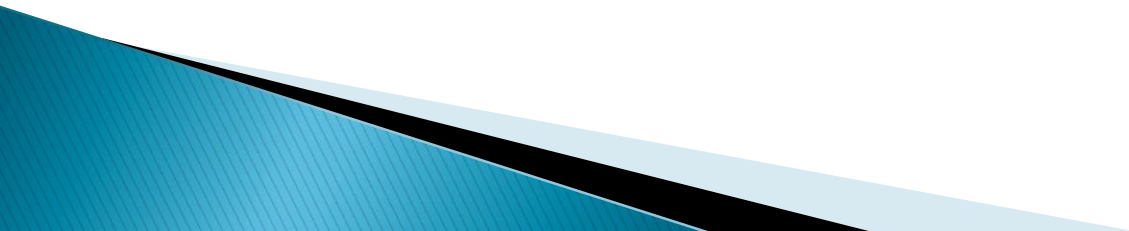


Data Manipulation tidyverse

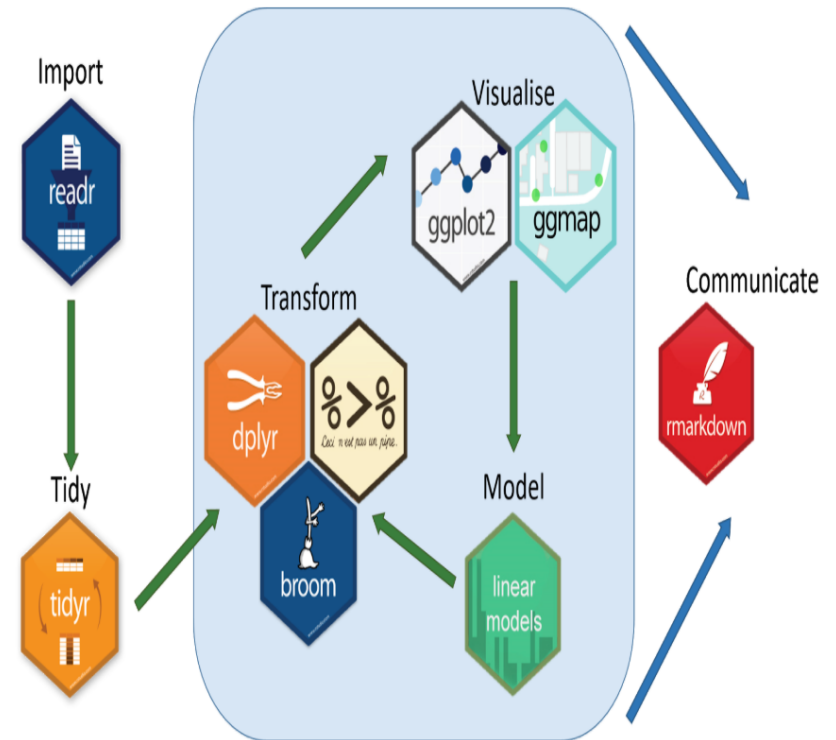
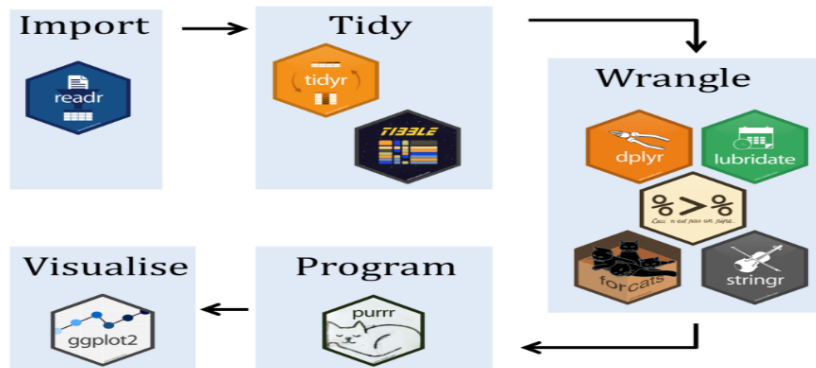


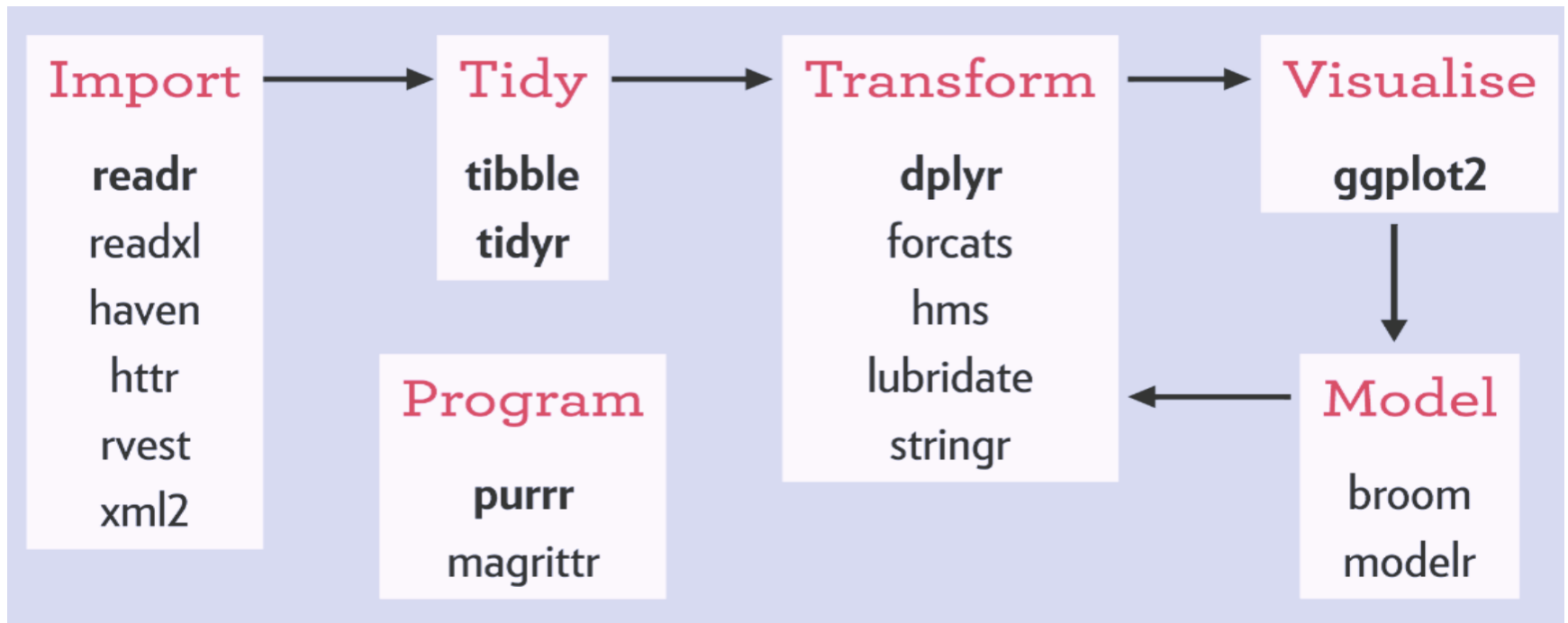
TIDYVERSE

The tidyverse is an opinionated [collection of R packages](#) designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```





Tidyverse is a coherent system of packages for data manipulation, exploration and visualization that share a common design philosophy. These were mostly developed by Hadley Wickham himself, but they are now being expanded by several contributors.

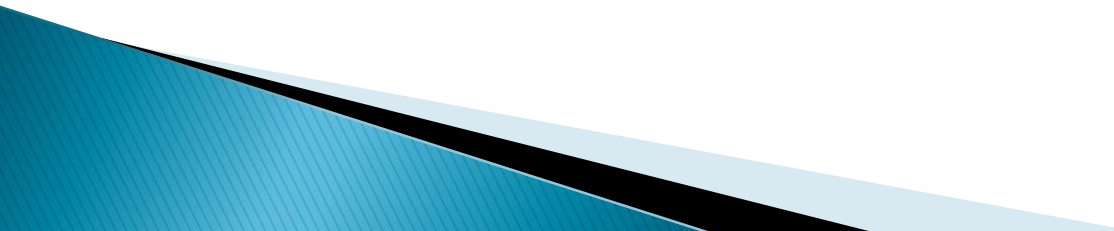
Tidyverse packages are intended to make statisticians and data scientists more productive by guiding them through workflows that facilitate communication, and result in reproducible work products. Fundamentally, the tidyverse is about the connections between the tools that make the workflow possible

Installing all or individual Packages of tidyverse

The easiest way to get tidyr is to install the whole tidyverse:
install.packages("tidyverse")

Alternatively, install just tidyr:
install.packages("tidyr")

Or the the development version from GitHub:
install.packages("devtools")
devtools::install_github("tidyverse/tidyr")



Package – magrittr

- to decrease development time and to improve readability and maintainability of code
- provides a new “pipe”-like operator, `%>%`, with which you may pipe a value forward into an expression or function call; something along the lines of `x %>% f`, rather than `f(x)`. This is not an unknown feature elsewhere;
- it semantically changes your code in a way that makes it more intuitive to both read and write.

Pipe concept makes your code read more like a sentence, branching from left to right.

`f(x)` becomes this: `x %>% f`

`h(g(f(x)))` becomes this: `x %>% f %>% g %>% h`

The “pipe” and is from the magrittr package. [Read about using it here](#)

Sample Code : magrittr

```
library(magrittr)
```

```
car_data <-  
  mtcars %>%  
  subset(hp > 100) %>%  
  aggregate(. ~ cyl, data = ., FUN = . %>% mean %>% round(2)) %>%  
  transform(kpl = mpg %>% multiply_by(0.4251)) %>%  
  print
```

	cyl	mpg	disp	hp	drat	wt	qsec	vs	am	gear	carb	kpl
1	4	25.90	108.0	111.0	3.94	2.15	17.75	1.00	1.00	4.50	2.00	11.010
2	6	19.74	183.3	122.3	3.59	3.12	17.98	0.57	0.43	3.86	3.43	8.391
3	8	15.10	353.1	209.2	3.23	4.00	16.77	0.00	0.14	3.29	3.50	6.419

We start with a value, here `mtcars` (a `data.frame`). Based on this, we first extract a subset, then we aggregate the information based on the number of cylinders, and then we transform the dataset by adding a variable for kilometers per liter as supplement to miles per gallon. Finally we print the result before assigning it. Note how the code is arranged in the logical order of how you think about the task: `data->transform->aggregate`, which is also the same order as the code will execute. It's like a recipe – easy to read, easy to follow!

```
car_data <- transform(aggregate(. ~ cyl, data = subset(mtcars, hp > 100),  
  FUN = function(x) round(mean(x, 2))), kpl = mpg*0.4251)
```

Cluttered Code

Sample Code ...

```
car_data <-  
  mtcars %>%  
  subset(hp > 100) %>%  
  aggregate(. ~ cyl, data = ., FUN = . %>% mean %>% round(2)) %>%  
  transform(kpl = mpg %>% multiply_by(0.4251)) %>%  
  print
```

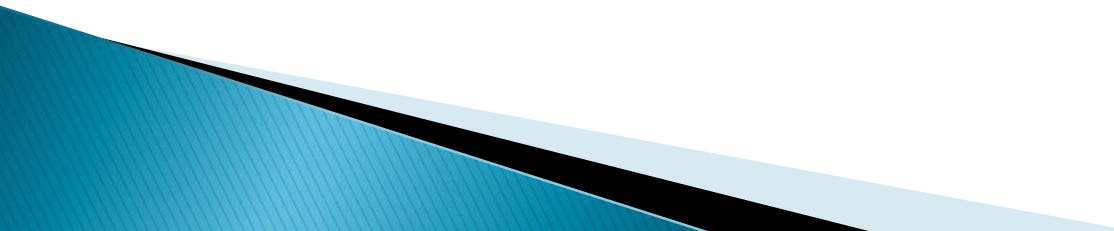
> explanation

- By default the left-hand side (LHS) will be *pipled in* as the first argument of the function appearing on the right-hand side (RHS). This is the case in the subset and transform expressions.
- %>% may be used in a nested fashion, e.g. it may appear in expressions within arguments. This is used in the mpg to kpl conversion.
- When the LHS is needed at a position other than the first, one can use the dot, ' . ', as placeholder. This is used in the aggregate expression.
- The dot in e.g. a formula is *not* confused with a placeholder, which is utilized in the aggregate expression.
- Whenever only *one* argument is needed, the LHS, then one can omit the empty parentheses. This is used in the call to print(which also returns its argument). Here, LHS %>% print(), or even LHS %>% print(.) would also work.
- A pipeline with a dot (.) as LHS will create a unary function. This is used to define the aggregator function.

Tidy Data

“Tidy data” is a term that describes a standardized approach to structuring datasets to make analyses and visualizations easier. If you’ve worked with SQL and relational databases, you’ll recognize most of these concepts. Hadley Wickham took a lot of the technical jargon from Edgar F. Codd’s normal form and applied it to a single data table. More importantly, he translated these principles into terms just about anyone doing data analysis should be able to recognize and understand.

three principles for tidy data:

- Variable make up the columns
 - Observations make up the rows
 - Values go into cells
- 

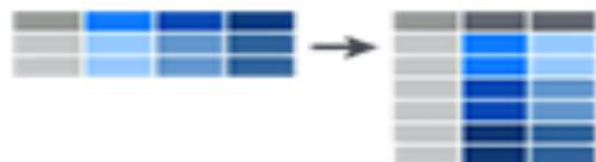
Tidying Data : tidyr

- tidyr is a package from the tidyverse that helps you structure (or restructure) your data so it is easier to visualize and model.
- Tidying a data set usually involves some combination of either converting rows to columns (spreading), or switching the columns to rows (gathering)
 - **gather()** takes multiple columns, and gathers them into key-value pairs: it makes “wide” data longer.
 - **spread()**. takes two columns (key & value) and spreads in to multiple columns, it makes “long” data wider.
- tidyr also provides **separate()** and **extract()** functions which makes it easier to pull apart a column that represents multiple variables. The complement to **separate()** is **unite()**.

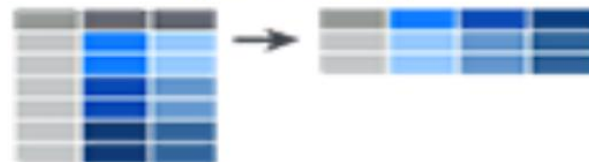
tidyr	gather	spread
reshape(2)	melt	cast
spreadsheets	unpivot	pivot
databases	fold	unfold

Organize Your Data for Easier Analyses in R

gather()



spread()



separate()



unite()



- **gather()**: collapse multiple columns into key-pair values
- **spread()**: reverse of gather. Separate one column into multiple
- **separate()**: separate one column into multiple
- **unite()**: unite multiple columns into one

tidyr R package

Collapses multiple columns into two columns:

1. a **key** column that contains the former column names
2. a **value** column that contains the former column cells

```
gather(cases, "year", "n", 2:4)
```

data frame
to reshape

name of the new
key column
(a character string)

name of the new
value column
(a character string)

names or numeric
indexes of columns
to collapse

gather() and spread()

spread(myData, Element, Temp)

Year	Month	Day	Element	Temp
1	2015	1 1	tmax	78
2	2015	1 1	tmin	72
3	2015	2 2	tmax	82
4	2015	2 2	tmin	74
5	2015	4 4	tmax	81
6	2015	4 4	tmin	71
7	2015	6 3	tmax	80
8	2015	6 3	tmin	71

	Year	Month	Day	tmax	tmin
1	2015	1	1	78	72
2	2015	2	2	82	74
3	2015	4	4	81	71
4	2015	6	3	80	71

```
myData %>% gather("Element", "Temp", -Year, -Month, -Day) %>%
  rename(c(variable = "Element", value = "Temp"))
```

**Cognitive
process:**

1. Take the **ydat** dataset, **then**
2. **filter()** for genes in the leucine biosynthesis pathway, **then**
3. **group_by()** the limiting nutrient, **then**
4. **summarize()** to correlate rate and expression, **then**
5. **mutate()** to round *r* to two digits, **then**
6. **arrange()** by rounded correlation coefficients

**The old
way:**

```
arrange(
  mutate(
    summarize(
      group_by(
        filter(ydat, bp=="leucine biosynthesis"),
        nutrient),
        r=cor(rate, expression)),
    r=round(r, 2)),
  r)
```

**The dplyr
way:**

```
ydat %>%
  filter(bp=="leucine biosynthesis") %>%
  group_by(nutrient) %>%
  summarize(r=cor(rate, expression)) %>%
  mutate(r=round(r,2)) %>%
  arrange(r)
```

Further Readings

<https://www.tidyverse.org/>

<http://r4ds.had.co.nz/>

<http://www.seec.uct.ac.za/r-tidyverse>

<http://www.storybench.org/getting-started-with-tidyverse-in-r/>

Thanks