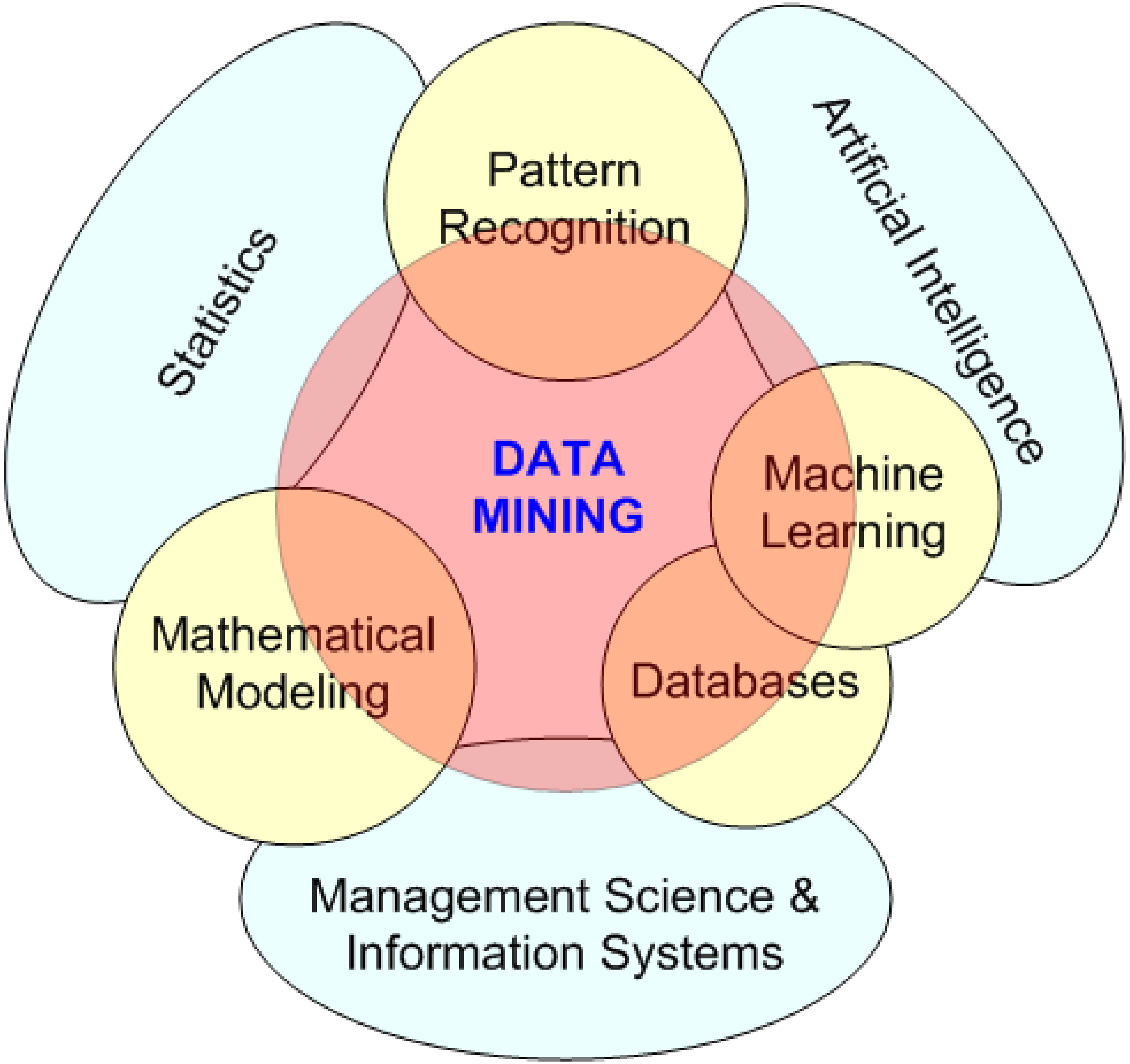
EDA II: Data Wrangling

Yvonne Phillips, Trainer
yphillips@beverasolutions.com
Bevera Solutions


Pattern Recognition
Statistics
Artificial Intelligence
DATA MINING
Machine Learning
Mathematical Modeling
Databases
Management Science & Information Systems

Learning objectives of this module:

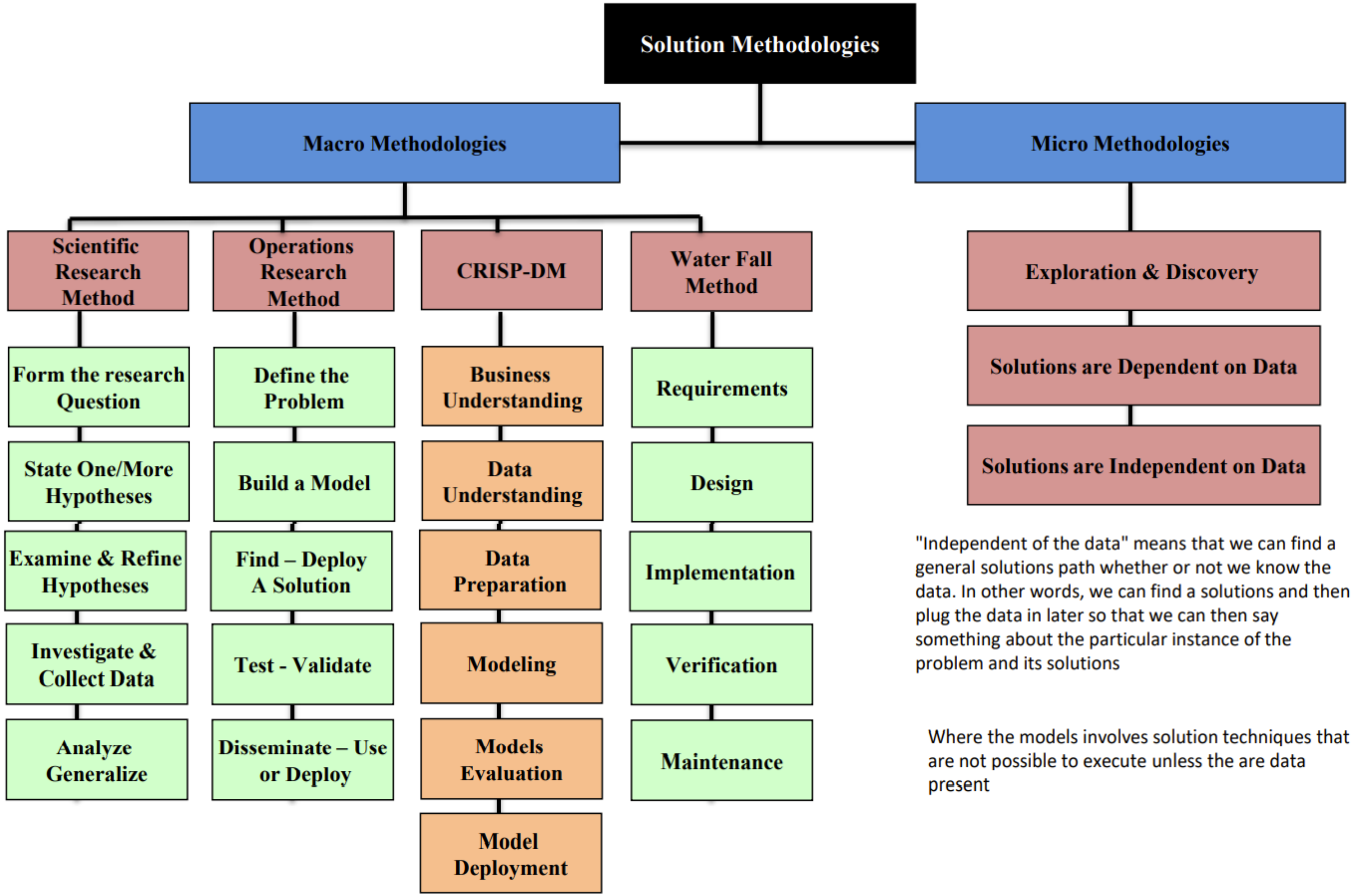
- Data Understanding
- Visit the tidyr package
- Exercise commands
- Introduction to data analysis
- Learn the basic vocabulary of dplyr

DATA WRANGLING: WHAT IT IS & WHY IT'S IMPORTANT

Article read

- https://online.hbs.edu/blog/post/data-wrangling

Technical Solution Methodologies



"Independent of the data" means that we can find a general solutions path whether or not we know the data. In other words, we can find a solutions and then plug the data in later so that we can then say something about the particular instance of the problem and its solutions

Where the models involves solution techniques that are not possible to execute unless the are data present

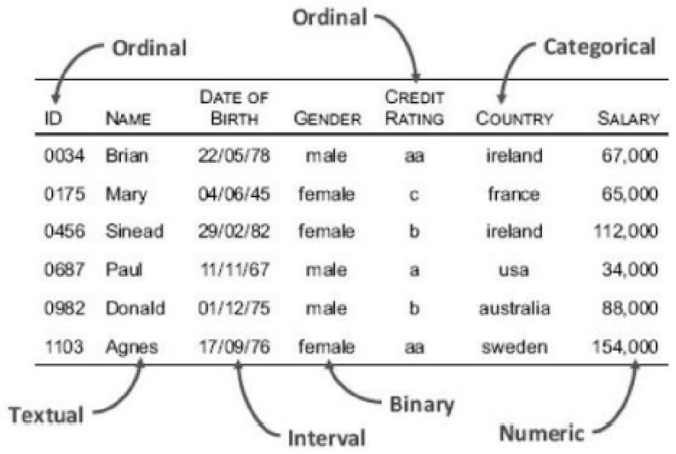
Data Scientist Day to Day Activities (CRISP-DM)

Business Understanding	Data Understanding	Data Preparation		Modeling	Optimization	Deployment
Determine Business Objectives	Design Features	Transform/Fix Target Variable	Data Normalization	Select The Model	Model Selection	Planning Deployment
Frame the Problem Assess Feasibility	Collect Initial Data	Redundant & Duplicates	Data Factorization	Split Data	Model Optimization	Monitoring & Maintenance
Define Success Measurements	Install & Import Packages	Data Quality Audit (Missing Values)	Data Binarization	Data Scaling	Parameters Tuning	Final Report
Identify Target Variables (Y)	Read the Data	Data Quality Audit (Outliers)	Data Standardizing	Dummy Model		Lessons Learned
Identify Analytical Approach	Data Manipulation & Wrangling	Data Quality Audit (Cardinality Check)	Data Correlations	Build Model		
Identify Deployment Plan	Exploratory Data Analysis (EDA)	Data Conversion	Data Aggregation Binning	Fit Model (Train)		
Produce Project Plan	Data Visualization	Data Transformation	Data Decomposition	Predict (Test)		
Identify the team & Stakeholders	Statistical Analysis	Feature Engineering (Importance, Low variance, PCA)	Feature Selections	Assess & Evaluate		
Analytics Base Table (ABT)	Code Book Quality Report	Data Version 2/3/4		Best Model	Best Parameters	ROI

Designing and Implementing Features

- Design and implement concrete feature based on the concepts
- A feature is any measure derived from a domain concept that can be directly included in an analytics-based table (ABT) for use by a machine learning algorithm
- Often it will take multiple features to express a domain concept
- We may have to use some proxy features to capture something that is closely related to a domain concept when direct measurement is not possible
- In some extreme cases we may have to abandon a domain concept completely if the data required to express it isn’t available

Different Types of Features



ID	NAME	DATE OF BIRTH	GENDER	CREDIT RATING	COUNTRY	SALARY
0034	Brian	22/05/78	male	aa	ireland	67,000
0175	Mary	04/06/45	female	c	france	65,000
0456	Sinead	29/02/82	female	b	ireland	112,000
0687	Paul	11/11/67	male	a	usa	34,000
0982	Donald	01/12/75	male	b	australia	88,000
1103	Agnes	17/09/76	female	aa	sweden	154,000

Sample descriptive feature data illustrating numeric, binary, ordinal, interval, categorical, and textual types.

The features in an ABT can be of two types: Raw features or Derived features.

- **Raw features**: are features that come directly from raw data sources. For example, patient age, patient gender, drug treatment amount, or blood type are all descriptive features that we would most likely be able to transfer directly from a raw data source to an ABT.
- **Derived descriptive features**: do not exist in any raw data source, so they must be constructed from data in one or more raw data sources.

Different Types of Features

- Aggregates: are measures defined over a group or period and are usually defined as the count, sum, average, minimum, or maximum of the values within a group.
- Flags: are binary features that indicate presence or absence of some characteristic within a dataset. For example, a flag indicating whether a bank account has ever been overdrawn might be a useful descriptive feature.
- Ratios: are continuous features that capture the relationship between two or more raw data values.
- Mappings: are used to convert continuous features into categorical features and are often used to reduce the number of unique values that a model will have to deal with.
- Other: There are no restrictions to the ways in which we can combine data to make derived features.

Now that data has been collected for a modeling project, it needs to be examined so the analyst knows what is there. In many situations, the analyst is the first person to even look at the data as compiled into the modeling table in-depth. The analyst, therefore, will see all the imperfections and problems in the data that were previously unknown or ignored. Without Data Understanding, you don't know what problems may arise in modeling.

Data Understanding, as the first analytical step in predictive modeling, has the following purposes:

■ **Examine** key summary characteristics about the data to be used for modeling, including how many records are available, how many variables are available, and how many target variables are included in the data.

■ Begin to **enumerate** problems with the data, including inaccurate or invalid values, missing values, unexpected distributions, and outliers.

■ **Visualize** data to gain further insights into the characteristics of the data, especially those masked by summary statistics.

Exploratory Data Analysis (EDA) and Visualization are very important steps in any analysis task.

- Get to know your data!

✓ Distributions (symmetric, normal, skewed)

✓ Data quality problems

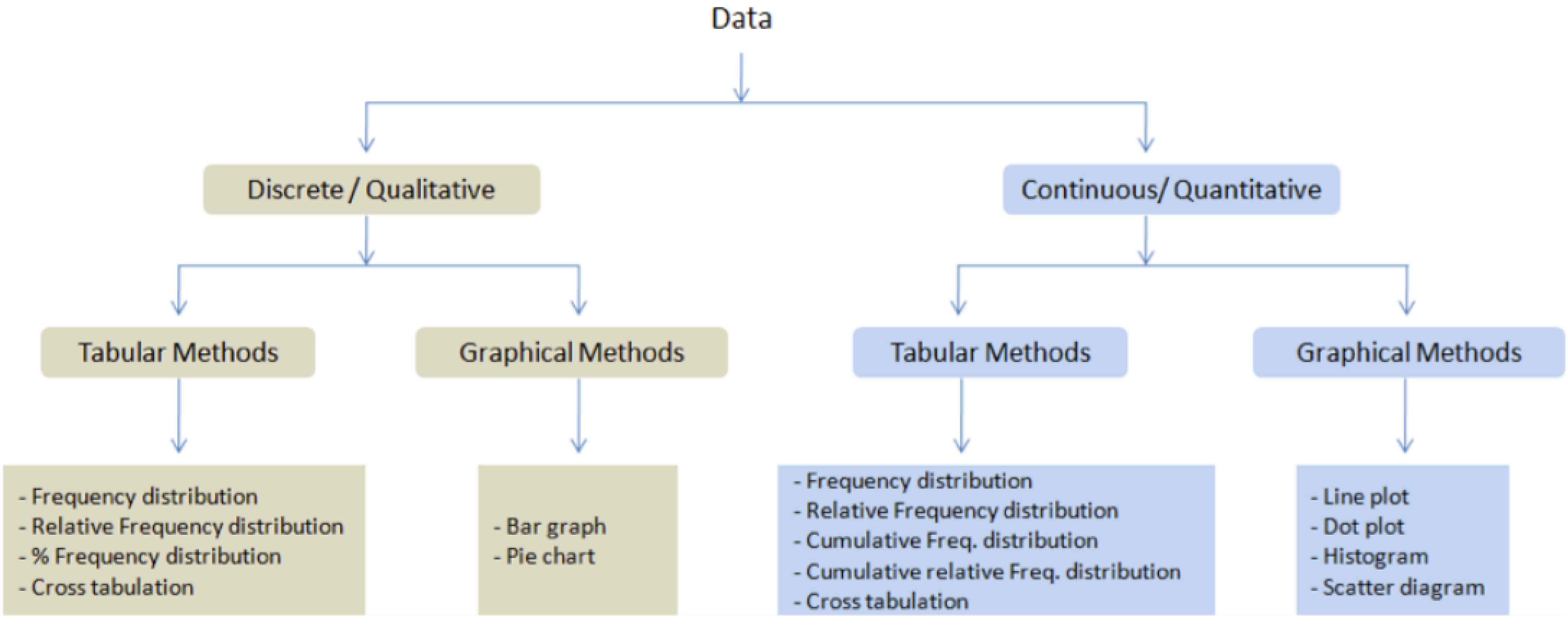
✓ Outliers

✓ Correlations and inter-relationships

✓ Subsets of interest

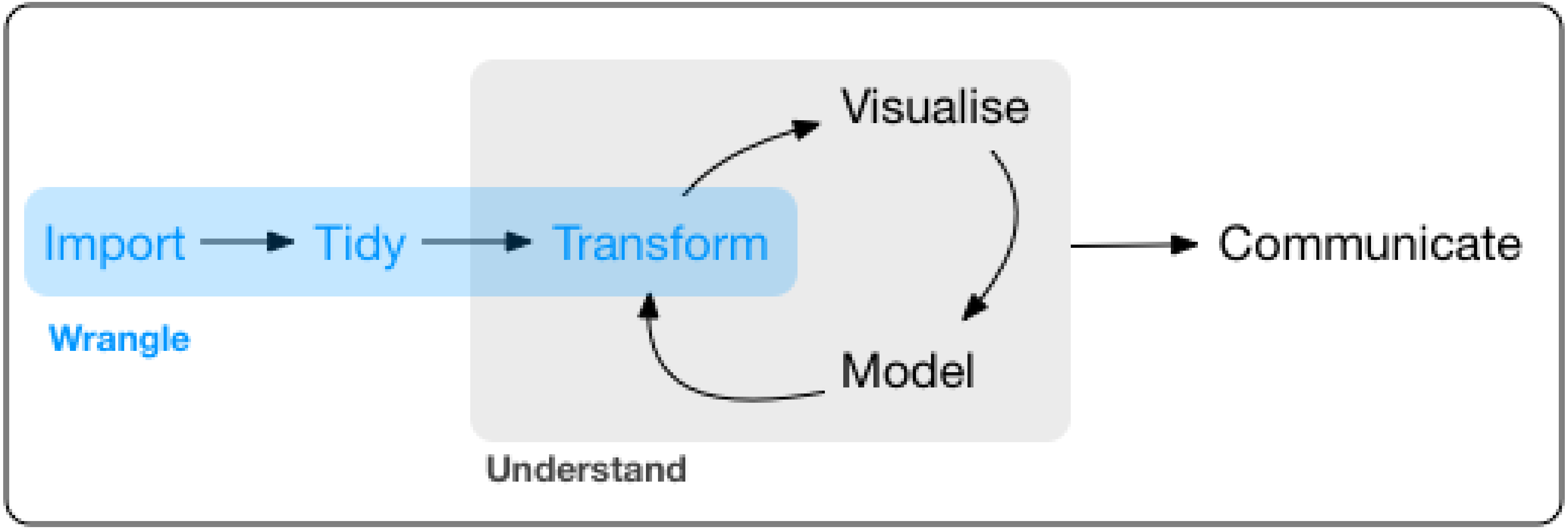
✓ Suggest functional relationships

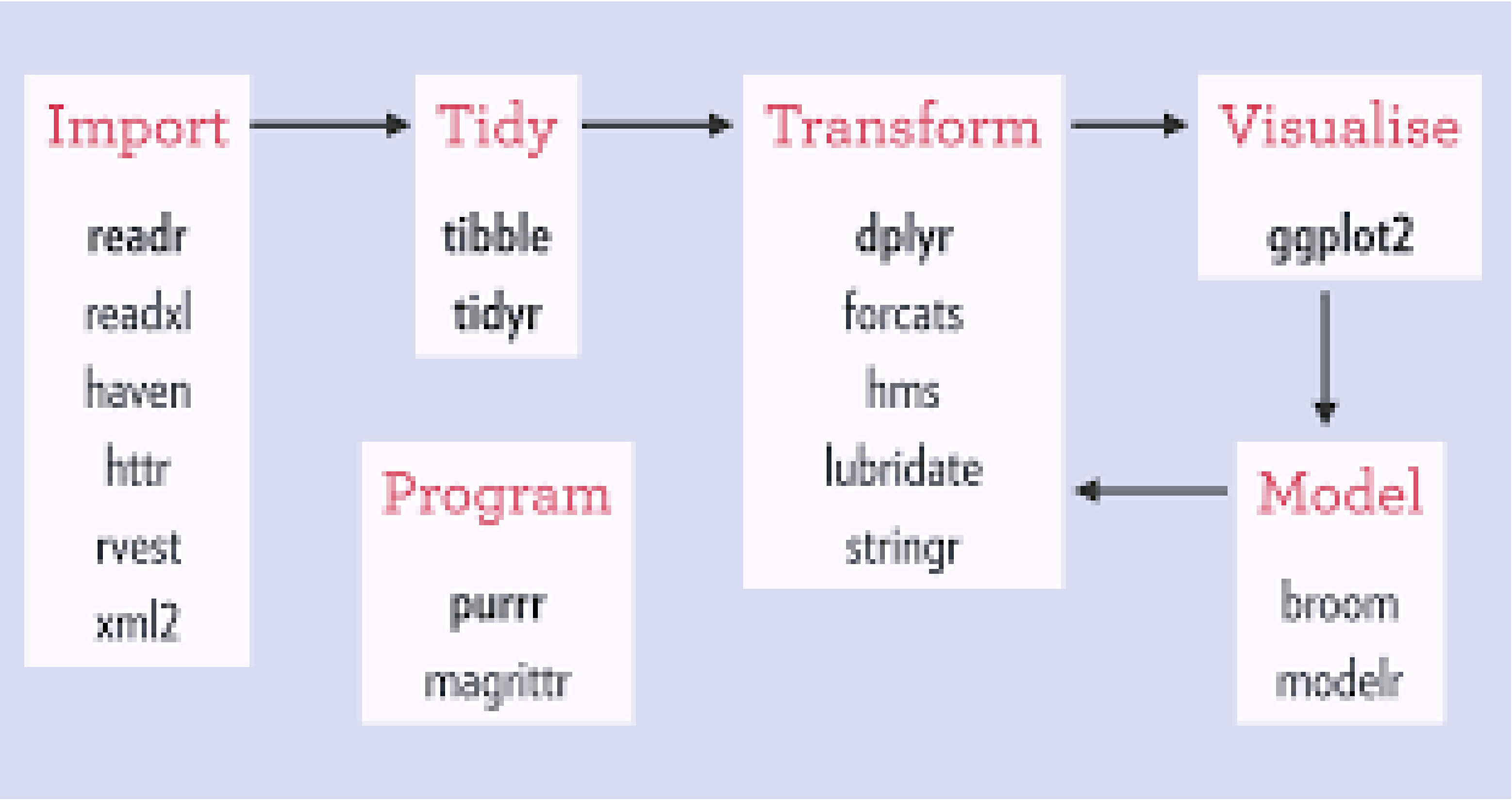
A Simple Taxonomy of Data



Two goals

1. Make data suitable to use with a particular piece of software
2. Reveal information


Visualise
Import
Tidy
Transform
Communicate
Wrangle
Model
Understand



https://www.tidyverse.org/

```
install.packages("tidyverse")

library("tidyverse")
```

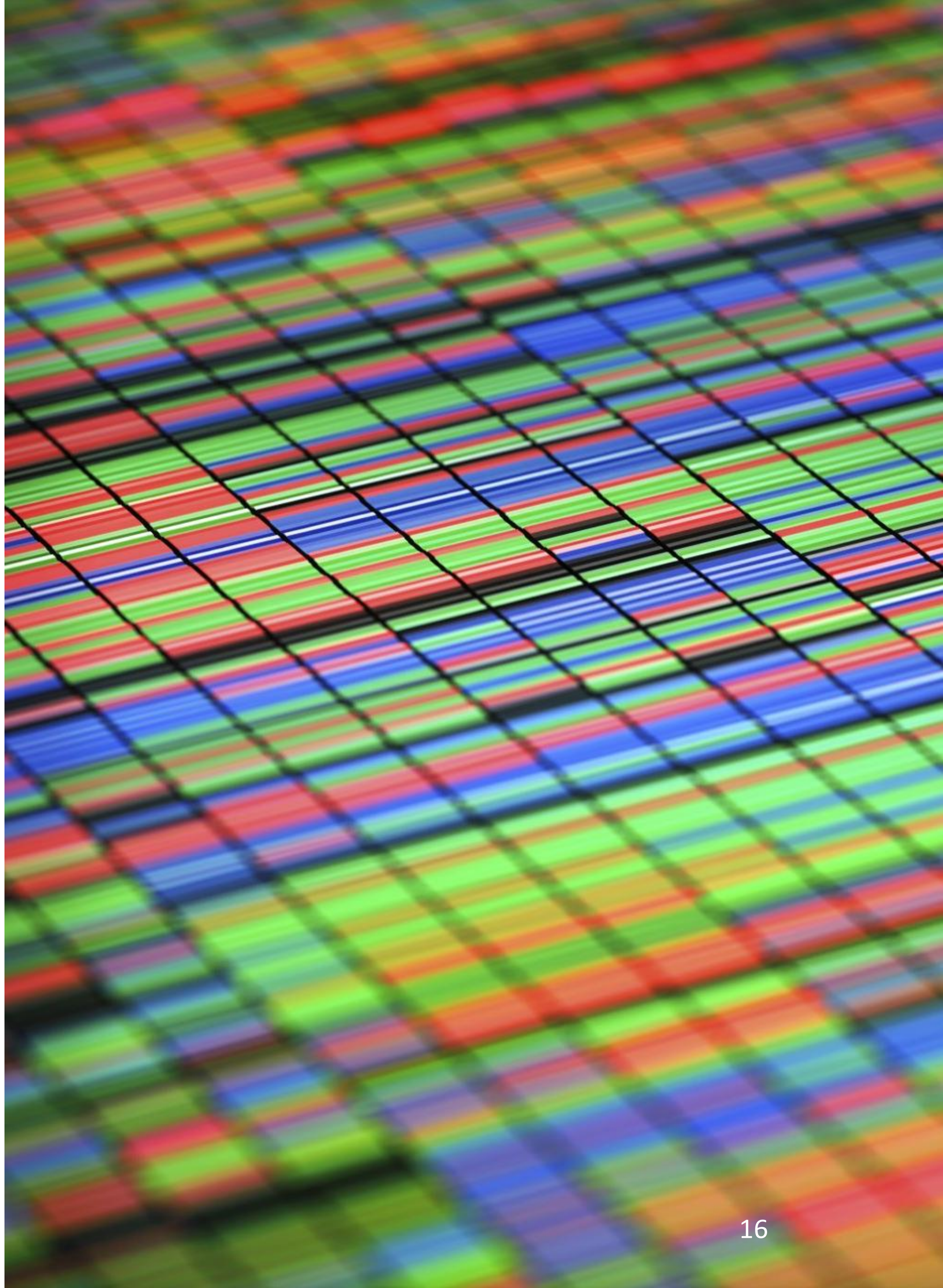
https://github.com/rstudio/master-the-tidyverse/archive/master.zip

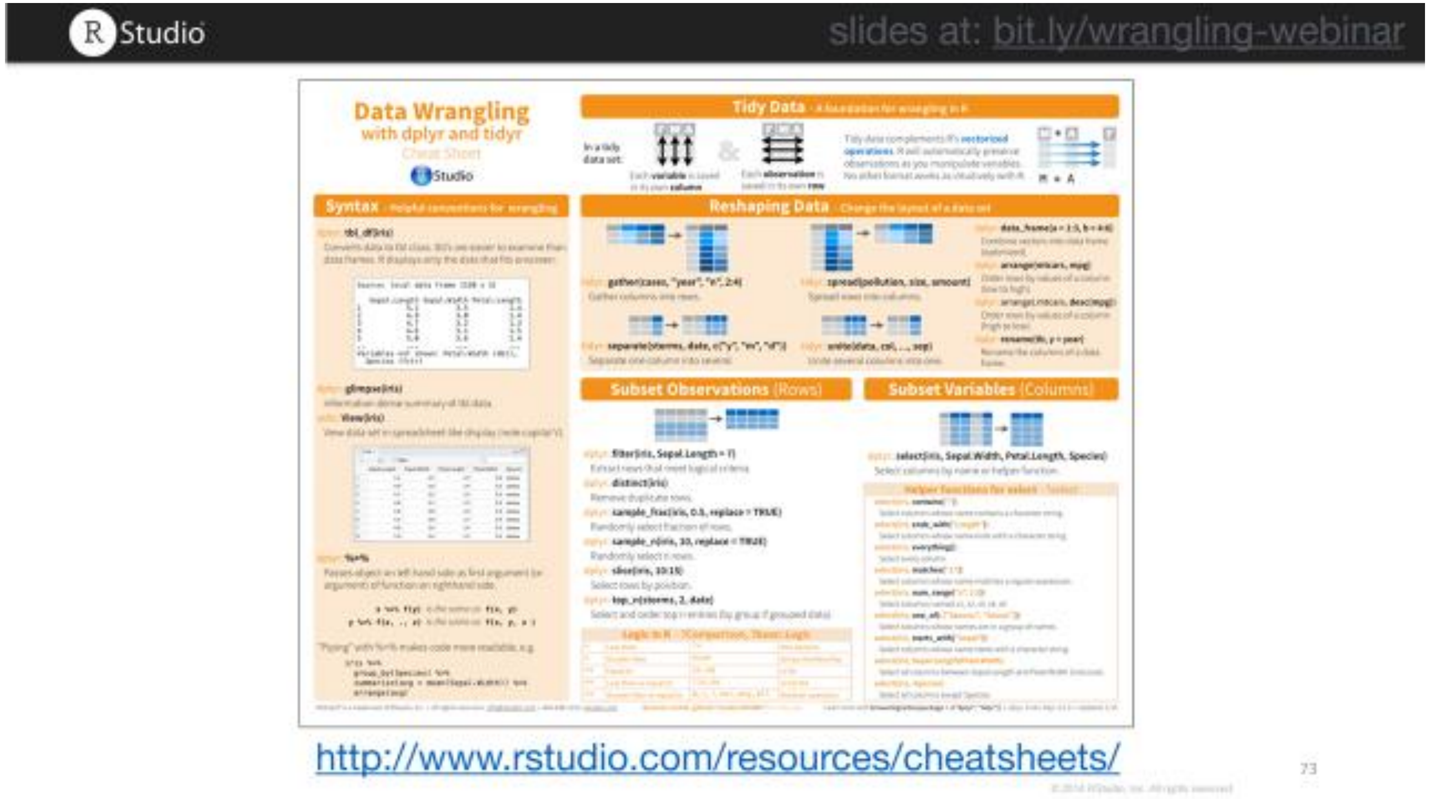
Untidy Data

There are various features of messy data that one can observe in practice.

Here are some of the more commonly observed patterns.

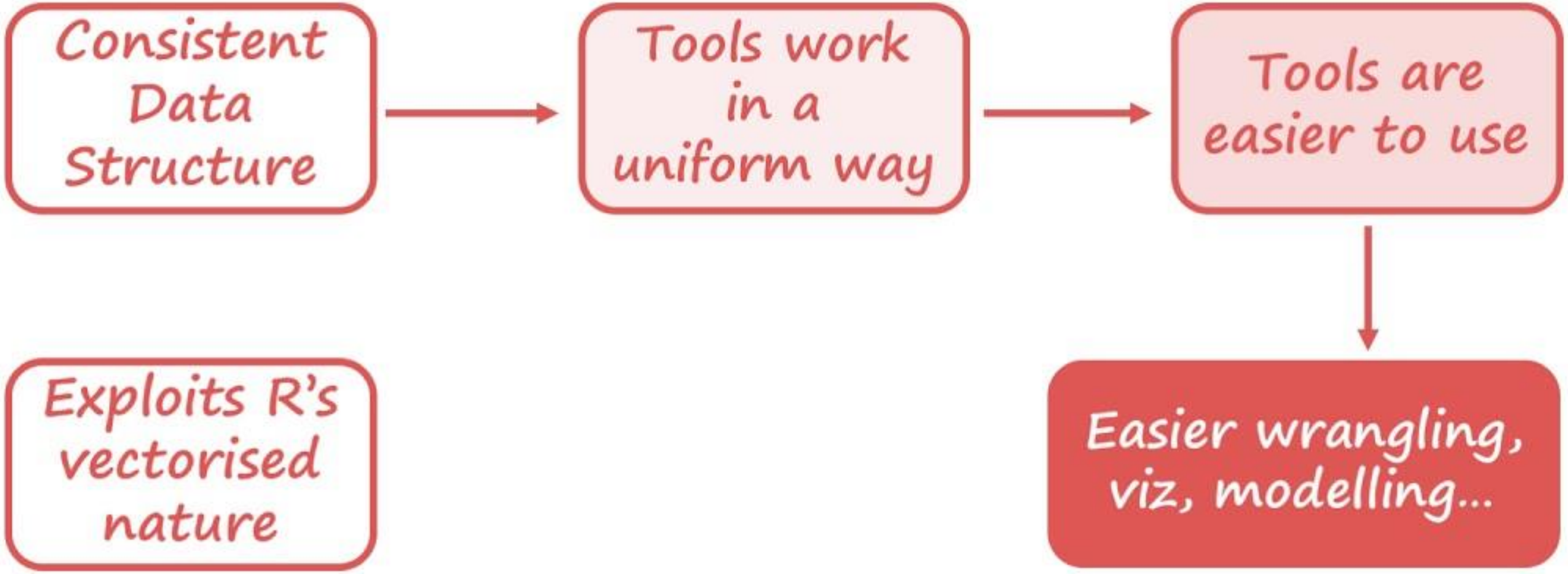
- Column headers are values, not variable names
- Multiple variables are stored in one column
- Variables are stored in both rows and columns
- Multiple types of experimental unit stored in the same table
- One type of experimental unit stored in multiple tables





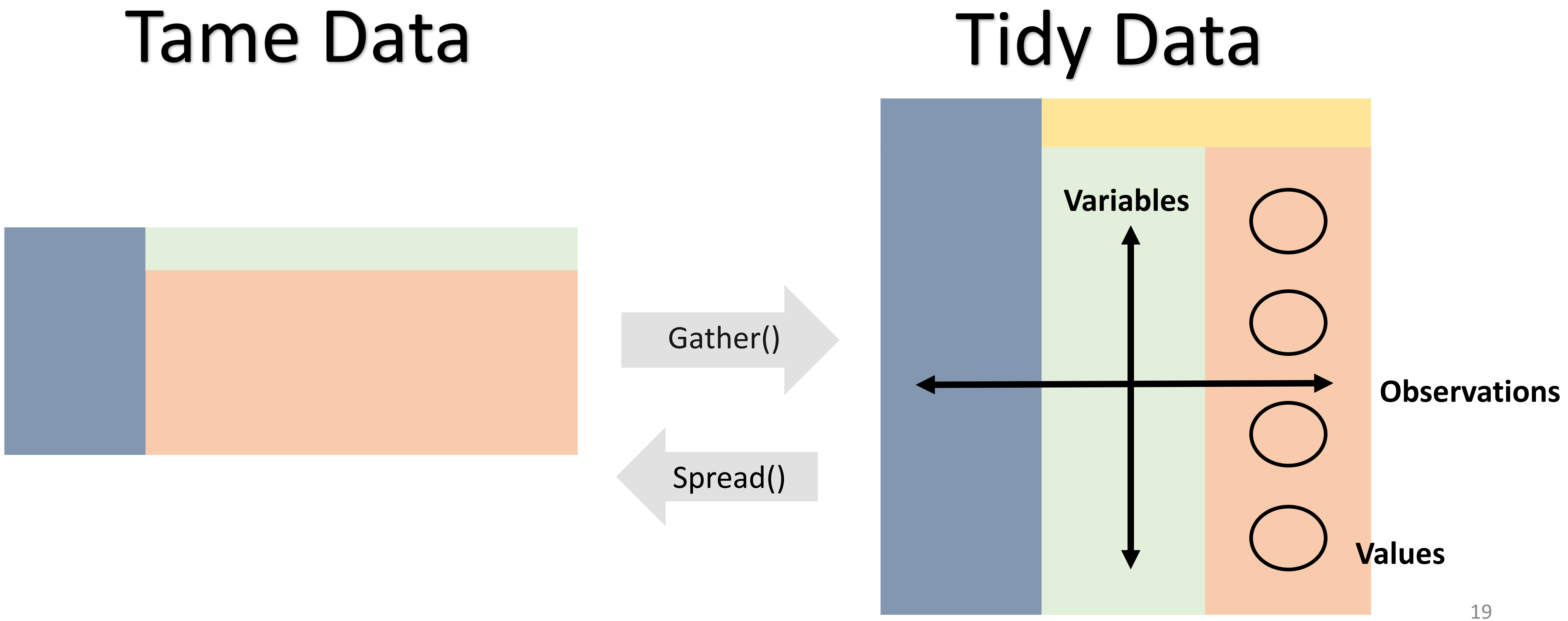
- Tidy Data - A foundation for wrangling in R
    - Syntax - Helpful conventions for wrangling
    - Reshaping Data - Change the layout of a data set
    - Subset Observations (Rows), Subset Variables (Columns)
    - Summarize Data
    - Make New Variables
    - Combine Data Sets
    - Group Data

Why tidy data?



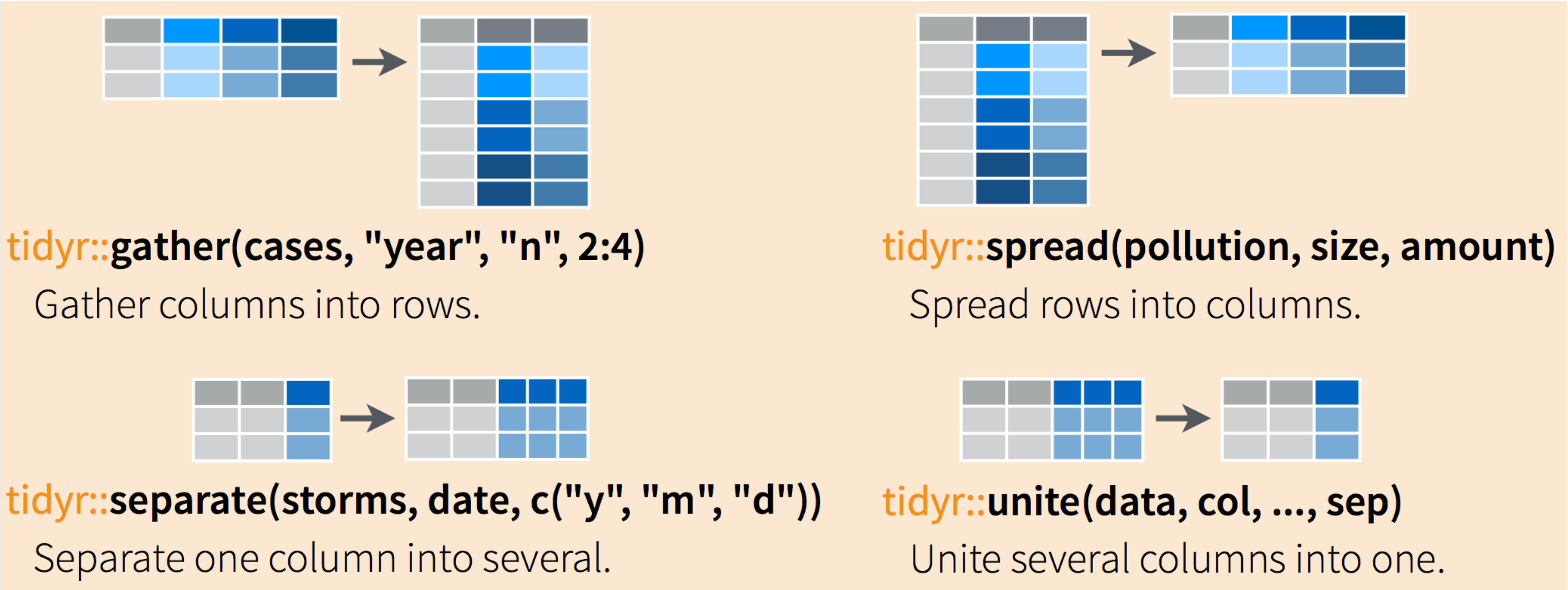
The tidyr package is used to manipulate the structure of your data while preserving all original information, using the following functions:

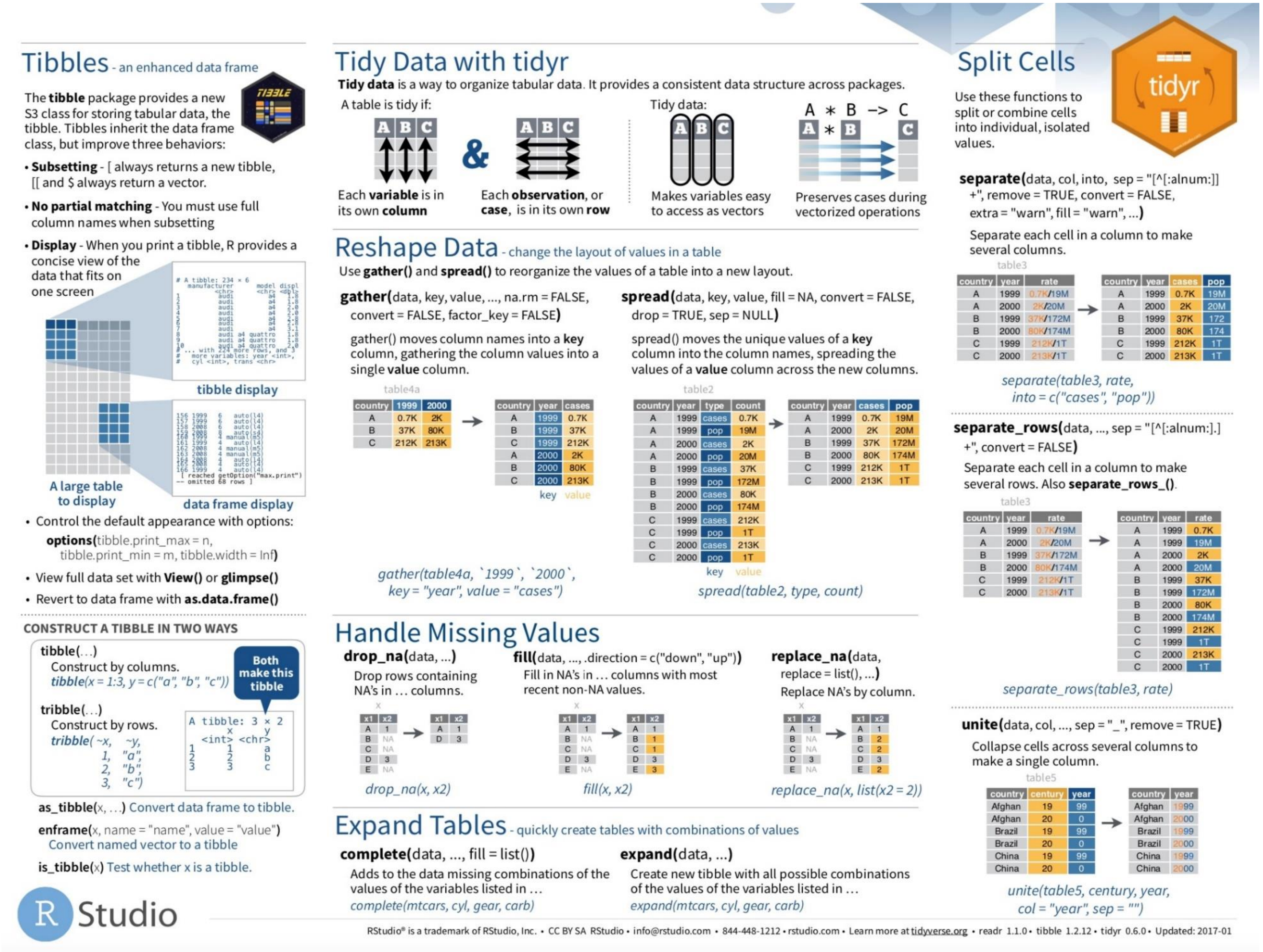
gather() our data (wide –> long)
spread() our data (long –> wide)



There are four main verbs which are essentially pairs of opposites:

turn columns into rows (gather()),
turn rows into columns (spread()),
turn a character column into multiple columns (separate()),
turn multiple character columns into a single column (unite())





Tibbles - an enhanced data frame

The **tibble** package provides a new S3 class for storing tabular data, the tibble. Tibbles inherit the data frame class, but improve three behaviors:

- **Subsetting** - [always returns a new tibble, [[and $ always return a vector.
- **No partial matching** - You must use full column names when subsetting
- **Display** - When you print a tibble, R provides a concise view of the data that fits on one screen

A large table to display

tibble display

data frame display

- Control the default appearance with options:

  options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf**)**

- View full data set with **View()** or **glimpse()**
- Revert to data frame with **as.data.frame()**

CONSTRUCT A TIBBLE IN TWO WAYS

tibble(…**)**
Construct by columns.
tibble(x = 1:3, y = c("a", "b", "c"))

Both make this tibble

tribble(…**)**
Construct by rows.
tribble(~x, ~y, 1, "a", 2, "b", 3, "c")

```
A tibble: 3 × 2
      x     y
  <int> <chr>
1     1     a
2     2     b
3     3     c
```

as_tibble(x, …**)** Convert data frame to tibble.

enframe(x, name = "name", value = "value"**)** Convert named vector to a tibble

is_tibble(x**)** Test whether x is a tibble.

Tidy Data with tidyr

Tidy data is a way to organize tabular data. It provides a consistent data structure across packages.

A table is tidy if: Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

Tidy data: Makes variables easy to access as vectors. A * B -> C: Preserves cases during vectorized operations

Reshape Data - change the layout of values in a table

Use **gather()** and **spread()** to reorganize the values of a table into a new layout.

gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE**)**

gather() moves column names into a **key** column, gathering the column values into a single **value** column.

gather(table4a, `1999`, `2000`, key = "year", value = "cases")

spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL**)**

spread() moves the unique values of a **key** column into the column names, spreading the values of a **value** column across the new columns.

spread(table2, type, count)

Handle Missing Values

drop_na(data, ...**)** Drop rows containing NA's in … columns. *drop_na(x, x2)*

fill(data, ..., .direction = c("down", "up")**)** Fill in NA's in … columns with most recent non-NA values. *fill(x, x2)*

replace_na(data, replace = list(), ...**)** Replace NA's by column. *replace_na(x, list(x2 = 2))*

Expand Tables - quickly create tables with combinations of values

complete(data, ..., fill = list()**)** Adds to the data missing combinations of the values of the variables listed in … *complete(mtcars, cyl, gear, carb)*

expand(data, ...**)** Create new tibble with all possible combinations of the values of the variables listed in … *expand(mtcars, cyl, gear, carb)*

Split Cells

Use these functions to split or combine cells into individual, isolated values.

separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", ...**)**

Separate each cell in a column to make several columns.

separate(table3, rate, into = c("cases", "pop"))

separate_rows(data, ..., sep = "[^[:alnum:].]+", convert = FALSE**)**

Separate each cell in a column to make several rows. Also **separate_rows_()**.

separate_rows(table3, rate)

unite(data, col, ..., sep = "_", remove = TRUE**)**

Collapse cells across several columns to make a single column.

unite(table5, century, year, col = "year", sep = "")

RStudio

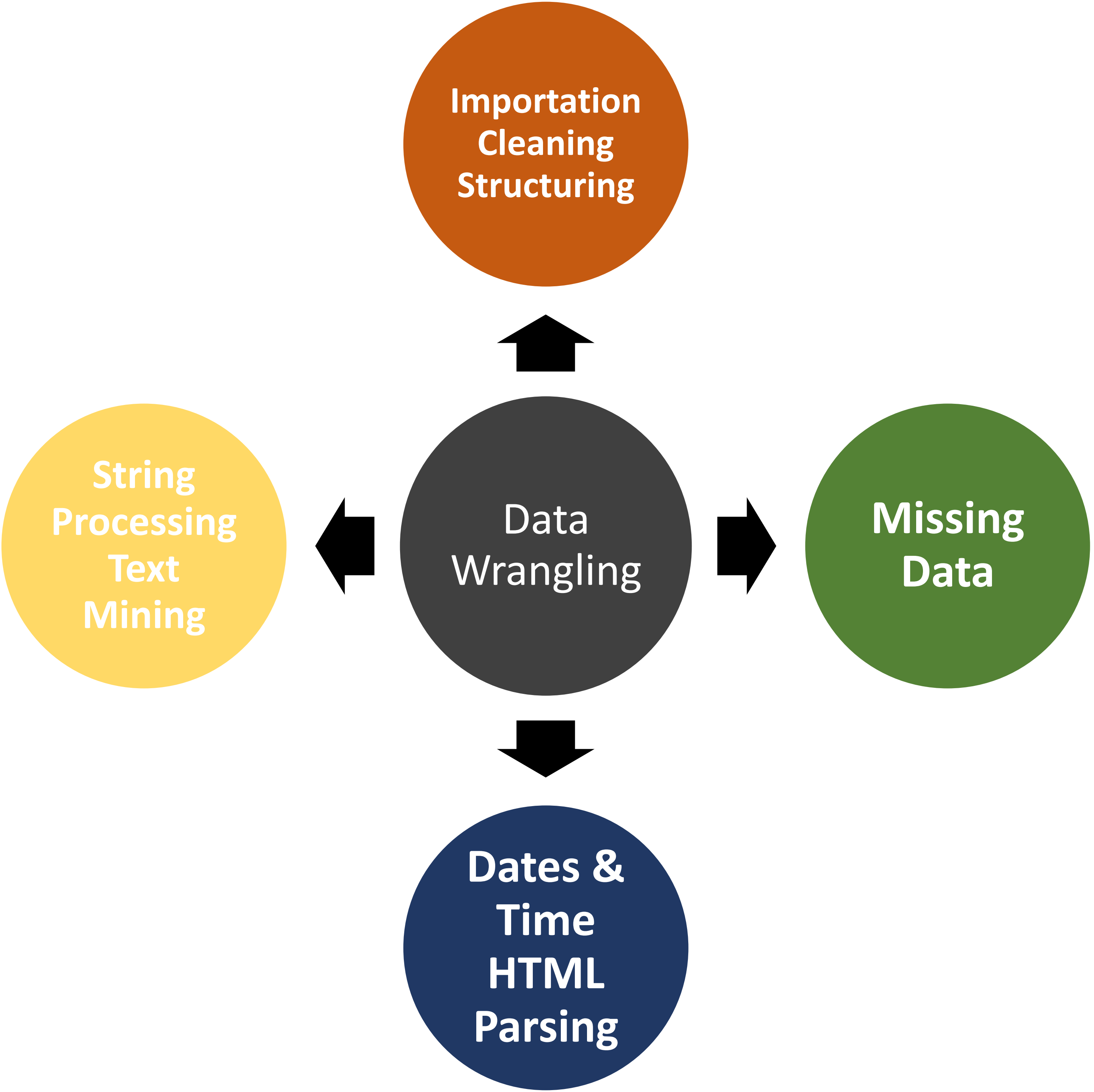
http://tidyr.tidyverse.org/ vignette("tidy-data")


Practice
Practice
Practice

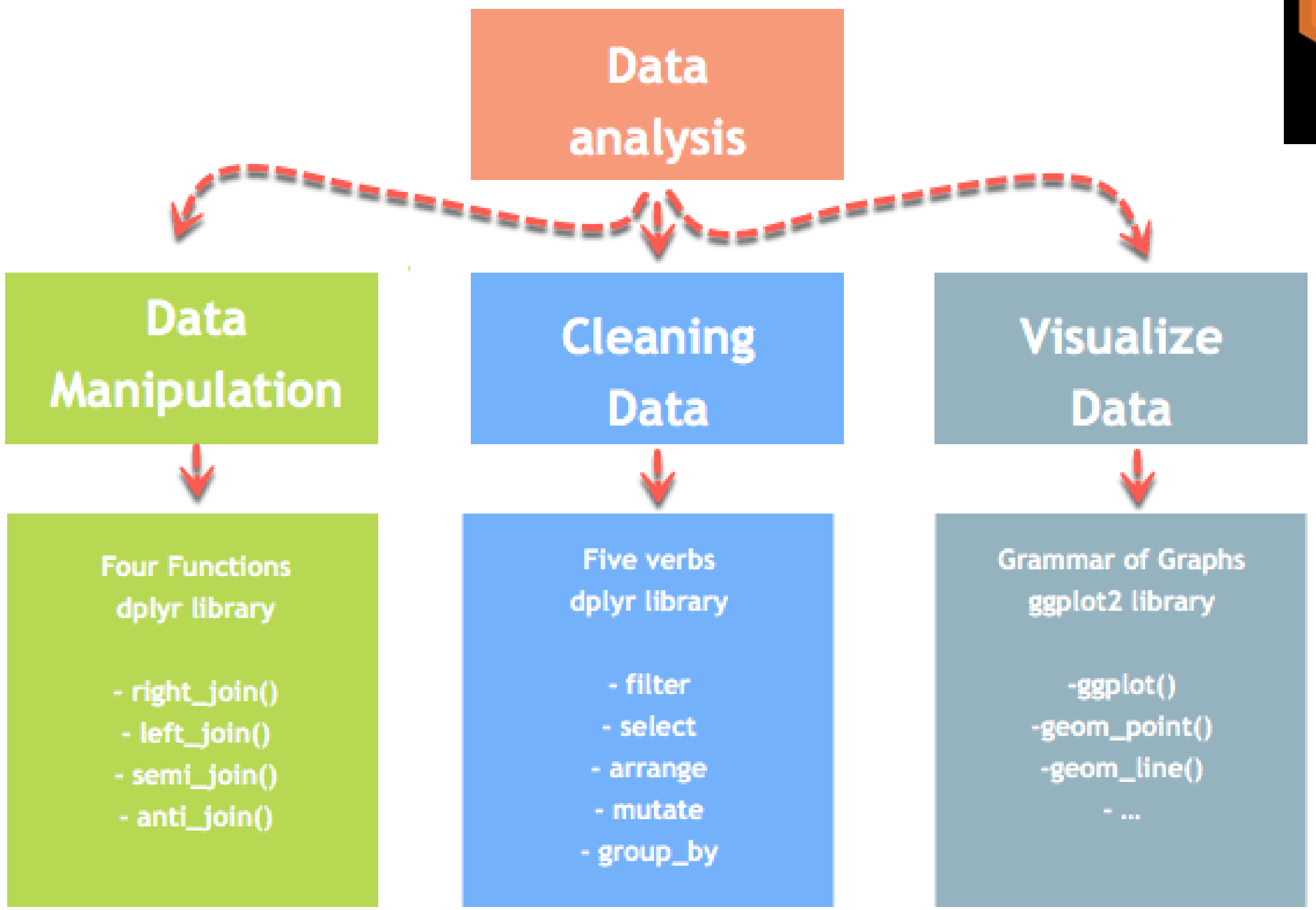
Tidy Data

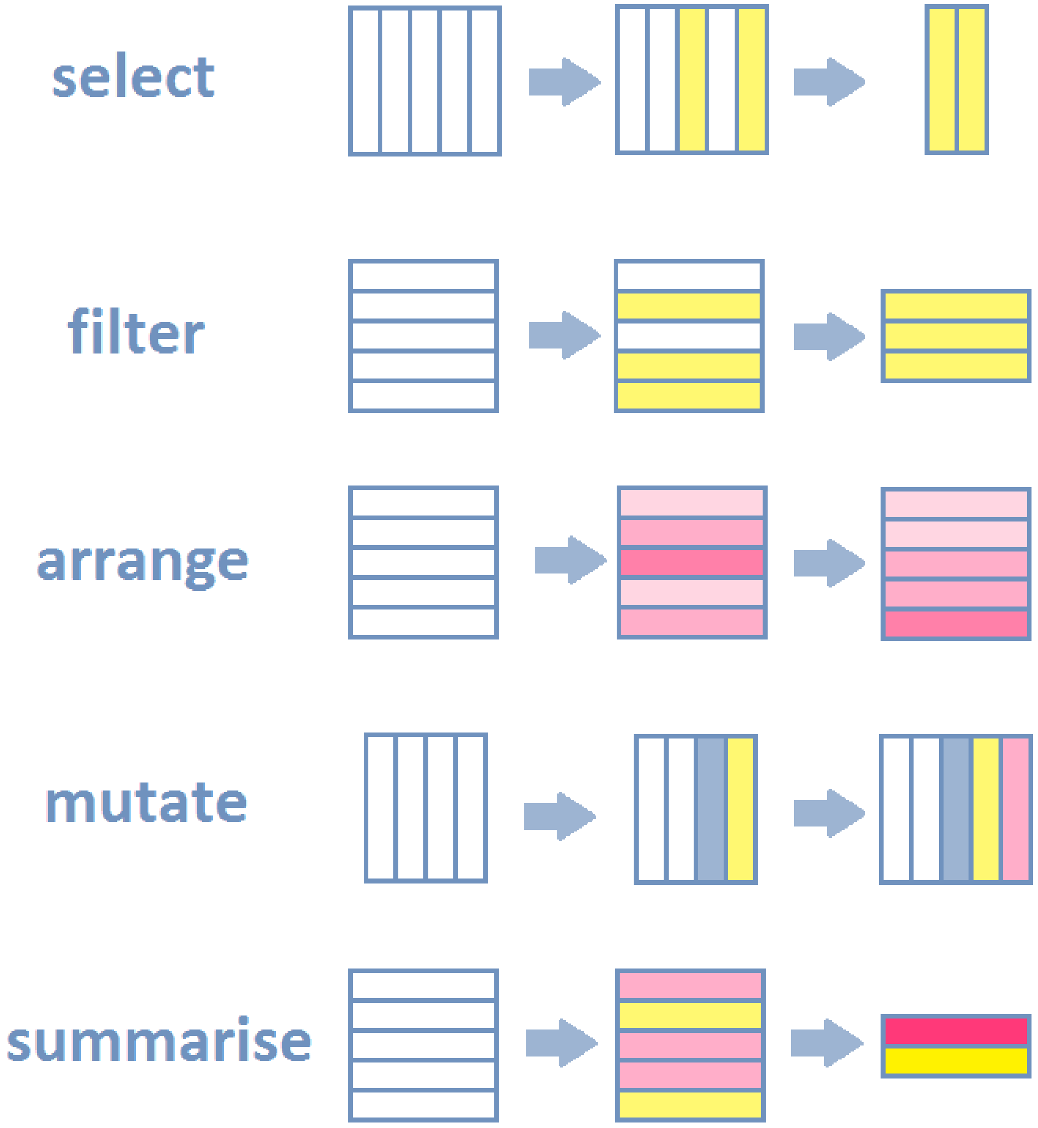
See the paper Tidy Data by Hadley Wickham in Journal of Statistical Software (2014)

https://github.com/rstudio/master-the-tidyverse/archive/master.zip


Importation
Cleaning
Structuring
String
Processing
Text
Mining
Data
Wrangling
Missing
Data
Dates &
Time
HTML
Parsing

Data Wrangling – Dplyr Package





Source: http://perso.ens-lyon.fr/lise.vaudor/dplyr/

- Inspect your tibble (glimpse())
- Select specific columns (select())
- Filter out a subset of rows (filter())
- Reorders rows by one or multiple columns (arrange())
- Change or add columns (mutate())
- Group observations by a grouping variable (group_by())
- Get a summary (in particular per group) (summarise())

Wickham describes functions within dplyr as a set of “verbs” that fall in the broader categories of subsetting, sorting, and transforming

Subsetting data

- `select()` variables
- `filter()` observations

Sorting data

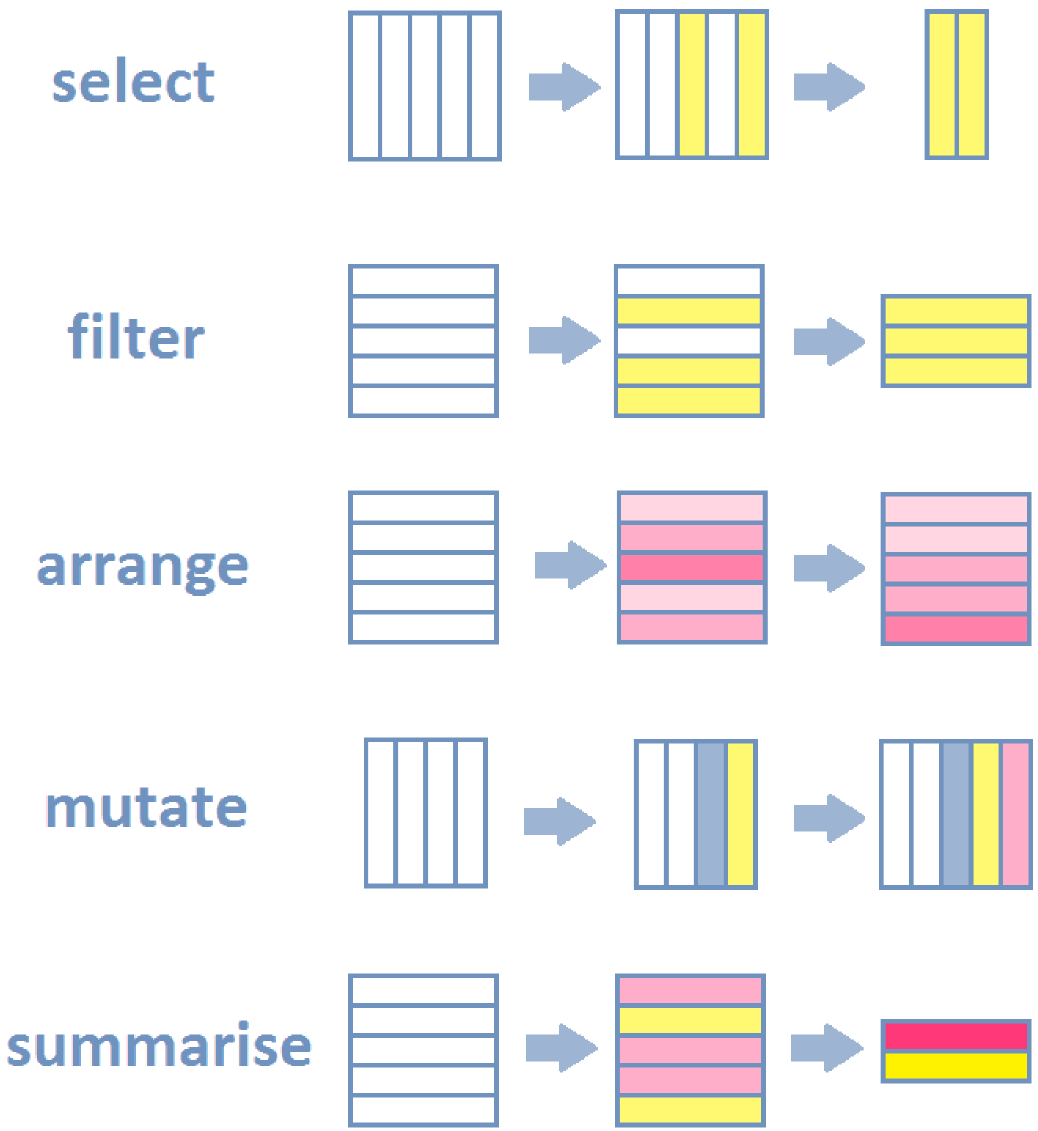
- `arrange()`

Transforming data

- `mutate()` creates new variables
- `summarize()` calculates across rows
- `group_by()` to calculate across rows within groups

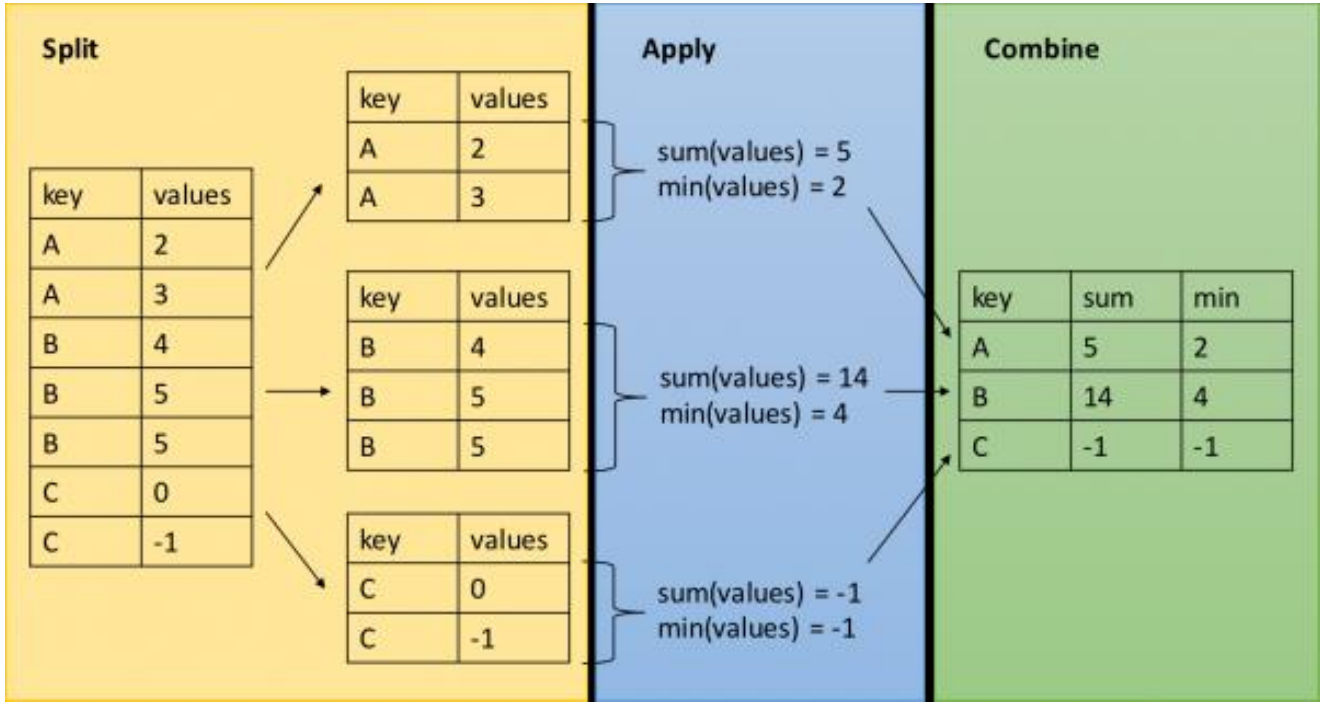
All dplyr verbs (i.e., functions) work as follows

1. first argument is a data frame
2. subsequent arguments describe what to do with variables and observations in data frame
   ▶ refer to variable names without quotes
3. result of the function is a new data frame



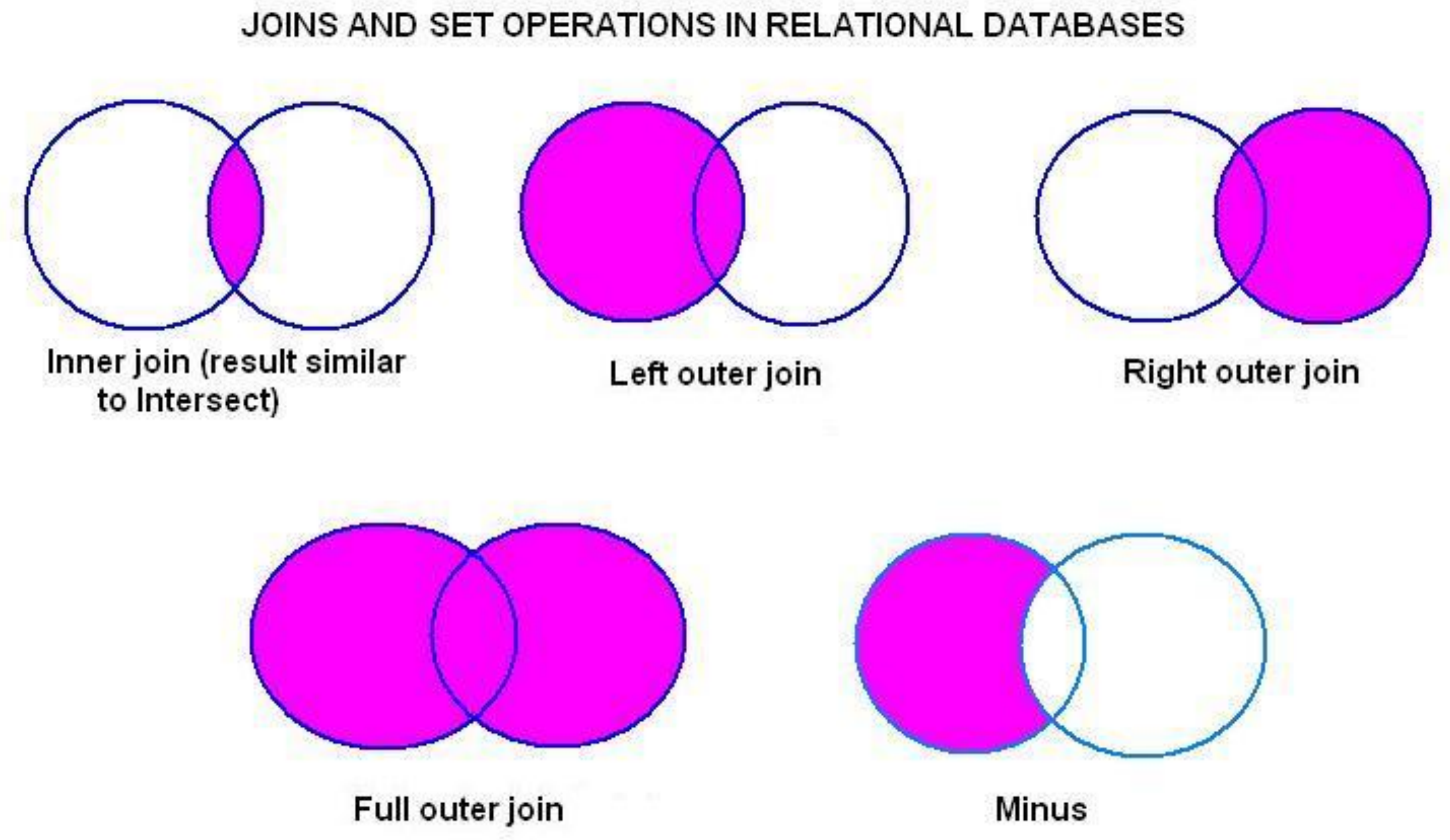
- select(dataframe, column1, column2, …)
- filter(dataframe, logical statement 1, logical statement 2, …)
- arrange(data, variable1, desc(variable2), …)
- mutate(data, newVar1 = expression1, newVar2 = expression2, …)

Source: http://perso.ens-lyon.fr/lise.vaudor/dplyr/

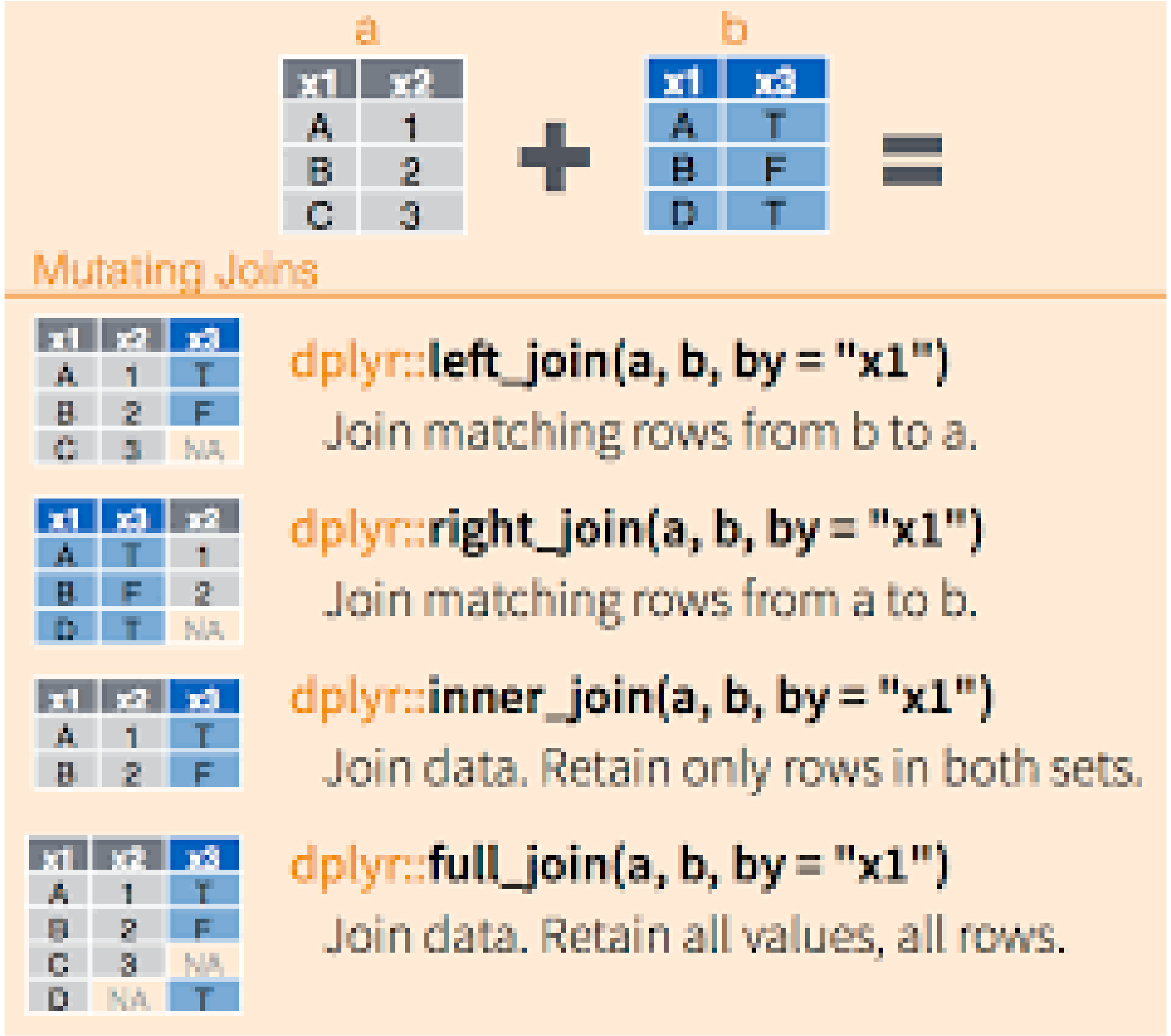


group_by(): group data frame by a factor for downstream commands (usually summarise)

summarise(): summarise values in a data frame or in groups within the data frame with aggregation functions (e.g. min(), max(), mean(), etc…)


JOINS AND SET OPERATIONS IN RELATIONAL DATABASES
Inner join (result similar to Intersect)
Left outer join
Right outer join
Full outer join
Minus

Functions that allow you to join two data frames together.



Magrittr package: using the pipe operator

- Pipe operators provide ways of linking functions together so that the output of a function flows into the input of the next function in the chain.
- Chaining increases readability significantly when there are many commands. With many packages, we can replace the need to perform nested arguments.
- Specify the dataset first, then “pipe” into the next function in the chain.

```
#1.
dlpyr::select (Tb, child:elderly)

# chaining method
Tb %>% dlpyr::select(child:elderly)
```

```
#2.
x1 <- 1:5; x2 <- 2:6
sqrt(sum((x1-x2)^2))

# chaining method
(x1-x2)^2 %>% sum() %>% sqrt()
```


Practice
Practice
Practice