CT5102: Programming for Data Analytics

Lecture 7: Data Manipulation: tidyr and dplyr

Dr. Jim Duggan,

School of Engineering & Informatics

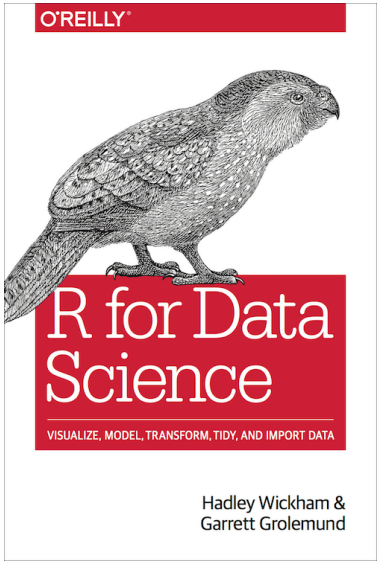
National University of Ireland Galway.

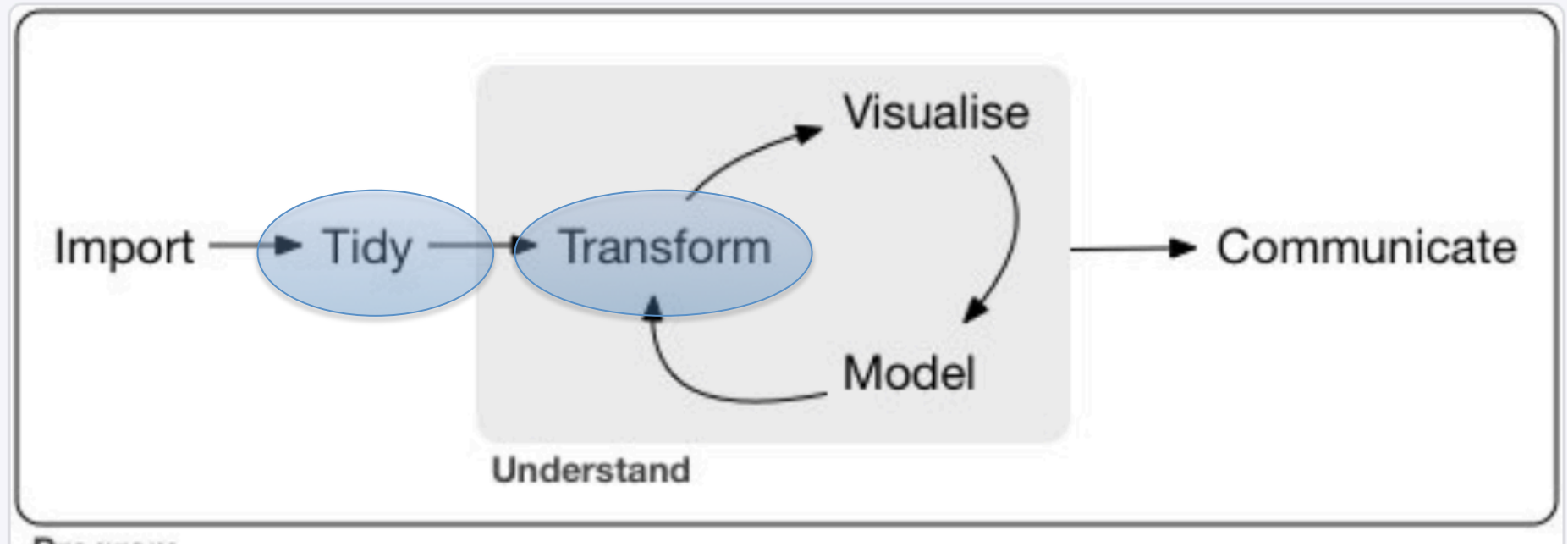
https://github.com/JimDuggan/PDAR

https://twitter.com/_jimduggan

Overview

- Tidy Data, tidyr package
- Data manipulation with dplyr





http://r4ds.had.co.nz

Overview

- What is data tidying?
  - Structuring datasets to facilitate analysis
- The tidy data standard is designed to:
  - Facilitate initial exploration and analysis of data
  - Simplify the development of data analysis tools that work well together
- Principles closely related to relational algebra (Codd 1990)
- Related packages: tidyr, ggplot2, dplyr

Typical Structure: Rows and Columns (Wickham 2014)

	treatmenta	treatmentb
John Smith	—	2
Jane Doe	16	11
Mary Johnson	3	1

Table 1: Typical presentation dataset.

	John Smith	Jane Doe	Mary Johnson
treatmenta	—	16	3
treatmentb	2	11	1

Table 2: The same data as in Table 1 but structured differently.

Numbers refer to the result of the treatments on a given person.

Example in R

```
untidy <- data.frame(
  name = c("John Smith", "Jane Doe", "Mary Johnson"),
  treatmenta = c(NA, 16, 3),
  treatmentb = c(2, 11, 1)
)
```

```
>
> untidy
           name treatmenta treatmentb
1    John Smith         NA          2
2      Jane Doe         16         11
3 Mary Johnson          3          1
```

In a tidy data set…

Variables

- Person (John, Jane, and Mary)
- Treatments (a or b)
- Result (6 values including NA)
- 6 observations

```
>
> untidy
           name treatmenta treatmentb
1    John Smith          NA          2
2      Jane Doe          16         11
3 Mary Johnson           3          1
```

https://rpubs.com/bradleyboehmke/data_wrangling

The goal…

```
> untidy
          name treatmenta treatmentb
1   John Smith          NA           2
2     Jane Doe          16          11
3 Mary Johnson           3           1
```

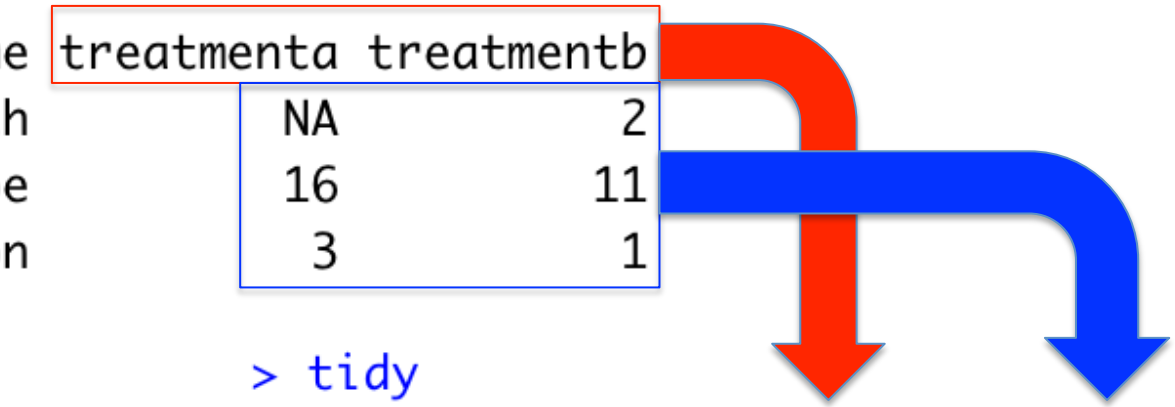
```
> tidy
          name  Treatment Outcome
1   John Smith treatmenta      NA
2     Jane Doe treatmenta      16
3 Mary Johnson treatmenta       3
4   John Smith treatmentb       2
5     Jane Doe treatmentb      11
6 Mary Johnson treatmentb       1
```

tidyr package – four fundamental functions of data tidying

- **gather()** takes multiple columns, and gathers them into key-value pairs: it makes "wide" data longer
- **spread()** takes two columns (key and value) and spreads into multiple columns, it makes long data wider
- **separate()** splits a single column into multiple columns
- **unite()** combines multiple columns into a single column

gather() process

```
>
> untidy
            name treatmenta treatmentb
1   John Smith          NA          2
2     Jane Doe          16         11
3 Mary Johnson           3          1
>
```



```
> tidy
            name  Treatment Outcome
1   John Smith  treatmenta      NA
2     Jane Doe  treatmenta      16
3 Mary Johnson  treatmenta       3
4   John Smith  treatmentb       2
5     Jane Doe  treatmentb      11
6 Mary Johnson  treatmentb       1
```

gather()

https://rpubs.com/bradleyboehmke/data_wrangling

```
Function:       gather(data, key, value, ..., na.rm = FALSE, convert = FALSE)
Same as:        data %>% gather(key, value, ..., na.rm = FALSE, convert = FALSE)

Arguments:
        data:           data frame
        key:            column name representing new variable
        value:          column name representing variable values
        ...:            names of columns to gather (or not gather)
        na.rm:          option to remove observations with missing values (represented by NAs)
        convert:        if TRUE will automatically convert values to logical, integer, numeric, complex or
                        factor as appropriate
```

```
> tidy <- gather(untidy,key=Treatment,value=Outcome,treatmenta:treatmentb)
>
> tidy
          name  Treatment Outcome
1   John Smith treatmenta      NA
2     Jane Doe treatmenta      16
3 Mary Johnson treatmenta       3
4   John Smith treatmentb       2
5     Jane Doe treatmentb      11
6 Mary Johnson treatmentb       1
```

spread()

https://rpubs.com/bradleyboehmke/data_wrangling

```
Function:       spread(data, key, value, fill = NA, convert = FALSE)
Same as:        data %>% spread(key, value, fill = NA, convert = FALSE)

Arguments:
        data:           data frame
        key:            column values to convert to multiple columns
        value:          single column values to convert to multiple columns' values
        fill:           If there isn't a value for every combination of the other variables and the key
                        column, this value will be substituted
        convert:        if TRUE will automatically convert values to logical, integer, numeric, complex or
                        factor as appropriate
```

```
> spread(tidy,Treatment,Outcome)
            name treatmenta treatmentb
1       Jane Doe         16         11
2     John Smith         NA          2
3 Mary Johnson          3          1
```

separate()

https://rpubs.com/bradleyboehmke/data_wrangling

```
Function:       separate(data, col, into, sep = " ", remove = TRUE, convert = FALSE)
Same as:        data %>% separate(col, into, sep = " ", remove = TRUE, convert = FALSE)

Arguments:
        data:           data frame
        col:            column name representing current variable
        into:           names of variables representing new variables
        sep:            how to separate current variable (char, num, or symbol)
        remove:         if TRUE, remove input column from output data frame
        convert:        if TRUE will automatically convert values to logical, integer, numeric, complex or
                        factor as appropriate
```

```
> unsep
      Date Product Sales
1 Jan_2005     ABC  1000
2 Jan_2006     DEF  2000
3 Jan_2007     ABC  3000
```

```
> separate(unsep,Date,c("Year","Month"))
  Year Month Product Sales
1  Jan  2005     ABC  1000
2  Jan  2006     DEF  2000
3  Jan  2007     ABC  3000
```

unite()

https://rpubs.com/bradleyboehmke/data_wrangling

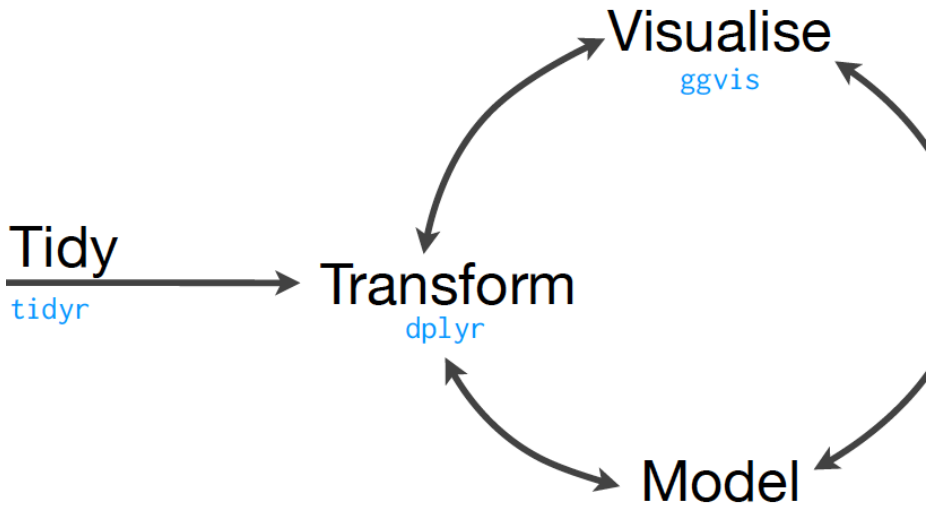
```
Function:       unite(data, col, ..., sep = " ", remove = TRUE)
Same as:        data %>% unite(col, ..., sep = " ", remove = TRUE)

Arguments:
        data:           data frame
        col:            column name of new "merged" column
        ...:            names of columns to merge
        sep:            separator to use between merged values
        remove:         if TRUE, remove input column from output data frame
```

```
> df
  Year Month Product Sales
1  Jan  2005     ABC  1000
2  Jan  2006     DEF  2000
3  Jan  2007     ABC  3000
```

```
> unite(df,Year,Year, Month,sep=".")
       Year Product Sales
1 Jan.2005     ABC  1000
2 Jan.2006     DEF  2000
3 Jan.2007     ABC  3000
```

Analysing Tidy Data

- Tidy data is only worthwhile if it makes analysis easier
- Tidy tools
  - Take tidy data sets as inputs and return tidy data sets as outputs
- Tools cover:
  - Data manipulation
  - Visualisation
  - Modelling



https://www.dropbox.com/sh/i8qnluwmuieicxc/AACsepZJvULCKkbIxK9KP-6Ea/dplyr-tutorial.pdf?dl=0

References…

- https://rpubs.com/justmarkham/dplyr-tutorial
- https://www.youtube.com/watch?v=8SGif63VW6E
- https://www.youtube.com/watch?v=Ue08LVuk790
- http://renkun.me/pipeR-tutorial/Examples/dplyr.html

Using dplyr

- Functions
  - **filter**: keep rows matching criteria
  - **select**: pick columns by name
  - **arrange**: reorder rows
  - **mutate**: add new variables
  - **summarise**: reduce variables to values
- Approach
  - First argument is a data frame
  - Subsequent arguments say what to do with data frame
  - Always returns a data frame (Tidy Data approach)

Example

```
df <- data.frame(
  color = c("blue", "black", "blue", "blue", "black"),
  value = 1:5)
```

color	value
blue	1
black	2
blue	3
blue	4
black	5

1. filter()
Keep rows by matching criteria

```
>
> filter(df, color == "blue")
  color value
1  blue     1
2  blue     3
3  blue     4
```

color	value
blue	1
blue	3
blue	4

```
> filter(df, value %in% c(1, 4))
  color value
1  blue     1
2  blue     4
```

color	value
blue	1
blue	4

2. select()
pick columns by name

```
> select(df,color)
  color
1  blue
2 black
3  blue
4  blue
5 black
```

color
blue
black
blue
blue
black

```
> select(df,-color)
  value
1     1
2     2
3     3
4     4
5     5
```

value
1
2
3
4
5

Special functions with select()

Special functions

As well as using existing functions like `:` and `c`, there are a number of special functions that only work inside `select`

- `starts_with(x, ignore.case = TRUE)`: names starts with `x`
- `ends_with(x, ignore.case = TRUE)`: names ends in `x`
- `contains(x, ignore.case = TRUE)`: selects all variables whose name contains `x`
- `matches(x, ignore.case = TRUE)`: selects all variables whose name matches the regular expression `x`
- `num_range("x", 1:5, width = 2)`: selects all variables (numerically) from x01 to x05.
- `one_of("x", "y", "z")`: selects variables provided in a character vector.
- `everything()`: selects all variables.

3. arrange()
reorder rows

color	value
blue	1
black	2
blue	3
blue	4
black	5

```
> arrange(df, color)
  color value
1 black     2
2 black     5
3  blue     1
4  blue     3
5  blue     4
```

```
> arrange(df, value)
  color value
1  blue     1
2 black     2
3  blue     3
4  blue     4
5 black     5
```

3. arrange()
reorder rows

color	value
blue	1
black	2
blue	3
blue	4
black	5

```
> arrange(df, desc(color))
  color value
1  blue     1
2  blue     3
3  blue     4
4 black     2
5 black     5
```

```
> arrange(df, desc(value))
  color value
1 black     5
2  blue     4
3  blue     3
4 black     2
5  blue     1
```

4. mutate()
Add new variables

color	value
blue	1
black	2
blue	3
blue	4
black	5

color	value	double
blue	1	2
black	2	4
blue	3	6
blue	4	8
black	5	10

```
> mutate(df, double = 2 * value)
  color value double
1  blue     1      2
2 black     2      4
3  blue     3      6
4  blue     4      8
5 black     5     10
```

5. summarise()
Reduce variables to values

color	value
blue	1
black	2
blue	3
blue	4
black	5

Total
15

```
> summarise(df, Total = sum(value))
  Total
1    15
```

group_by() function

color	value
blue	1
black	2
blue	3
blue	4
black	5

color	total
blue	8
black	7

```
> by_color <- group_by(df, color)
>
> summarise(by_color, total = sum(value))
# A tibble: 2 x 2
   color total
  <fctr> <int>
1  black     7
2   blue     8
```

Other summary functions…

color	value
blue	1
black	2
blue	3
blue	4
black	5

```
summarise(by_color,
          Mean = mean(value),
          Median=median(value),
          Distinct = n_distinct(value))
```

```
# A tibble: 2 x 4
   color     Mean Median Distinct
  <fctr>    <dbl>  <dbl>    <int>
1  black 3.500000    3.5        2
2   blue 2.666667    3.0        3
```

Summary Functions

- min(x), median(x), max(x),
- quantile(x, p)
- n(), n_distinct(), sum(x), mean(x)
- sum(x > 10), mean(x > 10)
- sd(x), var(x), iqr(x), mad(x)

Data Pipelines in R using %>%

- Pipe operator from magrittr
- x %>% f(y) → f(x, y)

```
> library(magrittr)
>
>
> 1:10 %>% sqrt()
 [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427 3.000000
[10] 3.162278
```

Example

```
> df
  color value
1  blue     1
2 black     2
3  blue     3
4  blue     4
5 black     5
>
> df1 <- mutate(df,double = 2 * value)
> by_color <- group_by(df1, color)
> summarise(by_color,total=sum(double))
# A tibble: 2 x 2
   color total
  <fctr> <dbl>
1  black    14
2   blue    16
```

```
> df
  color value
1  blue     1
2 black     2
3  blue     3
4  blue     4
5 black     5
>
> df %>%
+   mutate(double = 2 * value) %>%
+   group_by(color)            %>%
+   summarise(total=sum(double))
# A tibble: 2 x 2
   color total
  <fctr> <dbl>
1  black    14
2   blue    16
```

Joining Tables x and y in dplyr

Type	Action
inner	Include only rows in **both** x and y
left	Include all of x, and matching rows of y
semi	Include rows of x that match y
anti	Include rows of x that **don't** match y

Example

name	instrument
John	guitar
Paul	bass
George	guitar
Ringo	drums
Stuart	bass
Pete	drums

name	band
John	T
Paul	T
George	T
Ringo	T
Brian	F

```
x <- data.frame(
  name = c("John", "Paul", "George", "Ringo", "Stuart", "Pete"),
  instrument = c("guitar","bass","guitar","drums","bass","drums"),
  stringsAsFactors = F
)
```

```
y <- data.frame(
  name = c("John", "Paul", "George", "Ringo", "Brian"),
  band = c(T,T,T,T,F),
  stringsAsFactors = F
)
```

Type	Action
inner	Include only rows in **both** x and y

name	instrument
John	guitar
Paul	bass
George	guitar
Ringo	drums
Stuart	bass
Pete	drums

name	band
John	T
Paul	T
George	T
Ringo	T
Brian	F

```
> inner_join(x,y)
Joining, by = "name"
    name instrument band
1   John     guitar TRUE
2   Paul       bass TRUE
3 George     guitar TRUE
4  Ringo      drums TRUE
```

Type	Action
left	Include all of x, and matching rows of y

name	instrument
John	guitar
Paul	bass
George	guitar
Ringo	drums
Stuart	bass
Pete	drums

name	band
John	T
Paul	T
George	T
Ringo	T
Brian	F

```
> left_join(x,y)
Joining, by = "name"
    name instrument band
1   John     guitar TRUE
2   Paul       bass TRUE
3 George     guitar TRUE
4  Ringo      drums TRUE
5 Stuart       bass   NA
6   Pete      drums   NA
```

Type	Action
semi	Include rows of x that match y

name	instrument
John	guitar
Paul	bass
George	guitar
Ringo	drums
Stuart	bass
Pete	drums

name	band
John	T
Paul	T
George	T
Ringo	T
Brian	F

```
>
> semi_join(x,y)
Joining, by = "name"
    name instrument
1   John     guitar
2   Paul       bass
3 George     guitar
4  Ringo      drums
```

Type	Action
anti	Include rows of x that **don't** match y

name	instrument
John	guitar
Paul	bass
George	guitar
Ringo	drums
Stuart	bass
Pete	drums

name	band
John	T
Paul	T
George	T
Ringo	T
Brian	F

```
> anti_join(x,y)
Joining, by = "name"
    name instrument
1   Pete      drums
2 Stuart       bass
```

References

- Wickham, H. 2015. Advanced R. Taylor & Francis