{workr}: a very simple R data pipeline

Phuse Connect 2026

Zelos Zhu + Jeremy Wildfire

2026-03-26

Agenda

1. What is {workr}?
2. Why {workr}?
3. {workr} ❤️ {pharmaverse}
4. {workr} Apps

What is {workr}?

A **very** simple R data pipeline

{workr} workflows are YAML files

easily parsed to an R `list`

```yaml
# hello_cars.yaml
meta:
  ID: hello_cars
  col: speed
steps:
  - name: dplyr::pull
    output: speed
    params:
      df: df
      col: col
  - name: mean
    output: result
    params:
      lData: speed
```

`meta` captures metadata

```yaml
# hello_cars.yaml
meta:
  ID: hello_cars
  col: speed
steps:
  - name: dplyr::pull
    output: speed
    params:
      df: df
      col: col
  - name: mean
    output: result
    params:
      lData: speed
```

`steps` are function calls

```yaml
# hello_cars.yaml
meta:
  ID: hello_cars
  col: speed
steps:
  - name: dplyr::pull
    output: speed
    params:
      df: df
      col: col
  - name: mean
    output: result
    params:
      lData: speed
```

`meta` can be used in `steps` via `params`

```yaml
# hello_cars.yaml
meta:
  ID: hello_cars
  col: speed
steps:
  - name: dplyr::pull
    output: speed
    params:
      df: df
      col: col
  - name: mean
    output: result
    params:
      lData: speed
```

`workr::RunWorkflow()`

Inputs

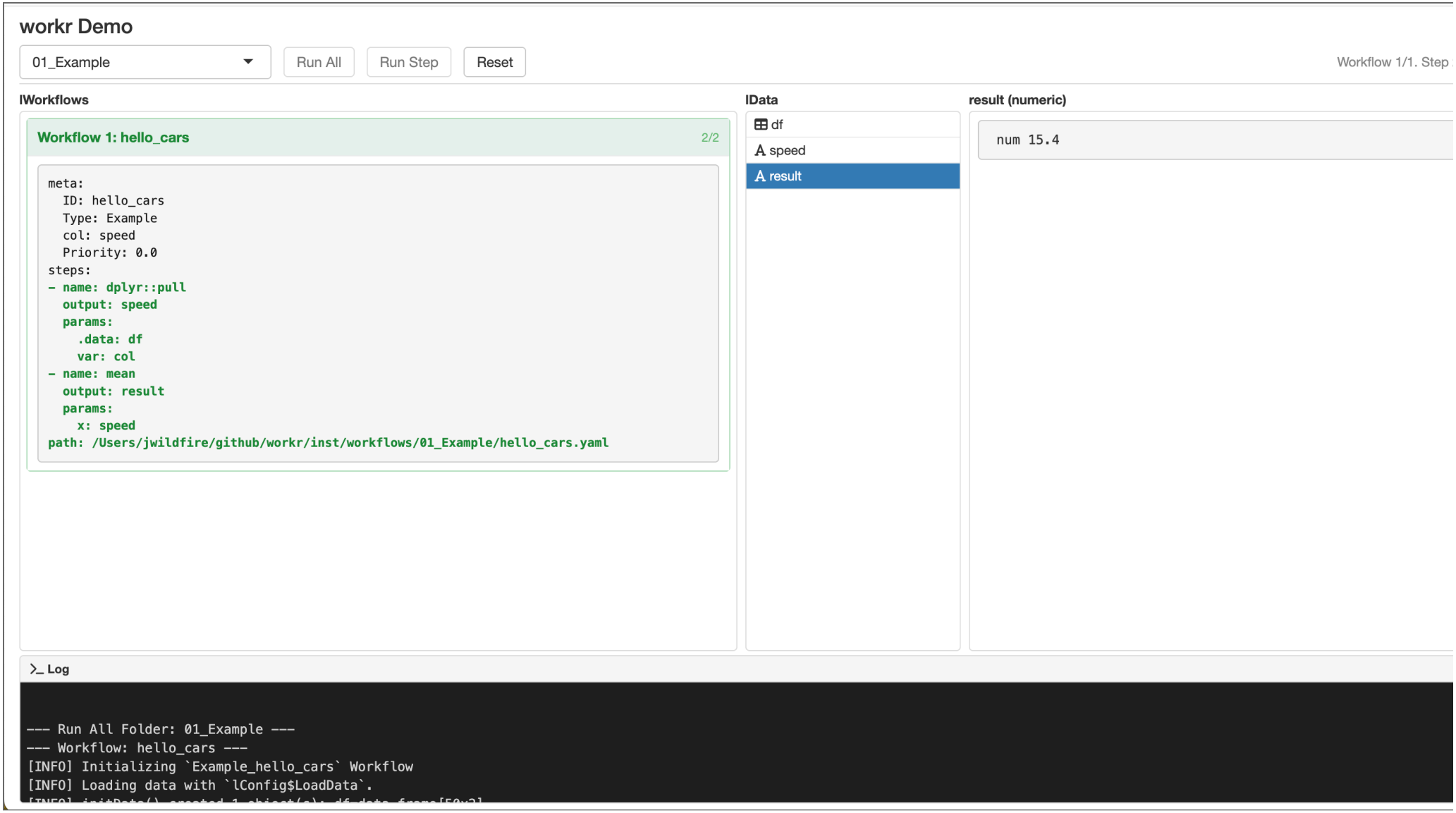
- workflow (YAML file parsed to a `list`)
- initial data (`lData`)
- available to all `steps` and can be updated by each step

Output

- Returns result of last step by default
- If `bReturnResult = FALSE` returns full workflow object with all intermediate outputs

`workr::demoApp()`

Example 1: Hello Cars

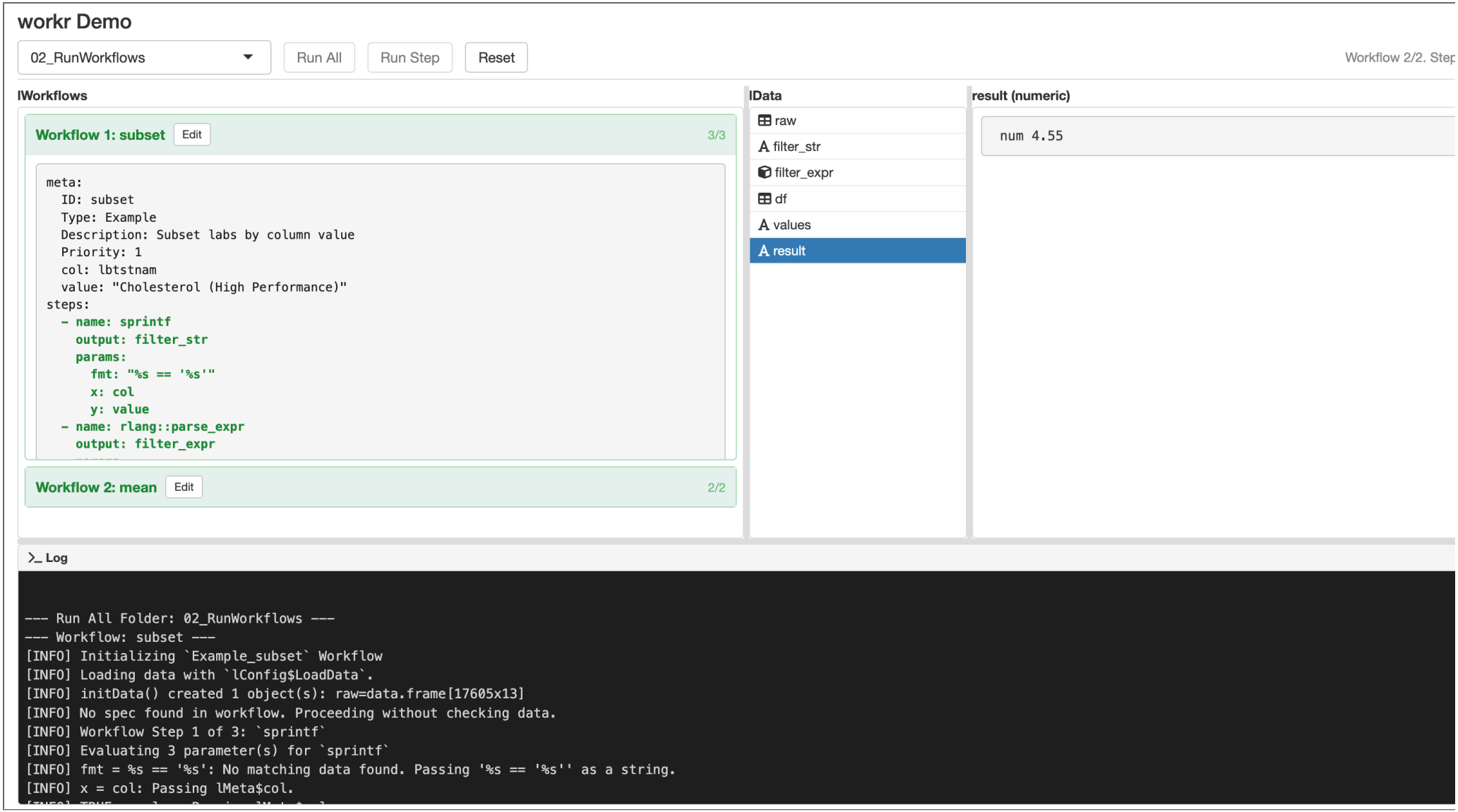

workr Demo
01_Example
Run All
Run Step
Reset
Workflow 1/1. Step
lWorkflows
lData
result (numeric)
Workflow 1: hello_cars
2/2
meta:
ID: hello_cars
Type: Example
col: speed
Priority: 0.0
steps:
- name: dplyr::pull
output: speed
params:
.data: df
var: col
- name: mean
output: result
params:
x: speed
path: /Users/jwildfire/github/workr/inst/workflows/01_Example/hello_cars.yaml
df
speed
result
num 15.4
>_ Log
--- Run All Folder: 01_Example ---
--- Workflow: hello_cars ---
[INFO] Initializing `Example_hello_cars` Workflow
[INFO] Loading data with `lConfig$LoadData`.

`workr::RunWorkflows()`

- Convenience function to run multiple workflows in sequence
- Output of each workflow added to `lData` for the next workflow

`workr::demoApp()`

Example 2: Chaining Workflows


workr Demo
02_RunWorkflows
Run All
Run Step
Reset
Workflow 2/2. Step
lWorkflows
Workflow 1: subset
Edit
3/3
meta:
ID: subset
Type: Example
Description: Subset labs by column value
Priority: 1
col: lbtstnam
value: "Cholesterol (High Performance)"
steps:
- name: sprintf
output: filter_str
params:
fmt: "%s == '%s'"
x: col
y: value
- name: rlang::parse_expr
output: filter_expr
Workflow 2: mean
Edit
2/2
lData
raw
filter_str
filter_expr
df
values
result
result (numeric)
num 4.55
Log
--- Run All Folder: 02_RunWorkflows ---
--- Workflow: subset ---
[INFO] Initializing `Example_subset` Workflow
[INFO] Loading data with `lConfig$LoadData`.
[INFO] initData() created 1 object(s): raw=data.frame[17605x13]
[INFO] No spec found in workflow. Proceeding without checking data.
[INFO] Workflow Step 1 of 3: `sprintf`
[INFO] Evaluating 3 parameter(s) for `sprintf`
[INFO] fmt = %s == '%s': No matching data found. Passing '%s == '%s'' as a string.
[INFO] x = col: Passing lMeta$col.

That’s really about it

- Minimal mental model
- Easy to read / debug
- Surprisingly scalable

Why {workr}?

Scalable Clinical Trial Operations

From Safety Monitoring …

{safetyGraphics}

With gnarly data pipelines …

To Risk-Based Quality Monitoring

- `{gsm}` (Good Statistical Monitoring) is a qualified set of R packages that provides a GxP framework for central monitoring in clinical trials.
- Designed to support a variety of business processes, from manually executed RBQM reporting to enterprise-wide monitoring platforms.

Repeatability and scalability are key

- Data pipeline: 30 studies × monthly snapshots × 15 metrics × 5 steps
- ~27,000 metrics / year

But Science is Hard

Clinical trials are complex

- Study designs vary
- Metrics need tweaks
- Study-level (and sometimes metric-level) customization required

🚫 Custom Study Scripts 🚫

- Needed reusable pipelines / workflows
- Existing tools felt a bit complicated (`targets`, `glue`, etc.)
- So we created `gsm::RunWorkflow()` in April 2022
- Migrated to `{workr}` for better modularity and extensibility

{gsm} metrics

{gsm} provides a framework that allows users to assess and visualize site-level risk in clinical trial data.
12 Core Site KRIs

1. Adverse Event Reporting Rate
2. Serious Adverse Event Reporting Rate
3. Non-important Protocol Deviation Rate
4. Important Protocol Deviation Rate
5. Grade 3+ Lab Abnormality Rate
6. Study Discontinuation Rate
7. Treatment Discontinuation Rate
8. Query Rate
9. Outstanding Query Rate
0. Outstanding Data Entry Rate
1. Data Change Rate

2. Screen Failure Rate

Highly Standardized Metric Workflows

Analysis Workflow Vignette

`workr::demoApp()`

Example 3: AE KRI Workflow

{gsm} extensions

Reporting and Visualization Layer

- Site KRI Report
- Country KRI Report
- Cross-Study KRI Report
- Eligibility Report
- QTL Report
- {gsm} Deep Dive App

Site KRI Report

Open report

Country KRI Report

Open report

Cross-Study KRI Report

Open report

Eligibility Report

Open report

QTL Report

Open report

{gsm} Deep Dive App

Open app

Full {gsm} Data Model

{gsm} ❤️ {workr}

workr explorer

{gsm} + {workr} Study Operations

- `workr::snapshot()` combines workflows across packages
- Quarterly snapshots of {gsm} workflows
- `rv` for managing package versions
- GitHub repo for each study
- Clone default workflows, customize as needed
- `workr::runProject()` runs folders of workflows
- Schedule monthly and/or trigger as needed
- Load/save hooks for preferred locations
  - folder, database, S3, etc.

GxP Considerations

- Extensive qualification via **{qcthat}**
- {workr}-driven Action Log
- SMEs assess and address detected risks
- In use on 20+ studies.

San Antonio | 1:30-2:30 PM

{workr} ❤️ {pharmaverse}

Not just for ClinOps!

From Raw to SDTM

- {sdtm.oak} is a popular R package that modularizes SDTM programming that is EDC/data-standards agnostic
- The algorithms and sub-algorithms provided can be reused across multiple SDTM domains
- We aim to replicate the results of this **vignette**

using workflows!

SDTM Example: RAW to VS

What you would run in R

```
wf <- yaml::read_yaml(system.file("demo_gsmpharmaverse/workflows/1_RAW_TO_SDTM/VS.yaml"))
lData <- list(
  dm_raw = read.csv(system.file("raw_data/dm.csv", package = "sdtm.oak")),
  vs_raw =  read.csv(system.file("raw_data/vitals_raw_data.csv", package = "sdtm.oak")),
  study_ct = read.csv(system.file("raw_data/sdtm_ct.csv", package = "sdtm.oak"))
)
RunWorkflow(
```

What's happening inside

```
meta:
  ID: VS
  Type: SDTM
  Description: Transform Raw VS to SDTM VS following sc
  Priority: 1
spec:
  # Read in data
  vs_raw:
```

=

```
lData$vs_raw2 <- generate_oak_id_vars(
  raw_dat = lData$vs_raw,
  pat_var = "PATNUM",
  raw_src = "vitals"
)
```

SDTM Example: Naming convention considerations

```
# lData after Step 1
list(
    vs_raw = {THE RAW DATASET}
    vs_raw2 = {THE RAW DATASET after we ran generate_oak_id_vars}
)
```

It is important to define how interim objects are handled within `lData`. As in pipe-based workflows (`%>%` and `|>`), teams can either preserve each step by assigning a new output name or overwrite the existing object; in this example, that means choosing vs_raw2 for traceability or overwriting `vs_raw`.

SDTM Example: CT assignment Steps

Using workflows

```
# Map topic variable SYSBP and its qualifiers.
- output: vs_sysbp
  name: sdtm.oak::hardcode_ct
  params:
    raw_dat: vs_raw
    raw_var: "SYS_BP"
    tgt_var: "VSTESTCD"
    tgt_val: "SYSBP"
    ct_spec: study_ct
    ct_clst: "C66741"
- output: vs sysbp
```

Using pipes

```
# Map topic variable SYSBP and its qualifiers.
vs_sysbp <-
  hardcode_ct(
    raw_dat = vs_raw,
    raw_var = "SYS_BP",
    tgt_var = "VSTESTCD",
    tgt_val = "SYSBP",
    ct_spec = study_ct,
    ct_clst = "C66741"
  ) %>%
  dplyr::filter(!is.na(.data$VSTESTCD))  %>%
```

Each pipe statement is equivalent to a step in the workflow; integration of any package would mostly be depend on familiarity with a particular R Package, not necessarily this workr framework.

From SDTM to ADaM

- {admiral} is a popular R package that modularizes ADaM programming with many extension packages that address specific therapeutic area needs
- Here we'll highlight just a few specific functions, but the documentation and user guides for {admiral} are some of the best amongst all R packages

ADaM Example: VS to ADVS (add MAP)

What you would run in R

```r
wf <- yaml::read_yaml(system.file("demo_gsmpharmaverse/workflows/2_SDTM_TO_ADAM/ADVS.yaml"))
sdtm <- list(
  SDTM_DM = arrow::read_parquet(system.file("demo_gsmpharmaverse/data/SDTM/SDTM_DM.parquet", package = "workr")),
  SDTM_VS = arrow::read_parquet(system.file("demo_gsmpharmaverse/data/SDTM/SDTM_VS.parquet", package = "workr"))
)
RunWorkflow(lWorkflow = wf, lData = sdtm)
```

What’s happening inside

```yaml
meta:
  ID: ADVS
  Type: ADAM
  Description: Create Basic ADVS
  Priority: 1
spec:
  SDTM_DM:
     all:
```

=

```r
advs <- admiral::derive_vars_merged(
    dataset = SDTM_VS,
    dataset_add = SDTM_DM,
    new_vars =  exprs(TRT01A),
    by_vars =  exprs(STUDYID, USUBJID)
  ) %>%
  mutate(
    PARAMCD = VSTESTCD,
```

ADaM Example: Create MAP

- The prior examples establish a consistent workflow pattern for producing an ADaM dataset. In this step, `derive_param_map()` derives mean arterial pressure (MAP) from `SYSBP` and `DIABP`.
- Compared with stitching together multiple tidyverse transformations, this approach reduces boilerplate and makes the derivation intent clearer.
- A valuable area for future collaboration, particularly in admiral, is harmonizing quasiquotation patterns to improve readability and adoption, especially around `expr()`, `exprs()`, and `!!`.

From ADaM to TFL

- How every team/organization handles data visualizations may just have the most variance
- We will go over a way that uses a workflow to render Rmarkdown documents with prespecified templates for tables/figures, specifically using `gtsummary` and `safetyCharts`
- The setup for this would be applicable for shiny apps, static reports, web-based html reports, where to host or view the final object is left to the user

TFL Example: ADVS to TFL

What you would run in R

```
wf <- yaml::read_yaml(system.file("demo_gsmpharmaverse/workflows/3_ADAM_TO_TFL/WorkProduct1.yaml", package = "workr"), wa
adam <- list(
  ADVS = arrow::read_parquet(system.file("demo_gsmpharmaverse/data/ADAM/ADAM_ADVS.parquet", package = "workr"))
)
workr::RunWorkflows(lWorkflows = wf, lData = adam )
```

What's happening inside

```
meta:
  ID: WorkProduct1
  Type: TFL
  Description: Create Basic Work Product/Report which c
  Priority: 1
spec:
  ADVS:
     all:
```

=

```
lParams <- list(dfADVS = arrow::read_parquet(system.fil
table1 <- rmarkdown::render(
  input = here::here("demo_gsmpharmaverse", "report_tem
  output_file = here::here("demo_gsmpharmaverse", "TFLS
  envir = new.env(parent = globalenv()),
  params = lParams
)
```

TFL Preview

TFL Example: Render Rmarkdown (2)

- These rendered documents (html in this case) can be mounted into a preferred viewing environment (r shiny apps, websites, pdf over email, etc.) for whoever the end user may be.
- The methodology & technical infrastructure will be left to the user.

Parent Rmd

Child Rmd

TFL Overview

- Using smaller child .Rmds that may match a respective company standard table/figure template may be favorable in this scenario
- This allows a larger work product/bundle to be stitched together with these child rmarkdowns to reduce clutter of the main document.
- For some deliveries it may be necessary to only have a few tables and figures, and other deliveries a much heftier report. This framework allows a “lego set” style assembled work deliverable in a pick and choose fashion

From ADaM to ARS/ARD

- {cards} is an R package for creating CDISC Analysis Results Data (ARD)
- It is designed to support automation, reproducibility, reusability, and traceability of analysis results

ARS/ARD Example: ADVS summary

What you would run in R

```
wf <- yaml::read_yaml(system.file("demo_gsmpharmaverse/workflows/3_ADAM_TO_ARS/table_mean_arterial_pressure.yaml"))
adam <- list(
  ADVS = arrow::read_parquet(system.file("demo_gsmpharmaverse/data/ADAM/ADAM_ADVS.parquet", package = "workr"))
)
workr::RunWorkflows(lWorkflows = wf, lData = adam )
```

What's happening inside

```
meta:
  ID: table_mean_arterial_pressure
  Type: ars
  Description: Create table 1 ARS
  Priority: 1
spec:
  ADVS:
    all:
```

=

```
table_predose_visit1_map  <- adam$ADVS %>%
  filter(PARAMCD == 'MAP', VISIT == 'VISIT1', VSTPT ==
  cards::ard_summary(variables = c(AVAL))
```

Final Considerations

- The primary outputs (thus far) of these workflows is typically a derived dataset, but persistence (load/save) is intentionally decoupled
- Workflows orchestrate transformation logic; storage strategy is flexible and left to the user/organization
- Saving outputs (e.g., .csv, .parquet, .json, or a data lake) can be implemented as an additional workflow step/configuration

{workr} Apps

Color demo

Live Demo

Enterprise {workr} Platform

gismo

Enterprise {workr} Platform

gismo

Enterprise {workr} Platform

gismo

Enterprise {workr} Platform

gismo

Enterprise {workr} Platform

gismo

GitHub {workr} Platform

open.gismo

open.gismo demo

GitHub {workr} Platform

open.gismo

open.gismo demo

{workr} ❤️ 🤖

- Designed for use with Agentic AI
- Breaks down complex tasks into modular steps
- Agents can generate and execute workflows
- {gsm} and {workr} skills coming soon!

{workr} 🙏 you!

- Simple R workflows
- Highly scalable and flexible
- Designed for GxP use
- Open source and extensible

gilead-biostats.github.io/workr/

Speaker notes