Week 5 Demonstration

Tidy & Manipulate: Part II - Manipulate

Dr. Anil Dolgun

28/03/2018

Course Announcements

- Marking and feedback for Assignment 1 will be finalised by the next week.

Course Announcements

- Marking and feedback for Assignment 1 will be finalised by the next week.
- Suggested solutions to Week 4 Worksheet are available here

Course Announcements

- Marking and feedback for Assignment 1 will be finalised by the next week.
- Suggested solutions to Week 4 Worksheet are available here
- Details for Assignment 2 will be released at the end of this week.

Course Announcements

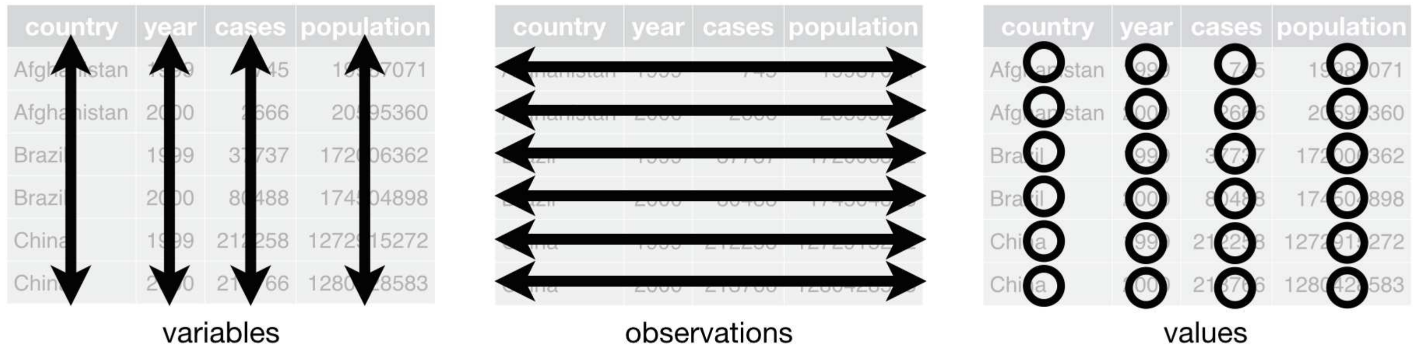
- Marking and feedback for Assignment 1 will be finalised by the next week.
- Suggested solutions to Week 4 Worksheet are available here
- Details for Assignment 2 will be released at the end of this week.
- Any other questions or concerns?

Tidy & Manipulate: Part II - Manipulate

Recall: Tidy Data Principles

Previous week:

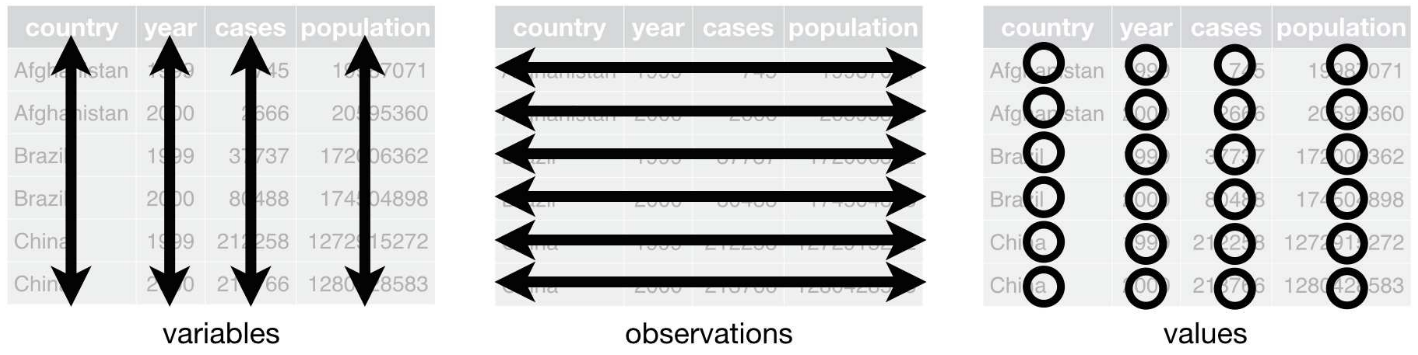
- The framework of " " provides a standard and consistent way of storing data that makes transformation, visualization, and modeling easier.



Recall: Tidy Data Principles

Previous week:

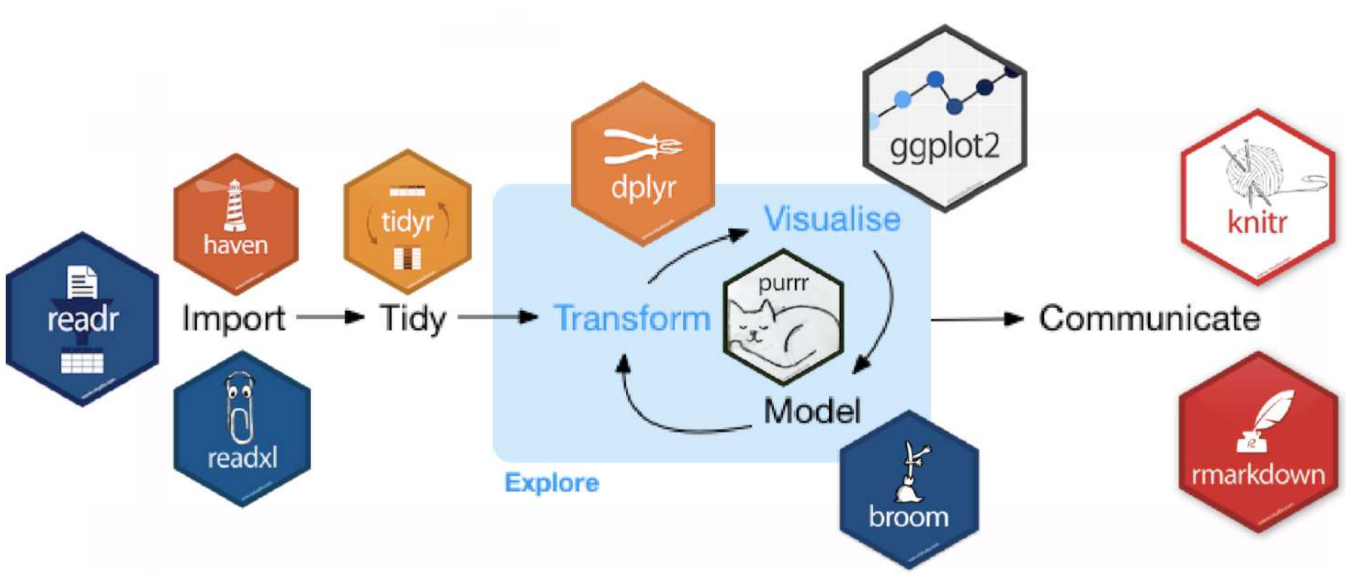
- The framework of "   " provides a standard and consistent way of storing data that makes transformation, visualization, and modeling easier.



- Each variable must have its own column.
- Each observation must have its own row.
- Each value must have its own cell.

Recall: `tidyr`

- `tidyr` is a one such package which was built for the sole purpose of simplifying the process of creating tidy data.
- Following tidy principles makes manipulation, transformation, visualization, and modeling easier.
- is a set of packages that work in harmony because they share common data representations and API design



5 / 30

The grammar of Data Manipulation: dplyr

- There are many data manipulation packages/functions in R,
- Most of them lack consistent coding and the ability to easily flow together.
- This leads to difficult-to-read nested functions and/or choppy code.

The grammar of Data Manipulation: dplyr

- There are many data manipulation packages/functions in R,
- Most of them lack consistent coding and the ability to easily flow together.
- This leads to difficult-to-read nested functions and/or choppy code.
- The `dplyr` package is regarded as the " " in R.
- It provides a consistent set of verbs that help you solve the most common data manipulation challenges.
- Remember: `dplyr` functions work with pipes `%>%` and expect .

The grammar of Data Manipulation: dplyr

- There are six fundamental functions of data manipulation that `dplyr` provides:
    - **`select()`** pick/select variables
    - **`filter()`** pick/filter observations based on values
    - **`arrange()`** sort variables
    - **`mutate()`** create new variables
    - **`summarise()`** summarise data by functions of choice
    - **`group_by()`** + **`summarise()`**
- There are also functions to join and merge data sets:
    -
    -
    -
    -
- The full list of capabilities can be found in the dplyr reference manual.
- I highly recommend going through it as there are many great functions provided by `dplyr` that I will not cover here.

select() : select variables

- Often we only assess specific variables. The `select()` function allows us to select variables.
- In addition to the existing functions like `:` and `c()`, there are a number of special functions that can work inside select.

`-`	Select everything but
`:`	Select range
`contains()`	Select columns whose name contains a character string
`ends_with()`	Select columns whose name ends with a string
`everything()`	Select every column
`matches()`	Select columns whose name matches a regular expression
`num_range()`	Select columns named x1, x2, x3, x4, x5
`one_of()`	Select columns whose names are in a group of names
`starts_with()`	Select columns whose name starts with a character string

Class Activity:

- Visit https://b.socrative.com/login/student/
- Room Name: MATH2349
- Enter a nickname
- Work in small groups
- Import the CustomerData.csv data set and answer the questions on Socrative.
- No need to type any commands in R!

Import Customer Data

- The CustomerData.csv data set includes some characteristics of 5000 customers. Header of this data set is as follows.

```
customer <- read_csv("../data/CustomerData.csv")
head(customer[, 1:7], 5)
```

```
## # A tibble: 5 x 7
##   CustomerID      Region TownSize Gender   Age EducationYears JobCategory
##   <chr>            <int> <chr>    <chr>  <int>          <int> <chr>
## 1 3964-QJWTRG-NPN      1 2        Female    20             15 Professional
## 2 0648-AIPJSP-UVM      5 5        Male      22             17 Sales
## 3 5195-TLUDJE-HVO      3 4        Female    67             14 Sales
## 4 4459-VLPQUH-3OL      4 3        Male      23             16 Sales
## 5 8158-SMTQFB-CNO      2 2        Male      26             16 Sales
```

Class Activity: select()

Which of the following can be used to:

- Q1. select all variables between `CustomerID` and `Gender`.
- Q2. select all variables other than those between `CustomerID` and `Gender`.
- Q3. select CustomerID and all variables that contain the word "Card".

filter(): filter observations based on values

- `filter()` identifies or selects observations in which a particular variable matches a specific value/condition.
- The condition(s) can be any kind of logical comparison and Boolean operators, such as:

`<`	Less than
`>`	Greater than
`==`	Equal to
`<=`	Less than or equal to
`>=`	Greater than or equal to
`!=`	Not equal to
`%in%`	Group membership
`is.na`	Is NA

`!is.na`	Is not NA
`&, I`	Boolean AND, OR
`xor`	exactly or
`!`	not
`any`	any true
`all`	all true

Class Activity: filter()

Which of the following can be used to:

- Q4. filter for female customers only?
- Q5. filter for female customers that are greater than 45 years old AND live in region 3.
- Q6. filter for female customers that are greater than 45 years old OR live in region 3.

arrange(): order data by variables

- `arrange()` orders the data by variables in ascending (default) or descending order.
- For a descending order, use `desc()` within the `arrange()` function.

Class Activity: arrange()

Which of the following can be used to:

- Q7: select the variables `CustomerID`, `Region`, `Gender`, `Age`, `HHIncome`, `Cardspend` and save this as `sub_cust`.
- Q8: order `sub_cust` data by `Age` and `CardSpendMonth` (ascending order)
- Q9: order `sub_cust` data by `Age` (oldest to youngest) and `CardSpendMonth` (least to most)

mutate(): create new variables

- `mutate()` adds new variables while preserving the existing variables.
- `transmute()` creates a new variable and then drops the other variables.
- Here is the list of some useful functions used inside the `mutate()`.

`pmin()`, `pmax()`	Element wise min and max
`cummin()`, `cummax()`	Cumulative min and max
`cumsum()`, `cumprod()`	Cumulative sum and product
`between()`	Are values between a and b?
`cume_dist()`	Cumulative distribution of values
`cumall()`, `cumany()`	Cumulative all and any
`cummean()`	Cumulative mean

Class Activity: mutate()

Which of the following can be used to:

- Q10: create a ratio variable that computes the ratio of `CardSpendMonth` to `HHIncome` using `sub_cust` data
- Q11: create two variables: ratio1 = CardSpendMonth / HHIncome and ratio2 = CardSpendMonth / Age

summarise(): summarise data by functions of choice

- `summarise()` (or `summarize()`) performs the majority of summary statistics.

Functions	Usage
`min()`, `max()`	Minimum and maximum values
`mean()`	Mean value
`median()`	Median value
`sum()`	Sum of values
`var()`, `sd()`	Variance and standard deviation of a vector
`first()`	First value in a vector
`last()`	Last value in a vector
`nth()`	Nth value in a vector
`n()`	The number of values in a vector
`n_distinct()`	The number of distinct values in a vector

- All functions in this list takes a vector of values and returns a single summary value.

group_by() + summarise() function

- If we want to take the summary statistics grouped by a variable, then we need to use another function called `group_by()`.
- `group_by()` along with `summarise()` functions will allow us to take and compare summary statistics grouped by a variable.

Class Activity: group_by() + summarise()

Which of the following can be used to:

- Q12: compute the average `CardSpendMonth` across all customers in our sub_cust data.
- Q13: compute the average `CardSpendMonth` for each `Gender`.
- Q14: compute the average `CardSpendMonth` for each `Gender` and `Region`.

Joining data sets

- Often we have separate data frames that can have common and differing variables for similar observations (relational data sets).

```
band_members
```

```
## # A tibble: 3 x 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

```
band_instruments
```

```
## # A tibble: 3 x 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```

```
band_instruments2
```

```
## # A tibble: 3 x 2
##   artist plays
##   <chr>  <chr>
## 1 John   guitar
## 2 Paul   bass
## 3 Keith  guitar
```

Joining data sets

- `dplyr` offers three sets of joining functions to provide alternative ways to join data frames.
  - Mutating joins: add new variables to one data frame from matching observations in another.
  - Filtering joins: filter observations from one data frame based on whether or not they match an observation in the other table.
  - Set operations: treat observations as if they were set elements.
  - Merging data sets: merge data frames by row and column
- Please refer to the cheatsheets under the drive folder for details.

Mutating joins

```
- `left_join()`: prioritizes left dataset
- `right_join()`: prioritizes right dataset
- `inner_join()`: only retains rows in both datasets
- `full_join()`: retains all rows
```

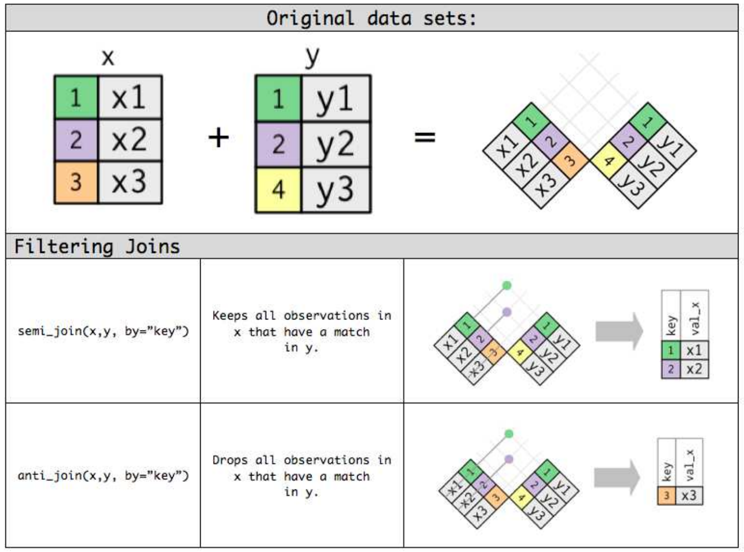
- Each mutating join takes an argument `by` that controls which variables are used to match observations in the two data sets.
  - `NULL`: The default value. `dplyr` will will use all variables that appear in both tables, a natural join.
  - A character vector, `by = "x"`

Example: Mutating joins

```
library(dplyr)

band_members
band_instruments
band_instruments2

# "Mutating" joins add variables to the LHS

band_members %>% inner_join(band_instruments)
band_members %>% left_join(band_instruments)
band_members %>% right_join(band_instruments)
band_members %>% full_join(band_instruments)
```

Filtering joins

```
- `semi_join(x, y)`: keeps all observations in x that have a match in y.
- `anti_join(x, y)`: drops all observations in x that have a match in y.
```



semi_join(x,y, by="key")	Keeps all observations in x that have a match in y.	
anti_join(x,y, by="key")	Drops all observations in x that have a match in y.	

Adapted from Wickham, Hadley, and Garrett Grolemund. 2016. R for Data Science: Import, Tidy, Transform, Visualize, and Model Data. O'Reilly Media, Inc.

Example: Filtering joins

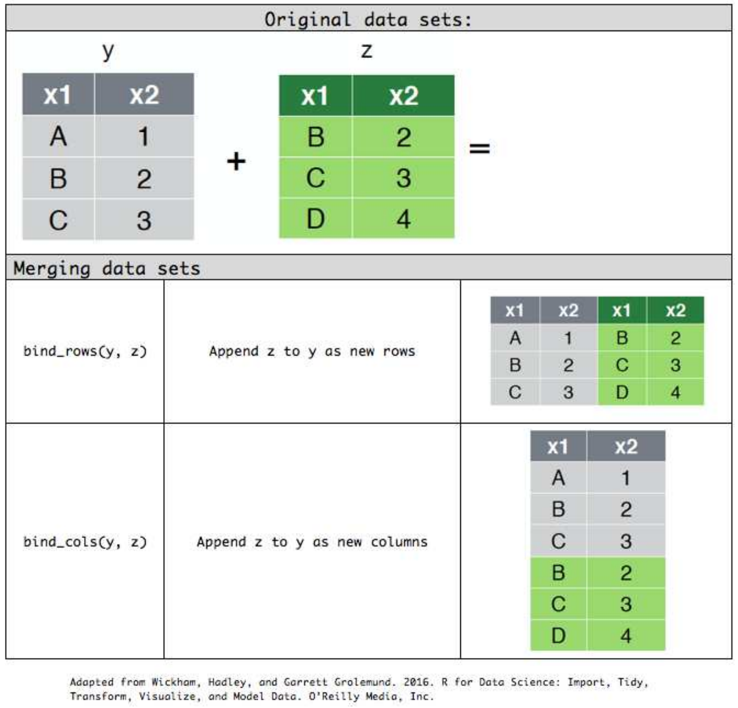
```
# "Filtering" joins keep cases from the LHS

band_members %>% semi_join(band_instruments)
band_members %>% anti_join(band_instruments)
```

Set operations

```
- `intersect(x, y)`: return only observations in both x and y.
- `union(x, y)`: return unique observations in x and y.
- `setdiff(x, y)`: return observations in x, but not in y.
```

Merging data sets

```
- `bind_rows(x, y)`: Append y to x as new rows.
- `bind_cols(x, y)`: Append y to x as new columns.
```



Original data sets:

y

x1	x2
A	1
B	2
C	3

+

z

x1	x2
B	2
C	3
D	4

=

Merging data sets

bind_rows(y, z)	Append z to y as new rows	x1 x2 x1 x2 / A 1 B 2 / B 2 C 3 / C 3 D 4
bind_cols(y, z)	Append z to y as new columns	x1 x2 / A 1 / B 2 / C 3 / B 2 / C 3 / D 4

Adapted from Wickham, Hadley, and Garrett Grolemund. 2016. R for Data Science: Import, Tidy, Transform, Visualize, and Model Data. O'Reilly Media, Inc.

Functions to Remember for Week 5 (dplyr)

Operator/Function	Description
`filter`	pick observations based on their values
`>, >=, <, <=, !=, ==`	comparison operators
`arrange`	re-order rows
`desc`	order in descending order
`select`	select variables
`starts_with, ends_with, contains, etc.`	select variables based on patterns
`mutate, transmute`	create new variables
`summarise`	summarize data
`group_by`	group based on categorical variables
`%>%`	pipe operator to chain together functions

- Practice!

Class Worksheet

- Working in small groups, complete the following class worksheet

Week 5 Class Worksheet

- Once completed, feel free to work on your Assignment and/or Skill Builders

Return to Data Preprocessing Website