uhero_scale_colour(palette = "primary", discrete = TRUE, reverse = FALSE, ...)
```

6.7.14.3 Arguments

`palette`	A string that defaults to "primary". This can also be set to either "secondary" or "all".
`discrete`	A boolean that defaults to TRUE. Set to FALSE for a continuous scale.
`reverse`	A boolean that defaults to FALSE. Set to TRUE to reverse the color scale.
`...`	Additional parameters that can be passed to `ggplot2::discrete_scale` or `ggplot2::scale_fil`

6.7.14.4 Examples

```
ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(hwy, class, color = class)) +
ggplot2::geom_point(show.legend = FALSE) +
uhero_scale_colour()
```

6.7.15 uherotheme::uhero_scale_fill()

6.7.15.1 Description

Uses `ggplot2::discrete_scale` for the fill aesthetic for discrete scales and `ggplot2::scale_fill_gradientn` for the fill aesthetic on continuous scales.

6.7.15.2 Usage

```
uhero_scale_fill(palette = "primary", discrete = TRUE, reverse = FALSE, ...)
```

6.7.15.3 Arguments

palette	A string that defaults to "primary". This can also be set to either "secondary" or "all".
discrete	A boolean that defaults to TRUE. Set to FALSE for a continuous scale.
reverse	A boolean that defaults to FALSE. Set to TRUE to reverse the color scale.
...	Additional parameters that can be passed to ggplot2::discrete_scale or ggplot2::scale_fil

6.7.15.4 Examples

```
ggplot2::ggplot(ggplot2::mpg, ggplot2::aes(hwy, fill = class)) +
ggplot2::geom_bar(show.legend = FALSE) +
uhero_scale_fill(palette = 'secondary')
```

6.7.16 uherotheme::uhero_scale_colour_diverge()

6.7.16.1 Description

Uses ggplot2::scale_color_gradient2. uhero_scale_color_diverge is available as an alias.

6.7.16.2 Usage

```
uhero_scale_colour_diverge(high = "blue", low = "orange", ...)
```

6.7.16.3 Arguments

high	A string that defaults to "blue". Can accept any of the other colors in uhero_color_list.
low	A string that defaults to "orange". Can accept any of the other colors in uhero_color_list.
...	Any other parameters that can be passed to ggplot2::scale_color_gradient2

6.7.16.4 Examples

```
set.seed(1)
df <- data.frame(
 x = runif(100),
 y = runif(100),
 z1 = rnorm(100),
 z2 = abs(rnorm(100))
)
ggplot2::ggplot(df, ggplot2::aes(x, y)) +
ggplot2::geom_point(ggplot2::aes(colour = z1)) +
uhero_scale_colour_diverge()
```

6.7.17 uherotheme::uhero_scale_fill_diverge()

6.7.17.1 Description

Uses `ggplot2::scale_fill_gradient2`

6.7.17.2 Usage

```
uhero_scale_fill_diverge(high = "blue", low = "orange", ...)
```

6.7.17.3 Arguments

`high`	A string that defaults to "blue". Can accept any of the other colors in `uhero_color_list`.
`low`	A string that defaults to "orange". Can accept any of the other colors in `uhero_color_list`.
`...`	Any other parameters that can be passed to `ggplot2::scale_fill_gradient2`

6.7.18 uherotheme::draw_fcast_layout()

6.7.18.1 Description

This modifies the size of the view port window to more accurately reflect the size of the figure including the placement of elements like data labels that would be used in a forecast layout.

6.7.18.2 Usage

```
draw_fcast_layout(plot, w = 4.5, h = 2.45)
```

6.7.18.3 Arguments

plot	Plot object
w	Width of the view port in inches, defaults to 4.5
h	Height of the view port in inches, defaults to 2.45

6.7.18.4 Examples

```
df <- data.frame(
  x = rnorm(10) * 100000,
  y = seq(0, 1, length.out = 10)
)
plot <- ggplot2::ggplot(df, ggplot2::aes(x, y)) + ggplot2::geom_point()
draw_fcast_layout(plot)
```

6.7.19 uherotheme::draw_report_layout()

6.7.19.1 Description

This modifies the size of the view port window to more accurately reflect the size of the figure including the placement of elements like data labels that would be used in a UHERO report layout. Sometimes charts may need to be a different size, so the function does accept parameters to change the width and height.

6.7.19.2 Usage

```
draw_report_layout(plot, w = 5.6931, h = 4)
```

6.7.19.3 Arguments

plot	Plot object
w	Width of the view port in inches, defaults to 5.6931
h	Height of the view port in inches, defaults to 4

6.7.19.4 Examples

```
df <- data.frame(
  x = rnorm(10) * 100000,
  y = seq(0, 1, length.out = 10)
)
plot <- ggplot2::ggplot(df, ggplot2::aes(x, y)) + ggplot2::geom_point()
draw_report_layout(plot)
```

6.7.20 uherotheme::export_fcast_layout()

6.7.20.1 Description

Uses ggplot2::ggsave to save a copy of the chart. By default, charts are sized at 4.5 x 2.45 inches. Sometimes charts may need to be larger, so the export function does accept parameters to change the width and height. Please try not to exceed 5 inches for the width. If the exported file is intended for use in a forecast layout, please use a '.svg', '.pdf', or '.eps' extension.

6.7.20.2 Usage

```
export_fcast_layout(file_name, forecast_plot, w = 4.5, h = 2.45, u = "in", ...)
```

6.7.20.3 Arguments

file_name	A string for the file name, including the extension.
forecast_plot	Ggplot plot object.
w	Integer - width of the exported image, defaults to 4.5
h	Integer - height of the exported image, defaults to 2.45
u	A string for the units, defaults to ”in” for inches

...	Additional parameters that can be passed to ggplot2::ggsave

6.7.20.4 Examples

```
df <- data.frame(
  x = rnorm(10) * 100000,
  y = seq(0, 1, length.out = 10)
)
plot <- ggplot2::ggplot(df, ggplot2::aes(x, y)) + ggplot2::geom_point()
export_fcast_layout('plot.svg', plot)
```

6.7.21 uherotheme::export_report_layout()

6.7.21.1 Description

Uses `ggplot2::ggsave` to save a copy of the chart. By default, charts are sized at 5.6931 x 4 inches. Sometimes charts may need to be larger, so the export function does accept parameters to change the width and height. Please use a '.svg', '.pdf', or '.eps' extension when exporting for a report layout.

6.7.21.2 Usage

```
export_report_layout(file_name, plot, w = 5.6931, h = 4, u = "in", ...)
```

6.7.21.3 Arguments

`file_name`	A string for the file name, including the extension.
`plot`	Ggplot plot object.
`w`	Integer - width of the exported image, defaults to 5.6931
`h`	Integer - height of the exported image, defaults to 4
`u`	A string for the units, defaults to ”in” for inches
`...`	Additional parameters that can be passed to ggplot2::ggsave

6.7.21.4 Examples

```
df <- data.frame(
  x = rnorm(10) * 100000,
  y = seq(0, 1, length.out = 10)
)
plot <- ggplot2::ggplot(df, ggplot2::aes(x, y)) + ggplot2::geom_point()
export_report_layout('plot.svg', plot)
```

6.7.22 uherotheme::export_plot()

6.7.22.1 Description

Uses `ggplot2::ggsave` to save a copy of the chart. By default the exports are sized at 1920 x 1080 pixels.

6.7.22.2 Usage

```
export_plot(file_name, plot, w = 1920, h = 1080, u = "px", ...)
```

6.7.22.3 Arguments

`file_name`	A string for the file name, including the extension.
`plot`	Ggplot plot object.
`w`	Integer - width of the exported image, defaults to 1920
`h`	Integer - height of the exported image, defaults to 1080
`u`	A string for the units, defaults to ”px” for pixels
`...`	Additional parameters that can be passed to ggplot2::ggsave

6.7.22.4 Examples

```
df <- data.frame(
  x = rnorm(10) * 100000,
  y = seq(0, 1, length.out = 10)
)
```

```
plot <- ggplot2::ggplot(df, ggplot2::aes(x, y)) + ggplot2::geom_point()
export_report_layout('plot.png', plot)
```

6.8 tsbox extensions

6.8.1 tsbox::ts_()

6.8.1.1 Description

`ts_` turns an existing function into a function that can deal with ts-boxable time series objects.

6.8.1.2 Usage

```
load_suggested(pkg)

ts_(fun, class = "ts", vectorize = FALSE, reclass = TRUE)

ts_apply(x, fun, ...)
```

6.8.1.3 Arguments

`pkg`	external package, to be suggested (automatically added by `ts_`) `predict()`. (See examples)
`fun`	function, to be made available to all time series classes
`class`	class that the function uses as its first argument
`vectorize`	should the function be vectorized? (not yet implemented)
`reclass`	logical, should the new function return the same same ts-boxable output as imputed?
`x`	ts-boxable time series, an object of class `ts`, `xts`, `zoo`, `zooreg`, `data.frame`, `data.table`, `tbl`, `tb`
`...`	arguments passed to subfunction

6.8.1.4 Details

The `ts_` function is a constructor function for tsbox time series functions. It can be used to wrap any function that works with time series. The default is set to R base `"ts"` class. `ts_` deals with the conversion stuff, 'vectorizes' the function so that it can be used with multiple time series.

6.8.1.5 Value

A function that accepts ts-boxable time series as an input.

6.8.1.6 See Also

ts_examples, for a few useful examples of functions generated by `ts_`.

Vignette on how to make arbitrary functions ts-boxable.

6.8.1.7 Examples

```
ts_(rowSums)(ts_c(mdeaths, fdeaths))
ts_plot(mean = ts_(rowMeans)(ts_c(mdeaths, fdeaths)), mdeaths, fdeaths)
ts_(function(x) predict(prcomp(x)))(ts_c(mdeaths, fdeaths))
ts_(function(x) predict(prcomp(x, scale = TRUE)))(ts_c(mdeaths, fdeaths))
ts_(dygraphs::dygraph, class = "xts")

## attach series to serach path
ts_attach <- ts_(attach, class = "tslist", reclass = FALSE)
ts_attach(EuStockMarkets)
ts_plot(DAX, SMI)
detach()
```

6.8.2 tsbox::ts_examples()

6.8.2.1 Description

Example Functions, Generated by ts_. `ts_prcomp` calculates the principal components of multiple time series, `ts_dygraphs` generates an interactive graphical visualization, `ts_forecast` return an univariate forecast, `ts_seas` the seasonally adjusted series. `ts_na_interpolation` imputes missing values.

6.8.2.2 Usage

```
ts_prcomp(x, ...)

ts_dygraphs(x, ...)

ts_forecast(x, ...)

ts_seas(x, ...)

ts_na_interpolation(x, ...)
```

6.8.2.3 Arguments

x	ts-boxable time series, an object of class ts, xts, zoo, zooreg, data.frame, data.table, tbl, tbl_ts, t
...	further arguments, passed to the underlying function. For help, consider these functions, e.g., stats::prco

6.8.2.4 Details

With the exception of ts_prcomp, these functions depend on external packages.

6.8.2.5 Value

a ts-boxable object of the same class as x, i.e., an object of class ts, xts, zoo, zooreg, data.frame, data.table, tbl, tbl_ts, tbl_time, tis, irts or timeSeries.

6.8.2.6 See Also

Vignette on how to make arbitrary functions ts-boxable.

6.8.2.7 Examples

```
ts_plot(
  ts_scale(ts_c(
    Male = mdeaths,
    Female = fdeaths,
    `First principal compenent` = -ts_prcomp(ts_c(mdeaths, fdeaths))[, 1]
  )),
```

```
  title = "Deaths from lung diseases",
  subtitle = "Normalized values"
)

ts_plot(ts_c(
  male = mdeaths, female = fdeaths,
  ts_forecast(ts_c(`male (fct)` = mdeaths, `female (fct)` = fdeaths))
),
title = "Deaths from lung diseases",
subtitle = "Exponential smoothing forecast"
)

ts_plot(
  `Raw series` = AirPassengers,
  `Adjusted series` = ts_seas(AirPassengers),
  title = "Airline passengers",
  subtitle = "X-13 seasonal adjustment"
)


## See ?imputeTS::na_interpolation for options
dta <- ts_c(mdeaths, fdeaths)
dta[c(1, 3, 10), c(1, 2)] <- NA
head(ts_na_interpolation(dta, option = "spline"))

ts_dygraphs(ts_c(mdeaths, EuStockMarkets))
```

6.9 bimets and gets utilities

6.9.1 fcutils::set_tsrange()

6.9.1.1 Description

Set tsrange for behavioral equations to available data range

6.9.1.2 Usage

```
set_tsrange(model_w_dat, max_lag = 4, eqns = NULL)
```

6.9.1.3 Arguments

`model_w_dat`	bimets model (with data) to be estimated
`max_lag`	the largest lag (default = 4) in the model (to offset starting point for estimation)
`eqns`	names of behavioral equations to set tsrange for (default = NULL: all equations)

6.9.1.4 Details

Find periods where all variables in the equation are available. Shift beginning of the sample by max_lag periods. Set the tsrange for each equation (used in estimation).

6.9.1.5 Value

bimets model with tsrange set for estimation

6.9.1.6 Examples

```
set_tsrange(scen_model_dat, 4)
```

6.9.2 fcutils::update_eqs()

6.9.2.1 Description

Update a bimets model with new/modified equations

6.9.2.2 Usage

```
update_eqs(model_1, model_2, eqList)
```

6.9.2.3 Arguments

`model_1`	original estimated bimets model
`model_2`	estimated bimets model containing updates (only updated equations need to be estimated)
`eqList`	names of updated behavioral equations (vector of strings), others taken from model_1 (equations m

6.9.2.4 Details

Start by making a copy of the original model's equations (txt file). Re-specify some equations, add new equations, and remove not needed equations. Load the new model as model_2 and estimate the modified/new equations (ok to estimate all). Replace the equations in model_2 that should remain the same as in model_1 by the estimated equations from model_1. Equations that are to remain unchanged have to be present in both model_1 and model_2, and not present in eqList.

6.9.2.5 Value

estimated bimets model containing updates

6.9.2.6 Examples

```
update_eqs(scen_model_1_est, scen_model_2_est, c("E_NF_AT_HI_Q", "Y_R_AT_HI_Q"))
```

6.9.3 fcutils::extract_data()

6.9.3.1 Description

Parse gets output and extract underlying data (GETS model development)

6.9.3.2 Usage

```
extract_data(model_in, y_name)
```

6.9.3.3 Arguments

model_in	a model estimated by arx, isat, or getsm
y_name	the actual name of the y variable

6.9.3.4 Value

an xts containing the model variables

6.9.3.5 Examples

```
## save the data associated with a gets model
```

6.9.4 fcutils::model_equation()

6.9.4.1 Description

Parse lm() output and convert into bimets equation (GETS model development)

6.9.4.2 Usage

```
model_equation(model, ...)
```

6.9.4.3 Arguments

`model`	a model estimated by lm() (lm object)
`...`	arguments to format the coefficients e.g. digits = 3

6.9.4.4 Value

a character vector containing the estimated equation (1) and bimets components (2:4)

6.9.4.5 Examples

```
## this function combines coefficient estimates and variable names into an equation
## in vector element 1 and into bimets components in vector elements 2-4.
## https://stats.stackexchange.com/questions/63600/
## how-to-translate-the-results-from-lm-to-an-equation
data("UKDriverDeaths", package = "datasets")
uk <- log10(UKDriverDeaths)
dfm <- dynlm::dynlm(uk ~ L(uk, 1:3) + L(log(uk), c(5:6, 12)))
model_equation(dfm)
## (1) "uk = - 0.12255631 + 0.42870091 * L(uk, 1:3)1 + 0.06306114 * L(uk, 1:3)2 - 0.09778518
```

```
## L(uk, 1:3)3 + 0.37480999 * L(log(uk), c(5:6, 12))5 - 0.22709846 * L(log(uk), c(5:6, 12))6
## 1.62340449 * L(log(uk), c(5:6, 12))12"
## (2) "BEHAVIORAL> uk"
## (3) "EQ> uk = b0 + b1 * TSLAG(uk, 1) + b2 * TSLAG(uk, 2) + b3 * TSLAG(uk, 3) + b4 *
## TSLAG(LOG(uk), 5) + b5 * TSLAG(LOG(uk), 6) + b6 * TSLAG(LOG(uk), 12)"
## (4) "COEFF> b0 b1 b2 b3 b4 b5 b6"
### fcutils::model_equation()
dfm <- dynlm::dynlm(d(log(uk)) ~ L(uk, c(1, 11, 12)), start = c(1975, 1), end = c(1982, 12))
model_equation(dfm)
## (1) "d(log(uk)) = 0.1018542 - 0.2379287 * L(uk, c(1, 11, 12))1 + 0.0368355 *
## L(uk, c(1, 11, 12))11 + 0.1689896 * L(uk, c(1, 11, 12))12"
## (2) "BEHAVIORAL> TSDELTA_LOG_uk"
## (3) "EQ> TSDELTA(LOG(uk)) = b0 + b1 * TSLAG(uk, 1) + b2 * TSLAG(uk, 11) + b3 * TSLAG(uk,
## (4) "COEFF> b0 b1 b2 b3"
```

6.10 fcutils constants

The fcutils package assumes a date range for data sets. Unless specified by the user, the start and end date are set by default to:

```
bnk_start = "1970-01-01"
bnk_end = "2060-12-31"
```

7 Best practices for time series data manipulation

Use capital letters for series names. Special characters in variable names require putting the name between backticks (e.g. `N@US.A`). Eliminate special characters at the time of loading the data from udaman with the `rename` argument of the `fcutils::get_series()` function (Section 6.1) or using a long tibble.

```
hist_q_mod <- hist_q %>%
  tsbox::ts_tbl() %>%
  dplyr::mutate(id = stringr::str_replace_all(id, c("@" = "_AT_", "\\." = "_")))
```

Revert back to the udaman format with the `fcutils::rename_udaman()` function (Section 6.6).

Use the *xts* format whenever you need a time series object. Observations in a multivariate xts can be accessed by time and series name in two ways: `mul_var_xts[time, ser_name]` or `mul_var_xts$ser_name[time]`.

Make sure all series are defined on the same range (default start = bnk_start, end = bnk_end). Take advantage of the `fcutils::make_xts()` function and its defaults, e.g. start and end period (Section 6.1).

```
import_xts <- readr::read_csv(here::here("data/raw", stringr::str_glue("{exp_id_a}.csv"))) %
  dplyr::arrange(time) %>%
  tsbox::ts_long() %>%
  tsbox::ts_xts() %>%
  tsbox::ts_c(
    temp = fcutils::make_xts(per = "year") # temporary variable to force start and end in im
  ) %>%
  magrittr::extract(, str_subset(colnames(.), "temp", negate = TRUE)) # remove temp
```

If referring directly to a series with a static name, use the `bank$series` notation (this can be used on both the right and the left hand side of the assignment, while `bank[, series]` can only be used for existing series in bank).

```
# find the last value in history
dat_end <- fcutils::find_end(hist_q$N_AT_US_Q)
# same as
dat_end <- fcutils::find_end(hist_q[, N_AT_US_Q])
```

Use `[fcutils::p()]` to select a period in xts objects, otherwise use `tsbox::ts_span()` (both in Section 6.2).

```
# extend series with addfactored level
sol_q$N_AT_US_SOLQ <- hist_q$N_AT_US_Q[p("", dat_end)] %>%
  tsbox::ts_bind(sol_q$NCEN_AT_US_SOLQ[p(dat_end, "")] +
    as.numeric(sol_q$N_AT_US_SOLQ_ADDLEV[dat_end]))

# addfactor for growth
sol_q$N_AT_US_SOLQ_ADDGRO[p(dat_end + fcutils::qtrs(1), dat_end + fcutils::qtrs(4))] <- -0.35

# extend history using growth rate
sol_q$N_AT_US_SOLQ <- sol_q$N_AT_US_SOLQ[p("", dat_end)] %>%
  tsbox::ts_chain(tsbox::ts_compound(sol_q$N_AT_US_SOLQ_GRO[p(dat_end, "")]))
```

The `bank[,seriesname]` notation only works for *existing* xts series on the left of the assignment (it can also be used on the right). `seriesname` can be determined at runtime

```
# initialize the lhs series in the "bank"
hist_a$temp <- make_xts()
names(hist_a)[names(hist_a) == "temp"] <- stringr::str_glue("E{ser_i}_AT_{cnty_i}_ADD")

# calculate expression and assign to lhs
hist_a[, stringr::str_glue("E{ser_i}_AT_{cnty_i}_SH")] <-
  (hist_a[, stringr::str_glue("E{ser_i}_AT_{cnty_i}")] / hist_a[, stringr::str_glue("E_NF_AT_
```

Alternatively, make multiple series in *bank* available by `magrittr::%$%` and retrieve inividual series by `fcutils::get_var()` on the right (Section 6.6).

```
hist_a[, stringr::str_glue("E{ser_i}_AT_{cnty_i}_SH")] <- hist_a %$%
  (fcutils::get_var("E{ser_i}_AT_{cnty_i}") / fcutils::get_var("E_NF_AT_{cnty_i}"))
```

Conversion to bimets format requires data in a particular *tslist* format. Convert xts to tslist using `tsbox::ts_tslist()`. The `fcutils::set_attr_tslist()` function adds a "tslist" attribute to a list (both in Section 6.5).

```
# store series as tslist
hist_a_lst <- hist_a %>%
  tsbox::ts_tslist() %>%

# convert series to bimets format
hist_a_bimets <- hist_a_lst %>%
  purrr::map(bimets::as.bimets)

# bimets strips the attributes, need to reset them for further manipulation by tsbox
hist_a <- hist_a_bimets %>%
  fcutils::set_attr_tslist() %>%
  tsbox::ts_xts()
```

For series collected in a `tslist` on the left of the assignment use the `bank[[seriesname]]` notation (it can also be used on the right). Here the lhs series `seriesname` does not need to exist, and it might be easier to work with `tslist` than `xts` when variable names are determined at runtime.

```
# similar to above with a tslist variable
hist_a_lst[[stringr::str_glue("E{ser_i}_AT_{cnty_i}_ADD")]] <- hist_a_lst %$%
  (fcutils::get_var("E{ser_i}_AT_{cnty_i}") - fcutils::get_var("E_NF_AT_{cnty_i}"))
```

7.1 Harness the power of tsbox

Use the converter functions in tsbox to shift between various data types (`ts_tbl()`, `ts_xts()`, `ts_ts()`, `ts_tslist()`) and reshaping to the long and wide format (`ts_long()`, `ts_wide()`). `tsbox` further contains funtions for time period selection (`ts_span()`), merging and extension operations (`ts_c()`, `ts_bind()`, `ts_chain()`), transformations (`ts_lag()`, `ts_pc()`, `ts_pca()`, `ts_pcy()`, `ts_diff()`, `ts_diffy()`), and index construction (`ts_compound()`, `ts_index()`). These functions are described in Section 6. Consider these tsbox functions before turning to solutions that are specific to the `xts`, `ts`, `dplyr` or `tidyr` packages.

8 Model selection and simulation

The model selection process can be run line-by-line from an R script directly (*R/gets_model_select.R*) or via sourcing an Rmd document (*notes/gets_model_select.Rmd*) which collects all model selection results in an easier to digest html file. Running the full script (source) takes about 1 minute.

8.1 Main user settings

Before diving into the automated steps, users must specify several key parameters that control the model selection process:

- `mselect_start` and `mselect_end`: Define the start and end dates of the period used for model selection. This is the sample over which the *gets* algorithm will evaluate potential models.

- `est_end`: Specifies the end of the period used for estimation. While the model selection process might consider data up to `mselect_end`, the final model can be re-estimated using a shorter sample ending at `est_end`. This is useful when recent data is deemed less reliable or structurally different.

- `fcst_start` and `fcst_end`: Define the start and end dates of the quasi-forecast period. This is the period over which the model's forecasting performance will be evaluated. The model is simulated forward from `fcst_start`, and the simulated values are compared against actual data (if available) up to `fcst_end`.

- `max_lag`: Specifies the maximum number of lags to be considered for each predictor variable in the model. For example, if `max_lag` is 4, the model selection process will consider the current value and up to four lagged values of each predictor.

- `yvar_name`: The name of the response variable to be forecasted. This must correspond to the name of a variable in the downloaded dataset. For example: `yvar_name <- "E_NF_AT_HI_QL"` specifies the log of non-farm employment in Hawaii as the target variable.

- **xvar_list**: A vector of names of potential predictor variables. These can include economic indicators, other related time series, or even transformations of the response variable itself. For instance:

```
# list of candidate variables (logs are denoted by _QL extension)
xvar_list <- c(
  "Y_R_AT_HI_QL", # Log of real income in Hawaii
  "VIS_AT_HI_QL", # Log of visitor arrivals in Hawaii
  "CPI_AT_HON_QL" # Log of CPI in Honolulu
)
```

If no additional regressors are considered set `xvar_list <- c()`.

8.2 Data preparation (`tidyverse`)

The data preparation stage involves several steps to transform the raw data downloaded from UDAMAN into a format suitable for model selection:

8.2.1 Data Download and Initial Processing

- Download all series used in the model selection process from UDAMAN (about 500 rows and 1200 columns) and eliminate special characters from the series names.
- The data is converted to a long format using `ts_long()`.

8.2.2 Log Transformation

- All variables in the dataset are log-transformed using. This is a common practice in time series analysis to stabilize variance and linearize relationships. The names of the log-transformed variables are appended with "L". For example, `E_NF_AT_HI` becomes `E_NF_AT_HI_L`. Note that, depending on the data, this may result in `NaN` values which will be handled by the `gets` algorithm during model selection.

```
# take the log of all series (some will produce NaNs)
hist_ql <- hist_q %>%
  mutate(value = log(value)) %>%
  mutate(id = str_c(id, "L"))
```

8.2.3 Indicator Variable Creation

- Indicator variables, also known as dummy variables, are created to capture the effects of specific events or periods. The script allows for loading previously saved indicators (`indicators_from_disk`) or creating them from scratch.

- **Impulse Indicator Series (IIS)**: These are binary variables that take the value 1 in a specific quarter and 0 otherwise. They are used to capture the impact of one-time events or shocks. These dummies are useful to control for one-time temporary shocks such as hurricanes, policy changes or one-time bonuses paid out in a particular quarter. Without controlling for these events, the model would try to fit these extraordinary data points using regular predictors, which may lead to biased estimates.

- **Step Indicator Series (SIS)**: These variables take the value 0 before a certain date and 1 thereafter. They are used to model structural breaks or permanent level shifts in the time series. A typical example of when these indicator series might be useful is the case of a permanent policy change, introduction of a major competitor, or a permanent change in consumer preferences. By introducing a SIS, the model can account for these changes without biasing estimates for the entire sample.

- **Season Dummies**: These variables capture the seasonal patterns in the data. For quarterly data, four seasonal dummies are created (IQ1, IQ2, IQ3, IQ4), each taking the value 1 in the corresponding quarter and 0 otherwise. A typical example of quarterly seasonal pattern are fourth quarter spending sprees related to holidays, or seasonal variations in tourism related to weather conditions and holidays in other countries.

- **Trend Indicator**: A linear trend variable (`ITREND`) is created to capture the overall upward or downward movement of the time series over time. For example, technological progress or population growth may cause a steady upward trend in economic time series such as income or consumption.

```
# combine indicators into single data frame
indicator_vars <- ts_c(
  iis_indicators %>% ts_long(),
  sis_indicators %>% ts_long(),
  season_indicators %>% ts_long(),
  trend_indicator %>% ts_long()
)
```

8.2.4 Data Combination

- The original data, log-transformed data, and indicator variables are combined into a single dataset using `ts_c()`.

8.2.5 Data Subsetting

- The dataset is filtered to include only the observations within the `mselect_start` and `mselect_end` range. The data is also subset based on the `yvar_name` and `xvar_list` to retain only the variables relevant for the specific model being considered.

8.2.6 Lag Generation

- Lags of the predictor variables are generated using `ts_lag()` and combined using `ts_c()`. This creates a dataset containing the current and lagged values (up to `max_lag`) of each predictor.

```
# generate lags of selected predictors
xvar_lags <- 0:max_lag %>%
  map(~ts_lag(xvar_0, .x)) %>%
  reduce(ts_c)
```

8.2.7 Data Transformation to xts

- The response variable and the combined predictor variables (including lags, trend, and seasonal dummies) are converted to `xts` objects for use with the `gets` package.

```
# transform to xts
yvar_xts <- yvar_0 %>%
  ts_xts()

# transform to xts
xvar_xts <- xvar_all %>%
  ts_xts()
```

8.3 Model selection steps (`gets`)

The core of the model selection process is handled by the gets package, which implements a general-to-specific (GETS) algorithm:

8.3.1 Formulate a General Unrestricted Model (GUM)

- The GETS process begins with a GUM that includes all potential regressors (predictors, lags, indicators). In our case this includes an autoregressive term, trend and seasonal dummies. The `arx()` function is used to estimate the GUM using the `yvar_xts` and `xvar_xts` data. The `mc = TRUE` argument specifies that a constant term should be included. The `ar = 1:max_lag` argument indicates that autoregressive terms up to the specified `max_lag` should be included.
- The `arx()` function is also capable of detecting variance anomalies in the error terms, such as autoregressive conditional heteroskedasticity (ARCH).

```
# formulate a general unrestricted model
# NA-s are automatically eliminated from the data
  gum_model <- arx(y = yvar_xts, mc = TRUE, ar = 1:max_lag, mxreg = xvar_xts)
```

8.3.2 Run the `gets` Algorithm

- First, the `isat()` function is applied to the GUM data to identify outliers in the unrestricted model using indicator variables. Outliers are defined as those observations whose residuals are deemed too large, possibly due to some extraordinary events. The outlier identification takes into account heteroskedasticity or autocorrelation in the residuals.

```
# identify outliers in the GUM ----
isat_model <- isat(y = gum_data[, 1],
  mc = TRUE,
  mxreg = gum_data[, -1],
  ar.LjungB = list(lag=max_lag,
  pval=0.01),
  arch.LjungB = list(lag=max_lag, pval=0.01),
  iis = TRUE,
  sis = TRUE,
  plot = TRUE
)
```

- Next, the `getsm()` function is applied to the outlier-augmented model to perform the general-to-specific model selection. It starts with the GUM and iteratively eliminates insignificant regressors based on their t-statistics and diagnostic tests for autocorrelation (Ljung-Box Q-test) and autoregressive conditional heteroskedasticity (ARCH LM test) in the error terms. The `t.pval` argument sets the significance level for retaining regressors. The `keep` argument can be used to force certain variables to be kept in the model regardless of their significance.

```
# run the gets (general to specific) model selection algorithm ----
gets_model <- getsm(isat_model,
  t.pval = 0.01,
  ar.LjungB = list(lag=max_lag, pval=0.01),
  arch.LjungB = list(lag=max_lag,
  pval=0.01),
  keep = c(1))
```

8.3.3 Identify Outliers in the Relationship

- After the initial GETS selection, the `isat()` function is applied to the residuals of the selected model to identify any remaining outliers. This is important, because the initial model selection process may produce a different residual series which now exhibits new outliers.

```
# check outliers in the residuals
isat_res <- gets_model %>%
  residuals() %>%
  isat()
```

8.3.4 Repeat `gets` Model Selection (Optional)

- If `second_pass` is set to `TRUE`, the model selection process is repeated, but this time including the newly identified outliers as regressors. This step further refines the model and ensures that the final specification is robust to outliers in the data.

8.3.5 Verify if Additional Outliers Arose Due to Greater Model Parsimony

- A final check is performed using `isat()` on the final model's residuals to ensure that no additional outliers have emerged due to the model simplification process. The `isatvar()` function can then be used to extract the estimated path of the constant, incorporating any significant shifts indicated by the outlier analysis.
- The `isattest()` function generates a plot showing the significant shifts in the constant over time.

8.3.6 Handle Zero-Valued Predictors

- If the estimation period (`est_end`) is shorter than the model selection period (`mselect_end`), some predictors (e.g., outlier dummies) might contain only zeros in the estimation sample. These variables are identified and removed from the dataset to avoid issues during estimation.

- This step recognizes that the outliers that occurred after `est_end` will not be relevant for the estimation of the model going forward.

```
# find all variables with only zeros in the estimation period (if shorter than model selectio
zero_vars <- gets_data %>%
  ts_span(end = est_end) %>%
  ts_tbl() %>%
  mutate(value = abs(value)) %>%
  group_by(id) %>%
  summarize(sum_vals = sum(value)) %>%
  filter(sum_vals == 0) %>%
  pull(id)

# remove zero valued variables from the estimation sample
est_data <- gets_data %>%
  ts_span(end = est_end) %>%
  ts_tbl() %>%
  ts_wide() %>%
  select(-all_of(zero_vars)) %>%
  ts_long() %>%
  ts_xts()
```

8.3.7 Re-estimate Final Model

- The final selected model is re-estimated using `lm()`.

```
# re-estimate re-specified model via lm using the estimation sample
  est_lm <- lm(as.formula(str_c(yvar_name, " ~ .")), data = est_data)
```

8.3.8 Save Model Equation

- The final model equation is saved as a text file (*model_eq.txt*) using the `model_equation()` function. This function formats the model output into an equation that can be read by

the bimets package. Since the equation can be later re-estimated in bimets, the saved equation does not contain the estimated coefficients to keep it general.

```
# look at estimated model and bimets model components
model_equation(est_lm)

# save equation
if (save_eq) {
  sink(here("output", "model_eq.txt"), append = TRUE)
  cat(str_glue("

    {model_equation(est_lm)[2:4]}

  "))
  sink()
}
```

8.4 Produce a quasi-forecast with the selected model (`bimets`)

After selecting the model using `gets`, the `bimets` package is used to estimate the model and produce a quasi-forecast:

8.4.1 Load Model and Data

- The `LOAD_MODEL()` function loads the model equation from the saved text file. The `LOAD_MODEL_DATA()` function then combines the model with the relevant data, ensuring that all variables used in the model are present in the dataset.

```
# load model from stored txt file
model_eq <- bimets::LOAD_MODEL(modelFile = here("output", "model_eq.txt"))

# store variables in bimets format (no ragged edge: drop_na)
hist_q_bimets <- hist_q_mod %>%
  filter(id %in% c(model_eq$vendog, model_eq$vexog)) %>%
  ts_wide() %>%
  drop_na() %>%
  ts_long() %>%
  ts_tslist() %>%
  map(bimets::as.bimets)
```

```
# add data to model
model_eq_dat <- bimets::LOAD_MODEL_DATA(
  model_eq,
  hist_q_bimets
)
```

8.4.2 Estimate the Model

- The `ESTIMATE()` function estimates the model using the specified estimation range (`est_range`). If the forecast end date (`fcst_end`) is later than the estimation end date, a Chow test for structural stability is automatically performed. The Chow test assesses whether the model's parameters are constant over the estimation and forecast periods. The `CHOWPAR` argument specifies the date at which the data is split for the Chow test.

```
# determine range of history for estimation
est_range <- model_eq_dat$modelData %>%
  set_attr_tslist() %>%
  ts_xts() %>%
  zoo::index() %>%
  extract(c(max_lag+1, length(.)))

# estimate model
if (floor_date(fcst_end, "quarter") <= floor_date(est_range[2], "quarter")) {
  model_est <- bimets::ESTIMATE(model_eq_dat,
    eqList = model_eq_dat$vendog,
    TSRANGE = c(year(est_range[1]), quarter(est_range[1]), year(est_range[2]), quarter(est_ra
    quietly = FALSE
  )
} else {
  model_est <- bimets::ESTIMATE(model_eq_dat,
    eqList = model_eq_dat$vendog,
    TSRANGE = c(year(est_range[1]), quarter(est_range[1]), year(est_range[2]), quarter(est_ra
    CHOWTEST = TRUE,
    CHOWPAR = c(year(fcst_end), quarter(fcst_end)),
    quietly = FALSE
  )
}
```

8.4.3 Simulate the Model

- Statistical models generate forecasts based on historical data patterns and estimated relationships between variables. These models often assume that the underlying structure and relationships observed in the past will continue to hold in the future. There might be events or changes that are known (or believed) to occur in the future but are not reflected in the historical data or the model's structure. Set *addfactors* to adjust the forecast path. An addfactor is a value that is added to the model's forecast for a specific variable and time period. It's essentially a way to incorporate external information, expert opinion, or subjective judgment into a model-based forecast. Addfactors are used to bridge the gap between purely model-based forecasts and external information or judgment. They are expressed in the same units as the variable being forcasted.

```
# set value of addfactors
scen_addfactor <- hist_q_mod %>%
  filter(id %in% str_glue("{model_eq_dat$vendog}")) %>%
  mutate(value = 0) %>%
  ts_tslist() %>%
  map(bimets::as.bimets)

# update an addfactor stored in xts format
add_qmod.xts$NDEA_HI %+=% addf(2021.4, 2024.4, .01, .025)
```

- The `SIMULATE()` function simulates the model over the forecast period (`fcst_start` to `fcst_end`). The `simType = "FORECAST"` argument specifies that a deterministic forecast should be produced.
- The `simConvergence` and `simIterLimit` arguments control the convergence criteria and maximum number of iterations for the simulation algorithm. These parameters are particularly relevant for models with simultaneous equations, where the values of endogenous variables depend on each other. The simulation algorithm iteratively solves the system of equations until the solution converges or the maximum number of iterations is reached.
- Use the `fcutils::set_tsrange()` function to deal with a ragged edge in the data and prepare the exogenization range for the `Exogenize` parameter of the `SIMULATE()` function.

```
# simulate model
model_sim <- bimets::SIMULATE(model_est,
  simType = "FORECAST",
  TSRANGE = c(year(fcst_start), quarter(fcst_start), year(fcst_end), quarter(fcst_end)),
  ConstantAdjustment = scen_addfactor,
  Exogenize = exog_range,
  simConvergence = 0.00001,
```

```
  simIterLimit = 100,
  quietly = FALSE
)
```

8.4.4 Evaluate the Simulation

- The simulated values (forecast) are extracted from the `model_sim` object and combined with the historical data for evaluation. The `plot_comp_2()` function is used to generate plots comparing the quasi-forecast with the actual history. These plots help assess the model's ability to capture the dynamics of the time series and its potential forecasting accuracy.

```
# extract forecast
model_fcst <- model_sim$simulation %>%
  extract(model_sim$vendog) %>%
  set_attr_tslist() %>%
  ts_tbl() %>%
  mutate(id = str_c(model_sim$vendog, "SOL"), .before = time)

# combine history and forecast for plot
plot_data_fcst <- ts_c(hist_q_mod %>% filter(id %in% model_sim$vendog), model_fcst) %>%
  ts_wide() %>%
  slice(which(!is.na(!!sym(model_sim$vendog)) | !is.na(!!sym(str_glue("{model_sim$vendog}SOL

plot_comp_2(plot_data_fcst %>% ts_long(),
  rng_start = as.character(Sys.Date() - years(15)),
  rng_end = fcst_end %>% as.character(),
  height = 200, width = 400
)
```

8.5 Stochastic simulation

The preceding workflow can be extended with stochastic simulation. If necessary it is preceded by the same data preparation and model selection steps.

8.5.1 Simulate model deterministically to obtain mean forecast.

The first step for obtaining a forecast interval is to simulate the estimated model deterministically. In a deterministic simulation, the model's equations are solved without any random

shocks. This produces a single "mean" forecast, representing the most likely path of the endogenous variables given the model's structure, estimated parameters, and any specified exogenous assumptions.

8.5.2 Extract forecast and combine it with history.

After the deterministic simulation, the forecasted values are extracted from the simulation object. The historical data and the forecasted values are combined to facilitate comparison and visualization.

8.5.3 Inspect the forecast via plots.

Then generate plots to visually compare the historical data and the deterministic forecast. These plots allow for a visual assessment of the forecast's behavior compared to the historical data. For example, one can observe whether the forecast captures the overall cycles in the historical data, and whether there are any abrupt changes or discontinuities at the forecast origin.

8.5.4 Set parameters for stochastic simulations.

Stochastic simulation introduces randomness into the model to generate a distribution of possible future paths, rather than just a single mean forecast. This helps to quantify the uncertainty surrounding the forecast.

The first step in stochastic simulation is to define the structure of the random shocks. This is done by creating a list called `myStochStructure`:

```
# set parameters for stochastic simulation
myStochStructure <- list()
for (ser_i in scen_model$vendog) {
  myStochStructure[[ser_i]] <- list(
    TSRANGE = TRUE,
    TYPE = "NORM",
    PARS = c(0, scen_model_est$behaviorals[[ser_i]][["statistics"]][["StandardErrorRegressio
  )
}
```

- The code iterates through each endogenous variable (`ser_i`).
- For each variable, it creates a list specifying the parameters of the random shocks:
  - **`TSRANGE = TRUE`**: Indicates that the shocks should be applied throughout the entire simulation period.

- `TYPE = "NORM"`: Specifies that the shocks should be drawn from a normal distribution.
- `PARS`: Defines the parameters of the normal distribution. In this case, the mean is set to 0 and the standard deviation is set to the standard error of the regression for the corresponding equation. This means that the random shocks are scaled to the estimated uncertainty of each equation. The standard deviation of the error term is obtained from the estimation output, and it reflects the historical volatility of the residuals.

In addition to perturbing the error terms, the script also allows for perturbing the estimated coefficients. This captures the uncertainty in the parameter estimates themselves.

```
# perturb coefficients (draw from multivariate normal distro)
set.seed(987)
scen_model_est_copy <- scen_model_est
scen_model_est_pars <- scen_model_est$behaviorals %>%
  map(~ if (n_coeff_sim > 0) {
    cbind(
      .x$coefficients,
      MASS::mvrnorm(n = n_coeff_sim, mu = .x$coefficients, Sigma = .x$statistics$CoeffCovaria
    )
  } else {
    .x$coefficients
  })
```

- `set.seed(987)`: Sets the random seed for reproducibility.
- `scen_model_est_copy <- scen_model_est`: Creates a copy of the estimated model object.
- `scen_model_est_pars`: Stores the perturbed coefficients.
- The code iterates through each equation in `scen_model_est$behaviorals`.
- If `n_coeff_sim` is greater than 0, it draws `n_coeff_sim` sets of coefficients from a multivariate normal distribution using `MASS::mvrnorm()`. The mean of the distribution is the original estimated coefficients (`.x$coefficients`), and the covariance matrix is the estimated covariance matrix of the coefficients (`.x$statistics$CoeffCovariance`). This ensures that the perturbed coefficients are consistent with the estimated uncertainty and correlations among the parameters.
- If `n_coeff_sim` is 0 (coefficients are not drawn randomly), the original coefficients are used without perturbation.

8.5.5 Run stochastic simulation.

The stochastic simulation is performed using the `STOCHSIMULATE()` function in `bimets`:

```
# create variables to hold stochastic simulation objects and forecasts
scen_model_stochsim <- list()
scen_model_stochfcast <- list()

# loop over the parameter draws (first one is the estimate)
for (sim_i in 1:(n_coeff_sim + 1)) { # sim_i = 1

  # set params for each equation
  for (eq_i in scen_model_dat$vendog) { # eq_i = scen_model_dat$vendog[1]
    scen_model_est_copy$behaviorals[[eq_i]][["coefficients"]][, 1] <- scen_model_est_pars[[eq
  }

  # simulate model
  scen_model_stochsim[[sim_i]] <- bimets::STOCHSIMULATE(scen_model_est_copy,
    simType = "FORECAST",
    TSRANGE = c(year(fcst_start), quarter(fcst_start), year(fcst_end), quarter(fcst_end)),
    Exogenize = exog_range,
    simConvergence = 0.00001,
    simIterLimit = 100,
    StochStructure = myStochStructure,
    StochReplica = n_stoch_sim,
    StochSeed = 123 + sim_i,
    quietly = FALSE
  )

  # extract forecast
  scen_model_stochfcast[[sim_i]] <- scen_model_stochsim[[sim_i]][["simulation_MM"]] %>%
    extract(scen_model$vendog)
}
```

- The code loops `n_coeff_sim + 1` times. In the first iteration (`sim_i = 1`), the original estimated coefficients are used. In subsequent iterations, the perturbed coefficients are used.
- Inside the loop, for each iteration:
  - The coefficients in `scen_model_est_copy` are updated with the current set of perturbed coefficients.
  - `STOCHSIMULATE()` is called to perform the stochastic simulation.
  - **`StochStructure = myStochStructure`**: Specifies the structure of the random shocks defined earlier.
  - **`StochReplica = n_stoch_sim`**: Sets the number of stochastic replications for each set of coefficients. Each replication involves drawing a new set of random shocks and simulating the model.

- `StochSeed = 123 + sim_i`: Sets the random seed for each iteration, ensuring reproducibility.
- The simulated paths are extracted from `scen_model_stochsim[[sim_i]][["simulation_MM"]]` and stored in `scen_model_stochfcast[[sim_i]]`.

8.5.6 Extract simulated paths and obtain deviations from the mean forecast.

After the stochastic simulation, the simulated paths are processed to analyze the distribution of possible outcomes:

```
# convert matrices in the list to ts_long format
scen_model_stochfcast <- scen_model_stochfcast %>%
  map_depth(2, function(x) {
    as_tibble(x, .name_repair = NULL) %>%
      mutate(time = seq.Date(from = fcst_start, to = fcst_end, by = "quarter"), .before = 1)
      ts_long()
  })

# forecast deviations
scen_model_stochdev <- scen_model_stochfcast %>%
  map_depth(2, function(x) {
    ts_wide(x) %>%
      mutate(across(V1:last_col(), ~ (.x / V1) - 1)) %>%
      select(-V1) %>%
      ts_long()
  })
```

- `scen_model_stochfcast` is transformed using `map_depth(2, ...)` to apply a function to each simulated path (each element at depth 2 of the nested list).
  - `as_tibble()` converts the matrix of simulated values to a tibble.
  - `mutate(time = ...)` adds a time index.
  - `ts_long()` converts the data to long format.
- `scen_model_stochdev` calculates the deviations of each simulated path from the mean forecast (the first path, `V1`).
  - `ts_wide()` converts the data to wide format.
  - `mutate(across(V1:last_col(), ~ (.x / V1) - 1))` calculates the percentage deviation of each column (simulated path) from the first column (mean forecast).
  - `select(-V1)` removes the mean forecast column.
  - `ts_long()` converts the data back to long format.

The script then calculates quantiles of the distribution of deviations:

```
# quantiles
scen_model_quantiles <- scen_model_stochdev %>%
  transpose() %>%
  map_depth(2, ts_wide) %>%
  map(~ reduce(.x, function(x1, x2) full_join(x1, x2, by = "time"))) %>%
  map(function(x) {
    # ts_wide(x) %>%
    rowwise(x) %>%
      transmute(
        time = time,
        Q05 = quantile(c_across(-1), 0.05),
        Q10 = quantile(c_across(-1), 0.10),
        Q20 = quantile(c_across(-1), 0.20),
        Q50 = quantile(c_across(-1), 0.50),
        Q80 = quantile(c_across(-1), 0.80),
        Q90 = quantile(c_across(-1), 0.90),
        Q95 = quantile(c_across(-1), 0.95),
      ) %>%
      ungroup()
  })
```

- `transpose()` transposes the nested list structure of `scen_model_stochdev`.
- `map_depth(2, ts_wide)` converts each set of deviations to wide format.
- `map(~ reduce(.x, ...))` combines the deviations for each variable into a single tibble.
- `map(function(x) ...)` calculates the quantiles for each time period using `quantile()`.

The resulting `scen_model_quantiles` object contains the 5th, 10th, 20th, 50th, 80th, 90th, and 95th percentiles of the distribution of deviations for each variable.

8.5.7 Inspect the paths via plots.

Finally, the script generates plots to visualize the simulated paths and the quantiles:

```
# generate plots (compare levels)
pdf(here("output/plots", "stoch_plot.pdf"))

for (ser_i in scen_model$vendog) { # ser_i = scen_model$vendog[1]
  plot_out <- scen_model_stochfcast %>%
    transpose() %>%
    map_depth(2, ts_wide) %>%
    map(~ reduce(.x, function(x1, x2) full_join(x1, x2, by = "time"))) %>%
    map(~ rename_with(.x, ~ c("time", str_c("V", 1:((n_coeff_sim + 1) * (n_stoch_sim + 1)))))
```

```
    map(ts_long) %>%
    extract2(ser_i) %>%
    ggplot(aes(x = time)) +
    geom_line(aes(y = value, group = id), linetype = 1, alpha = 0.8, size = 0.8, color = "sea
    labs(x = NULL, y = str_glue("{ser_i}")) +
    theme_minimal() +
    theme(legend.position = "bottom")
  print(plot_out)
}

for (ser_i in scen_model$vendog) { # ser_i = scen_model$vendog[1]
  plot_out <- scen_model_stochdev %>%
    transpose() %>%
    map_depth(2, ts_wide) %>%
    map(~ reduce(.x, function(x1, x2) full_join(x1, x2, by = "time"))) %>%
    map(~ rename_with(.x, ~ c("time", str_c("V", 1:((n_coeff_sim + 1) * (n_stoch_sim)))))) %>
    map(ts_long) %>%
    extract2(ser_i) %>%
    ggplot(aes(x = time)) +
    geom_line(aes(y = value, group = id), linetype = 1, alpha = 0.8, size = 0.8, color = "sea
    labs(x = NULL, y = str_glue("{ser_i}")) +
    theme_minimal() +
    theme(legend.position = "bottom")
  print(plot_out)
}

for (ser_i in scen_model$vendog) { # ser_i = scen_model$vendog[1]
  plot_out <- scen_model_quantiles %>%
    extract2(ser_i) %>%
    ts_long() %>%
    ggplot(aes(x = time)) +
    geom_line(aes(y = value, group = id), linetype = 1, alpha = 0.8, size = 0.8, color = "sea
    labs(x = NULL, y = str_glue("{ser_i}")) +
    theme_minimal() +
    theme(legend.position = "bottom")
  print(plot_out)
}

dev.off()
```

- The code iterates through each endogenous variable (`ser_i`).
- For each variable, it creates three types of plots:

  - **Simulated levels**: Shows all simulated paths of the variable.
  - **Deviations from the mean forecast**: Shows the deviations of each simulated path from the mean forecast.
  - **Quantiles**: Shows the quantiles of the distribution of deviations.

- `ggplot2` is used to create the plots, with `geom_line()` used to display the paths.
- The plots are saved to a PDF file (`stoch_plot.pdf`).

These plots provide a visual representation of the uncertainty surrounding the forecast. The spread of the simulated paths and the width of the quantile bands indicate the range of possible outcomes. By examining these plots, one can assess the degree of uncertainty associated with different variables and time periods.

The script includes an additional set of plots based on smoothed quantile paths, which are generated by applying user-specified adjustment factors to certain quantiles at specific time points and then interpolating the values in between. This allows for incorporating subjective views about the evolution of uncertainty over time. However, the overall logic of plotting the smoothed paths is similar to that described above.

8.6 Conclusion

This detailed process, combining the strengths of `gets` and `bimets`, provides a robust and flexible framework for model selection and evaluation in time series forecasting. The use of indicator variables, diagnostic tests, and quasi-forecast evaluation further enhances the reliability and interpretability of the selected models. The use of stochastic simulation provides valuable insights into the range of possible outcomes and helps to quantify the risks associated with relying solely on a single mean forecast.