Tidy R programming with the OMOP common data model

Edward Burn, Adam Black, Marti Catala, Berta Raventós

2022-10-27T00:00:00+01:00

Table of contents

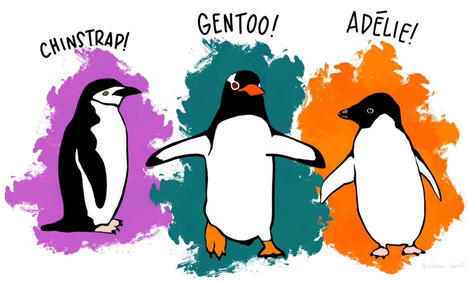
Preface

This book is written for analysts writing analytic code with R to run against the OMOP CDM. This source code for the book can be found at this Github repository Please open an issue there if you have a question or suggestion. Pull requests with suggested changes and additions are also most welcome.

1 Getting started with R

1.1 Installing R and R Studio

1.2 A first data analysis



Artwork by @allison_horst

For a quick example of a data analysis with R, let´s use the data from palmerpenguins package (https://allisonhorst.github.io/palmerpenguins/), which contains data on penguins collected from the Palmer Station in Antarctica.

Because we´ll be using a few packages not included in base R, first we need to install these if we don´t already have them.

```
install.packages("dplyr")
install.packages("ggplot2")
install.packages("palmerpenguins")
```

Once installed, we can load them like so.

```
library(dplyr)
library(ggplot2)
library(palmerpenguins)
```

We can get an overview of the data using the `glimpse()` command.

```r
glimpse(penguins)
```

```
Rows: 344
Columns: 8
$ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel~
$ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgerse~
$ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, ~
$ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, ~
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186~
$ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, ~
$ sex               <fct> male, female, female, NA, female, male, female, male~
$ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007~
```

Let´s get a count by species

```r
penguins %>%
  group_by(species) %>%
  count()
```

```
# A tibble: 3 x 2
# Groups:   species [3]
  species       n
  <fct>     <int>
1 Adelie      152
2 Chinstrap    68
3 Gentoo      124
```

Now suppose we are particularly interested in the body mass variable. We can first notice that there are a couple of missing records for this.

```r
penguins %>%
  group_by(species) %>%
  summarise(not_missing_body_mass_g=sum(!is.na(body_mass_g)==TRUE),
            missing_body_mass_g=sum(is.na(body_mass_g)==TRUE))
```

```
# A tibble: 3 x 3
  species   not_missing_body_mass_g missing_body_mass_g
  <fct>                       <int>               <int>
1 Adelie                        151                   1
2 Chinstrap                      68                   0
3 Gentoo                        123                   1
```

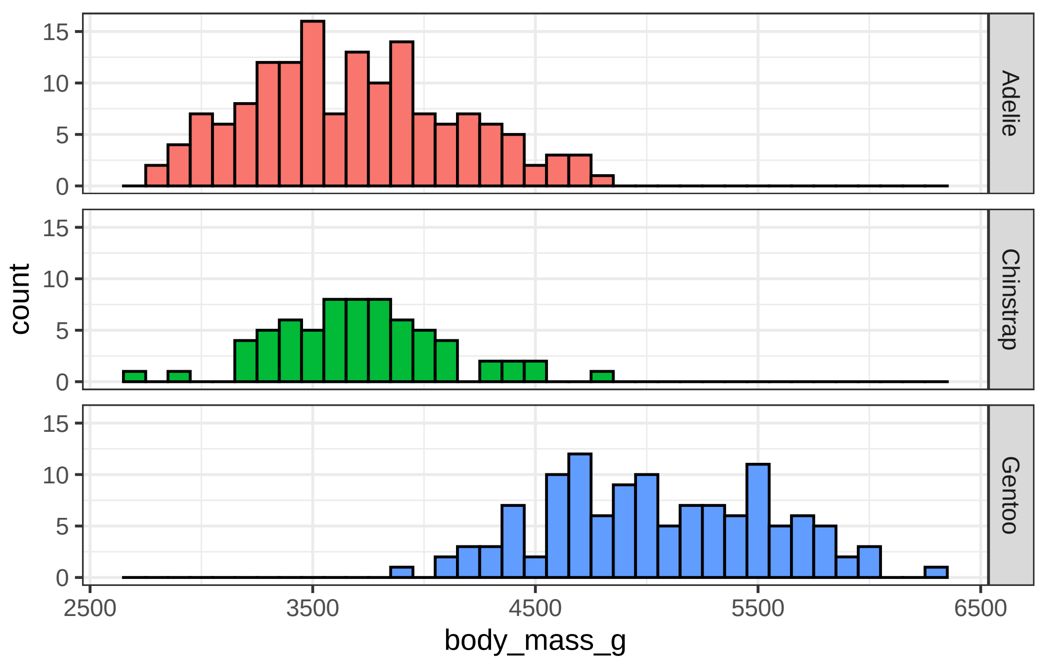
We can get the mean for each of the species (dropping those two missing records).

```
penguins %>%
  group_by(species) %>%
  summarise(mean_body_mass_g=round(mean(body_mass_g, na.rm=TRUE)))
```

```
# A tibble: 3 x 2
  species   mean_body_mass_g
  <fct>                <dbl>
1 Adelie                3701
2 Chinstrap             3733
3 Gentoo                5076
```

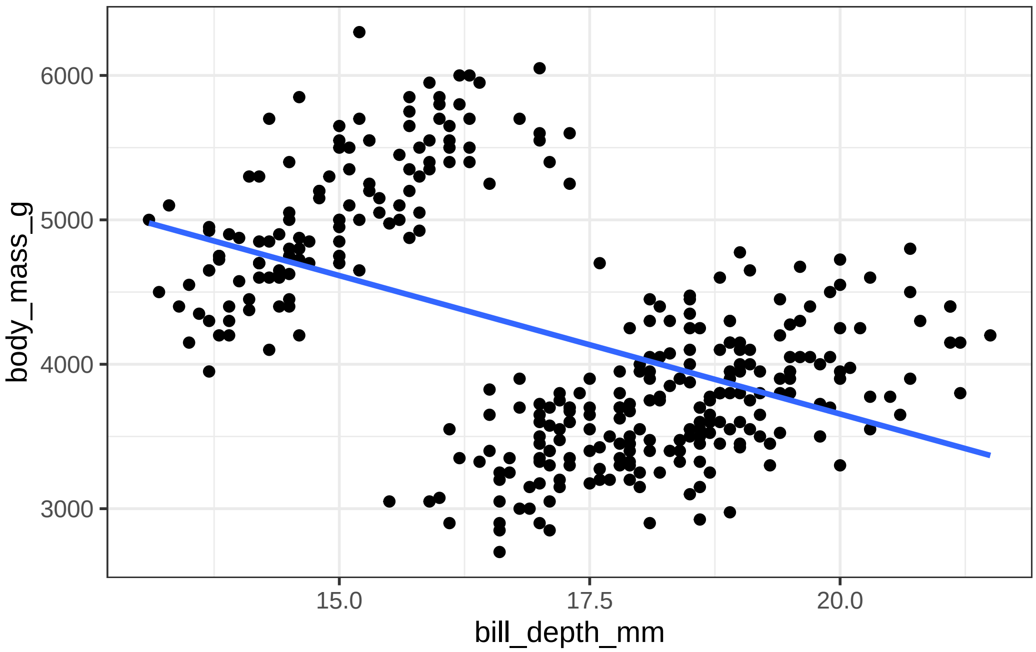
We can then also do a histogram for each of the species.

```
penguins %>%
  ggplot(aes(group=species, fill=species))+
  facet_grid(species~ .) +
  geom_histogram(aes(body_mass_g), colour="black", binwidth = 100)+
  theme_bw()+
  theme(legend.position = "none")
```



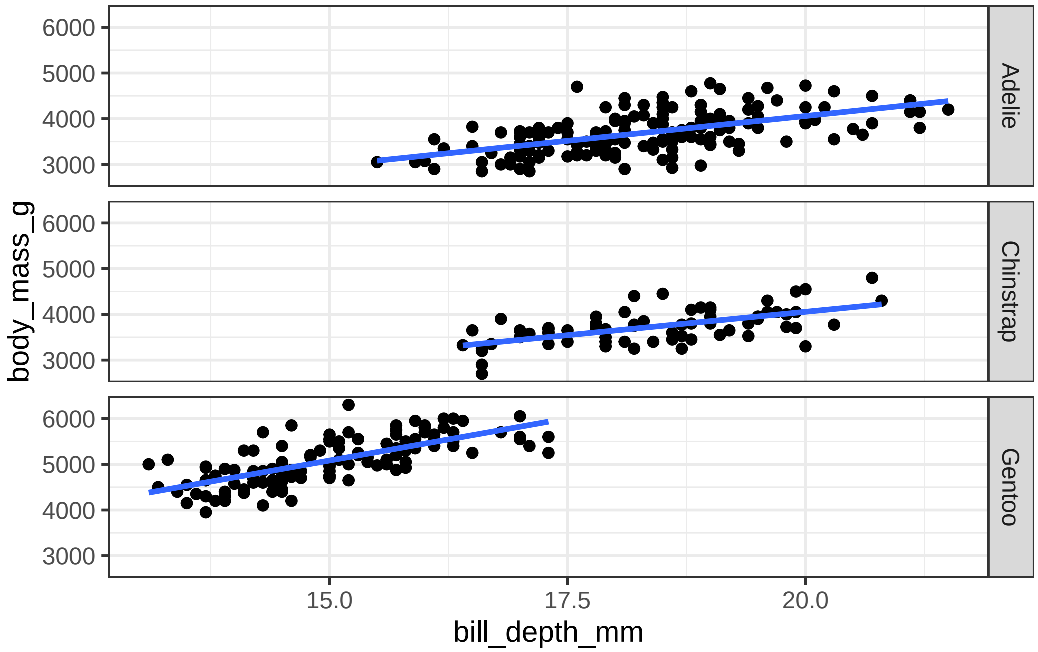
How about the relationship between body mass and bill depth?

```
penguins %>%
  ggplot(aes(x=bill_depth_mm,y=body_mass_g))+
  geom_point()+
  geom_smooth(method="lm",se=FALSE )+
  theme_bw()+
  theme(legend.position = "none")
```



But what about by species?

```
penguins %>%
  ggplot(aes(x=bill_depth_mm,y=body_mass_g))+
  facet_grid(species~ .) +
  geom_point()+
  geom_smooth(method="lm",se=FALSE )+
  theme_bw()+
  theme(legend.position = "none")
```



Oh, your first data analysis and you have already found an example of Simpson´s paradox!

2 Creating a reference to the common data model

2.0.1 Connecting to a database from R using DBI

Database connections from R can be made using the DBI package. The back-end for `DBI` is facilitate by database specific driver packages, with applications then using the front-end API. As an example, lets say we want to work with a local duckdb from R. In this case the we can use the duckdb R package as the driver. In this case we can also create the database in-memory

```r
library(DBI)
db<-dbConnect(duckdb::duckdb(), dbdir=":memory:")
```

If we instead wanted to connect to other database management systems, these connections could look like

```r
# Postgres
db <- DBI::dbConnect(RPostgres::Postgres(),
                      dbname = Sys.getenv("CDM5_POSTGRESQL_DBNAME"),
                      host = Sys.getenv("CDM5_POSTGRESQL_HOST"),
                      user = Sys.getenv("CDM5_POSTGRESQL_USER"),
                      password = Sys.getenv("CDM5_POSTGRESQL_PASSWORD"))
# Redshift (almost identical to Postgres)
db <- DBI::dbConnect(RPostgres::Redshift(),
                      dbname   = Sys.getenv("CDM5_REDSHIFT_DBNAME"),
                      host     = Sys.getenv("CDM5_REDSHIFT_HOST"),
                      port     = Sys.getenv("CDM5_REDSHIFT_PORT"),
                      user     = Sys.getenv("CDM5_REDSHIFT_USER"),
                      password = Sys.getenv("CDM5_REDSHIFT_PASSWORD"))
# SQL Server
db <- DBI::dbConnect(odbc::odbc(),
                      Driver   = "ODBC Driver 18 for SQL Server",
                      Server   = Sys.getenv("CDM5_SQL_SERVER_SERVER"),
                      Database = Sys.getenv("CDM5_SQL_SERVER_CDM_DATABASE"),
```

```r
                        UID      = Sys.getenv("CDM5_SQL_SERVER_USER"),
                        PWD      = Sys.getenv("CDM5_SQL_SERVER_PASSWORD"),
                        TrustServerCertificate="yes",
                        Port     = 1433)
```

2.0.2 Creating a reference to the OMOP common data model

If we have connected to a database which contains data mapped to the format of the OMOP common data model the CDMConnector provides functionality to simplify our work with a database. Because we already know the structure of the common data model, CDMConnector can be used to create a reference to the various tables that are used.

```r
library(CDMConnector)
```

```r
db <- DBI::dbConnect(duckdb::duckdb(),
                     dbdir = CDMConnector::eunomia_dir())
cdm <- CDMConnector::cdm_from_con(db,
                                  cdm_schema = "main")
cdm
```

```
# OMOP CDM reference (tbl_duckdb_connection)

Tables: person, observation_period, visit_occurrence, visit_detail, condition_occurrence, dru
```

From this reference we we can read the tables with "$" operator or [[" "]].

```r
cdm$observation_period
```

```
# Source:   table<main.observation_period> [?? x 5]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmp
   observation_period_id person_id observation_period_start~1 observat~2 perio~3
                   <dbl>     <dbl> <date>                     <date>       <dbl>
 1                     6         6 1963-12-31                 2007-02-06  4.48e7
 2                    13        13 2009-04-26                 2019-04-14  4.48e7
 3                    27        27 2002-01-30                 2018-11-21  4.48e7
 4                    16        16 1971-10-14                 2017-11-02  4.48e7
 5                    55        55 2009-05-30                 2019-03-23  4.48e7
 6                    60        60 1990-11-21                 2019-01-23  4.48e7
 7                    42        42 1909-11-03                 2019-03-13  4.48e7
```

```
 8                     33        33 1986-05-12                        2018-09-10  4.48e7
 9                     18        18 1965-11-17                        2018-11-07  4.48e7
10                     25        25 2007-03-18                        2019-04-07  4.48e7
# ... with more rows, and abbreviated variable names
#   1: observation_period_start_date, 2: observation_period_end_date,
#   3: period_type_concept_id
# i Use `print(n = ...)` to see more rows
```

```
cdm[["observation_period"]]
```

```
# Source:   table<main.observation_period> [?? x 5]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmp
   observation_period_id person_id observation_period_start~1 observat~2 perio~3
                   <dbl>     <dbl> <date>                     <date>       <dbl>
 1                     6         6 1963-12-31                 2007-02-06  4.48e7
 2                    13        13 2009-04-26                 2019-04-14  4.48e7
 3                    27        27 2002-01-30                 2018-11-21  4.48e7
 4                    16        16 1971-10-14                 2017-11-02  4.48e7
 5                    55        55 2009-05-30                 2019-03-23  4.48e7
 6                    60        60 1990-11-21                 2019-01-23  4.48e7
 7                    42        42 1909-11-03                 2019-03-13  4.48e7
 8                    33        33 1986-05-12                 2018-09-10  4.48e7
 9                    18        18 1965-11-17                 2018-11-07  4.48e7
10                    25        25 2007-03-18                 2019-04-07  4.48e7
# ... with more rows, and abbreviated variable names
#   1: observation_period_start_date, 2: observation_period_end_date,
#   3: period_type_concept_id
# i Use `print(n = ...)` to see more rows
```

When we create our cdm reference we could have also specified the tables we want to read:

```
cdm <- CDMConnector::cdm_from_con(db,
                                  cdm_tables = c("person","observation_period"))
cdm
```

```
# OMOP CDM reference (tbl_duckdb_connection)

Tables: person, observation_period
```

Moreover, we can also specify the writable schema and the tables that we are interested on it. For example, if we wanted to create a reference to the person and observation period tables in the common data model along with exposure and outcome cohort tables in a schema we have write access to, we could do this like so:

```
cdm <- CDMConnector::cdm_from_con(db,
  cdm_schema = "main",
  cdm_tables = c("person","observation_period"),
  write_schema = "results",
  cohort_tables = c("exposure_cohort", "outcome_cohort"))
```

3 Exploring the CDM

Let's first connect again to our Eunomia data and create the reference to the common data model.

```
library(DBI)
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

3.0.1 tally()

Let's say we want to get a count of the people in the person table. For this we can use the tally or count verbs from dbplyr

```
cdm$person %>%
  count()
```

```
# Source:   SQL [1 x 1]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmp
      n
  <dbl>
1  2694
```

This count was done on the database side, with the code we wrote in dplyr style translated into sql.

```
cdm$person %>%
  count() %>%
  show_query()
```

```
<SQL>
SELECT COUNT(*) AS n
FROM main.person
```

3.0.2 summarise()

Another way to get the same count would be to use the summarise verb

```
cdm$person %>%
  summarise(n = n())
```

```
# Source:   SQL [1 x 1]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtm
      n
  <dbl>
1  2694
```

```
cdm$person %>%
  summarise(n = n())%>%
  show_query()
```

```
<SQL>
SELECT COUNT(*) AS n
FROM main.person
```

We can also use summarise for various other calculations

```
cdm$person %>%
  summarise(median = median(year_of_birth, na.rm=TRUE))
```

```
# Source:   SQL [1 x 1]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtm
  median
   <dbl>
1   1961
```

```
cdm$person %>%
  summarise(median = median(year_of_birth, na.rm=TRUE))%>%
  show_query()
```

```
<SQL>
SELECT PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY year_of_birth) AS median
FROM main.person
```

3.0.3 group_by()

What if we want to get a count of people in the person table by gender concept id? In this case we can use group_by

```
cdm$person %>%
  group_by(gender_concept_id) %>%
  count()
```

```
# Source:   SQL [2 x 2]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmp
  gender_concept_id     n
              <dbl> <dbl>
1              8532  1373
2              8507  1321
```

```
cdm$person %>%
  group_by(gender_concept_id) %>%
  count() %>%
  show_query()
```

```
<SQL>
SELECT gender_concept_id, COUNT(*) AS n
FROM main.person
GROUP BY gender_concept_id
```

Similarly we could use group_by to calculate median year of birth by gender concept id.

```
cdm$person %>%
  group_by(gender_concept_id) %>%
```

```
  summarise(median = median(year_of_birth, na.rm=TRUE))
```

```
# Source:   SQL [2 x 2]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmp
  gender_concept_id median
              <dbl>  <dbl>
1              8532   1961
2              8507   1961
```

```
cdm$person %>%
  group_by(gender_concept_id) %>%
  summarise(median = median(year_of_birth, na.rm=TRUE)) %>%
  show_query()
```

```
<SQL>
SELECT
  gender_concept_id,
  PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY year_of_birth) AS median
FROM main.person
GROUP BY gender_concept_id
```

3.0.4 filter()

Or if we wanted a count within only for those with a specific gender concept id we can use the filter verb to subset the data before summarising it

```
cdm$person %>%
  filter(gender_concept_id == "8532") %>%
  count()
```

```
# Source:   SQL [1 x 1]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmp
      n
  <dbl>
1  1373
```

```
cdm$person %>%
  filter(gender_concept_id == "8532") %>%
```

```
  count() %>%
  show_query()
```

```
<SQL>
SELECT COUNT(*) AS n
FROM main.person
WHERE (gender_concept_id = '8532')
```

Similarly we could have

```
cdm$person %>%
  filter(year_of_birth < 1970) %>%
  summarise(median = median(year_of_birth, na.rm=TRUE))
```

```
# Source:   SQL [1 x 1]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmp
  median
   <dbl>
1   1955
```

```
cdm$person %>%
  filter(year_of_birth < 1970) %>%
  summarise(median = median(year_of_birth, na.rm=TRUE))%>%
  show_query()
```

```
<SQL>
SELECT PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY year_of_birth) AS median
FROM main.person
WHERE (year_of_birth < 1970.0)
```

We can combine the above, with a filter, followed by a group_by, and then followed by a summarise

```
cdm$person %>%
  filter(year_of_birth < 1970) %>%
  group_by(gender_concept_id) %>%
  summarise(median = median(year_of_birth, na.rm=TRUE))
```

```
# Source:   SQL [2 x 2]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmp
  gender_concept_id median
              <dbl>  <dbl>
1              8532   1955
2              8507   1956
```

```
cdm$person %>%
  filter(year_of_birth < 1970) %>%
  group_by(gender_concept_id) %>%
  summarise(median = median(year_of_birth, na.rm=TRUE))%>%
  show_query()
```

```
<SQL>
SELECT
  gender_concept_id,
  PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY year_of_birth) AS median
FROM main.person
WHERE (year_of_birth < 1970.0)
GROUP BY gender_concept_id
```

4 Working with databases from R

Let's start by taking some data and putting it in a database. Here we'll use an in-memory duckdb database, but the same code should work for other databases with only the connection details and the package used to connect to the database changing.

For this example let's use data on Darwin's finches as that seems rather appropriate (link to wiki article on darwin finches)

```
# install packages
# commented out as you might already have them
# but if not then uncomment and run
# install.packages("DBI")
# install.packages("SQLite")
# install.packages("dbplyr")
# install.packages("dplyr")
#
# # load packages
# library(DBI)
# library(SQLite)
# library(dbplyr)
# library(dplyr)

# get data

# move into a database
```

4.1 show_query()

4.2 filter(), select(), mutate()

4.3 right_join(), left_join(), inner_join(), and anti_join()

4.4 summarise()

4.5 collect() and compute()

4.6 working with dates

Here be dragons

4.7 working with strings

4.8 bespoke sql

Alternative approaches

1) Where to do computation

- Database side vs in local memory vs R

2) Scope of a package

3) Scope of analysis code All in one vs one at a time

5

strings

6

Dates

7 right_join(), left_join(), inner_join(), and anti_join()

7.0.1 and union() and union_all()

8 Getting to tidy data

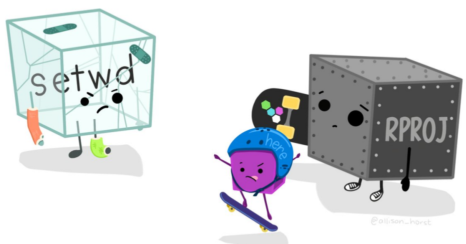
8.0.1 compute()

8.0.2 collect()

8.0.3 pull()

9 Analysis in R

10 Organising data analyses with projects and renv



Artwork by @allison_horst

References

11

Learning R