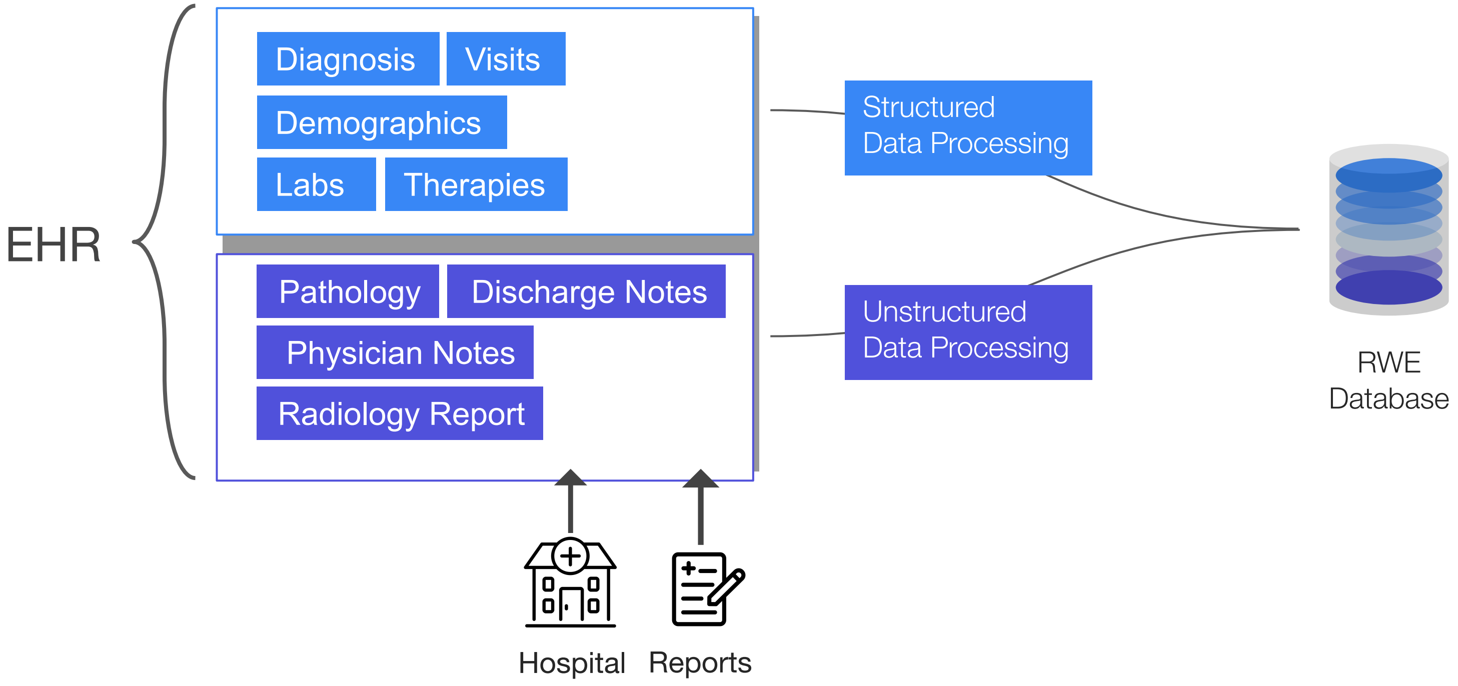
Simplified Data Quality Monitoring of Dynamic Longitudinal Data: A Functional Programming Approach

Jacqueline Gutman
@dynamicdataduo

rstudio::conf
SAN FRANCISCO // JANUARY 27 - 30, 2020

from RStudio

We process dynamic EHR data



National, longitudinal real-world datasets refreshed monthly, with 30-day recency

- Advanced Head & Neck Cancer
- Advanced Gastric Cancer
- Advanced Melanoma
- Advanced Non-Small Cell Lung Cancer
- Advanced Urothelial Carcinoma
- Chronic Lymphocytic Leukemia
- Acute Myeloid Leukemia
- Diffuse Large B-Cell Lymphoma
- Early Breast Cancer
- Hepatocellular Carcinoma
- Mesothelioma
- Metastatic Breast Cancer
- Metastatic Colorectal Cancer
- Metastatic Pancreatic Cancer
- Metastatic Prostate Cancer
- Metastatic Renal Cell Carcinoma
- Multiple Myeloma
- Ovarian Cancer
- Small Cell Lung Cancer
- Follicular Lymphoma
- Endometrial Cancer

How we approach our problem

OBJECTIVES

❏ Codify statistical and domain knowledge in a library of composable, modular functions

❏ Decouple the high-level checks and reasoning from the low-level implementation details

❏ Extend reasoning about a single dataset to apply across an expanding number of datasets

BENEFITS

★ Reproducible and documented best practices are easily shared

★ Abstraction reduces the cognitive complexity of an analysis and makes reasoning more transparent

★ Easier to scale analysis not just within a dataset, but as the number of datasets and elements increase

Core principles of functional programming

- ➔ Functions!
- ➔ that can be composed into higher-order functions *(compositionality)*
- ➔ which abstract out what is being done *(declarative)* from how it will be carried out *(imperative)*
- ➔ and which avoid side-effects and external state dependencies *(immutability)*

Building blocks of our approach to data quality

NESTED DATA STRUCTURES

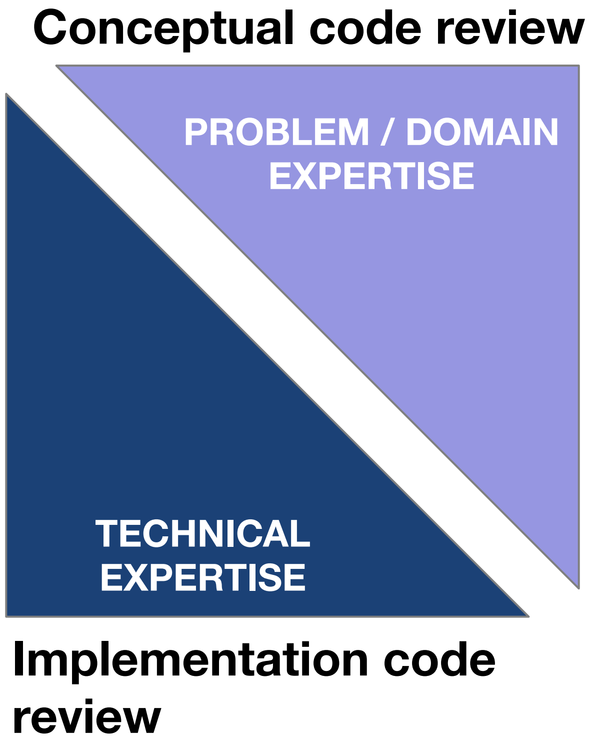
Tibbles to stash your tibbles!

PACKAGE DEVELOPMENT

Promote visibility, unit testing and documentation for your functions

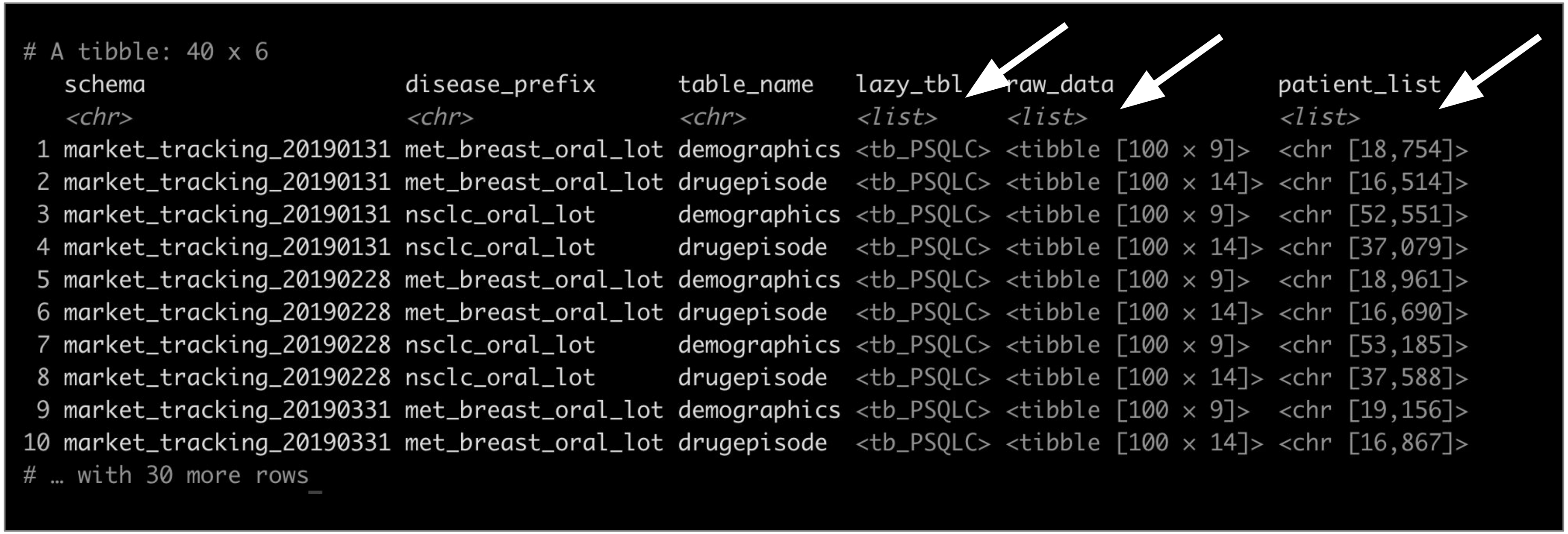
DIFFERENTIATED CODE REVIEW

For technical and non-technical project team members



Store tibbles, lists, and in-database lazy tbls as columns in a nested tibble

Data + metadata + evaluation metrics + summary statistics = FRIENDS FOREVER



```
# A tibble: 40 x 6
   schema                     disease_prefix      table_name   lazy_tbl    raw_data                  patient_list
   <chr>                      <chr>               <chr>        <list>      <list>                    <list>
 1 market_tracking_20190131 met_breast_oral_lot demographics <tb_PSQLC> <tibble [100 × 9]>  <chr [18,754]>
 2 market_tracking_20190131 met_breast_oral_lot drugepisode  <tb_PSQLC> <tibble [100 × 14]> <chr [16,514]>
 3 market_tracking_20190131 nsclc_oral_lot      demographics <tb_PSQLC> <tibble [100 × 9]>  <chr [52,551]>
 4 market_tracking_20190131 nsclc_oral_lot      drugepisode  <tb_PSQLC> <tibble [100 × 14]> <chr [37,079]>
 5 market_tracking_20190228 met_breast_oral_lot demographics <tb_PSQLC> <tibble [100 × 9]>  <chr [18,961]>
 6 market_tracking_20190228 met_breast_oral_lot drugepisode  <tb_PSQLC> <tibble [100 × 14]> <chr [16,690]>
 7 market_tracking_20190228 nsclc_oral_lot      demographics <tb_PSQLC> <tibble [100 × 9]>  <chr [53,185]>
 8 market_tracking_20190228 nsclc_oral_lot      drugepisode  <tb_PSQLC> <tibble [100 × 14]> <chr [37,588]>
 9 market_tracking_20190331 met_breast_oral_lot demographics <tb_PSQLC> <tibble [100 × 9]>  <chr [19,156]>
10 market_tracking_20190331 met_breast_oral_lot drugepisode  <tb_PSQLC> <tibble [100 × 14]> <chr [16,867]>
# … with 30 more rows
```

Nested data structures for comparing and evaluating multiple datasets

```r
count_records <- purrr::partial(execute_query,
                              select = "COUNT(*)")

count_patients <- purrr::partial(execute_query,
                                   select = "COUNT(distinct patientid)")

fetch_patients = purrr::partial(execute_query,
                                  select = "DISTINCT patientid",
                                  limit = 100)

data %<>%
    mutate(n_records = purrr::map(lazy_tbl, count_records, con = conn),
           n_patients = purrr::map(lazy_tbl, count_patients, con = conn),
           cohort = purrr::map(lazy_tbl, fetch_patients, con = conn))
```

What does all this get you?

- ★ Readability
- ★ Compositionality
- ★ Reproducibility
- ★ Robustness
- ★ Efficiency

flatiron

Readability

Abstraction and Declarative intent

Analysis code can be **understood** and **sanity-checked** by non-technical staff

Clarity of intent: abstract declarative code is self-documenting

Docstrings and unit tests to align implementation with intention

```
compare_snapshots(
    january_data,
    february_data,
    fn = compute_time_to_dx)

find_added_patients(
   old_data =
    january_data$cohort,
   new_data =
    february_data$cohort)
```

Compositionality

Reasoning with higher level functions

```
### Function definitions

process_drug_episodes <- . %>%
  filter_persistent_patients %>%
  compute_change_from_prev(
    var = “num_drug_episodes”,
    prev_var = “prev_drug_episodes”) %>%
  summarize_change_distribution(
    change_var = “change_num_drug_episodes”)
```

```
### Analysis code

data_processed <-
    process_drug_episodes(data)
```

Compositionality

Reasoning with higher level functions

```r
lm(Sepal.Length ~ Species,
   data = iris) %>%
broom::tidy(
   conf.int = TRUE) %>%
filter(p.value < 0.05) %>%
arrange(desc(statistic))
```

```r
tidy_lm <- purrr::compose(
    lm,
    ~ broom::tidy(.x,
        conf.int = TRUE),
    ~ filter(.x,
        p.value < 0.05),
    ~ arrange(.x,
        desc(statistic)),
    .dir = “forward”)

tidy_lm(Sepal.Length ~ Species,
    data = iris)
```

Example adapted from Colin Fay’s blog

Compositionality

Reasoning with higher level functions

```
### Analysis code
cohort %<>%
   left_join(demographics) %>%
   left_join(mortality) %>%
   left_join(drugepisode) %>%
   left_join(progression)
```

```
### Function definitions
left_join_all <- function(...) {
   purrr::reduce(list(...),
    .f = dplyr::left_join)
 }

### Analysis code
cohort %<>%
    left_join_all(demographics,
                  mortality,
                  drugepisode,
                  progression)
```

Reproducibility
Avoiding side effects and external states

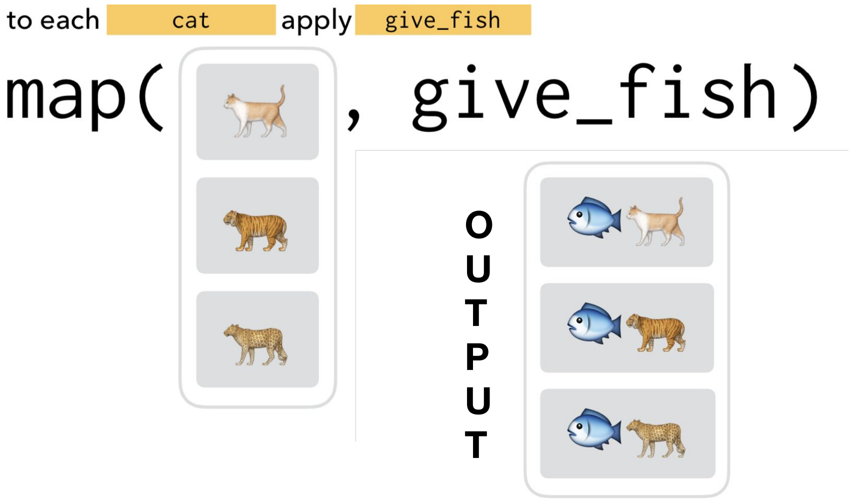
- Running the same function with the same input should always produce the same result
- Safeguard reproducibility with
  - Immutable inputs + outputs
  - Idempotent processes
  - Deterministic algorithms

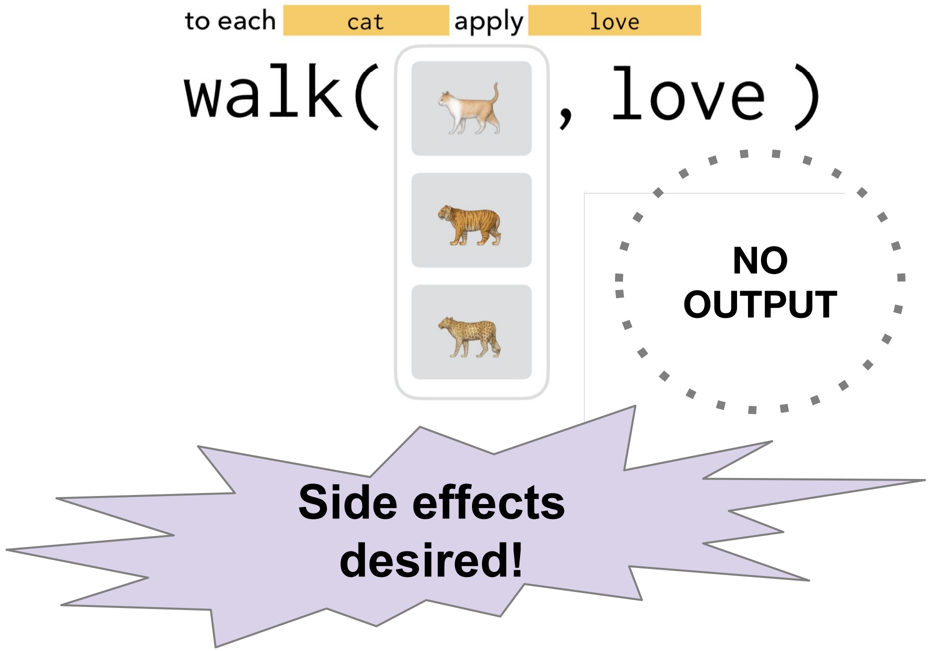
But sometimes we need side effects!

- Printing
- Plotting
- Saving output to disk

Reproducibility

Clearly separate pure functions from functions desired for their side effects





Side effects desired!

No side effects intended!
Returns something

Graphic adapted from Charlotte Wickham

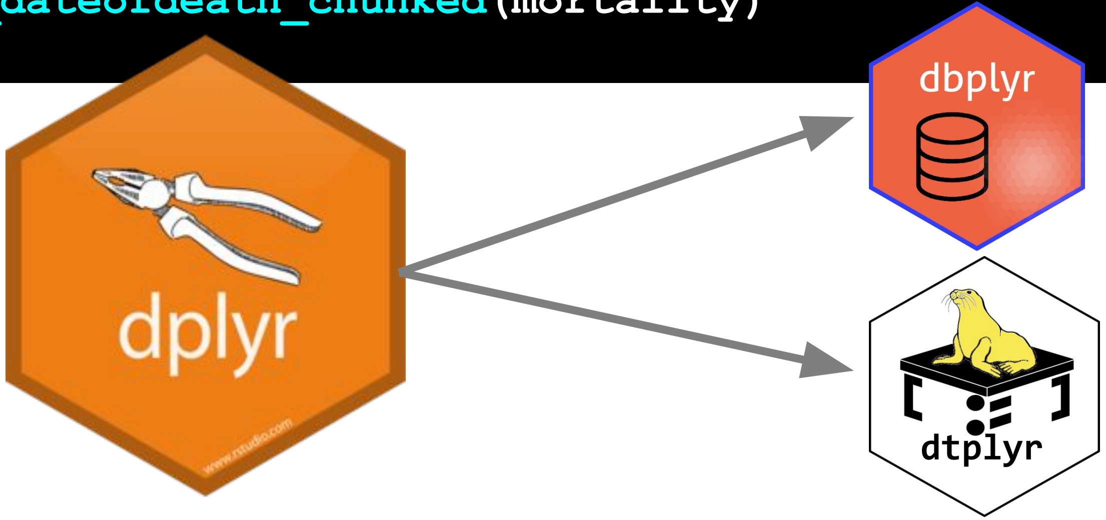
Robustness
Testability and elegant error handling

- Functional modular code makes for happy unit testing
  - Avoid difficult to test code stemming from **Mutability, Side-Effects**, **Responsibility overload**, and **Procedural instructions**
- Avoid side effects and maintain your functional flow
  - try-catch exception blocks can be refactored as higher order functions

```r
error_prone_fn <- function(data) {
    exprs
}

safe_fn <- purrr::safely(
    .f = error_prone_fn,
    otherwise = c())

data %>%
    mutate(results = purrr::map(
        input, safe_fn))
```

Efficiency
Delayed evaluation and lazy backends

```
parse_dateofdeath_chunked <- purrr::partial(dbplyr::do,
    data_modified = dplyr::mutate(., dateofdeath = parse_dod(a)),
    .chunk_size = 10000L)

parse_dateofdeath_chunked(mortality)
```



backend for databases

backend for data.tables

Efficiency

Caching with **memoise**

Use when repeatedly evaluating a function over rows/chunks of data, with some repeated inputs

Performance gains depend on how often a function is being called with same arguments

```
compute_biomarker_status <- function(
    biomarker_results){
    exprs
}

compute_biomarker_status_cached <-
    memoise::memoise(
        compute_biomarker_status)

data %<>%
  mutate(biomarker_status =
    purrr::map(biomarker_results,
        compute_biomarker_status_cached)
```

Thanks!

@jgutman

@dynamicdataduo