Data Wrangling in the Tidyverse

21st Century R

DS Portugal Meetup, at Farfetch, Porto, Portugal
April 19, 2017

Jim Porzak
Data Science for Customer Insights

Outline

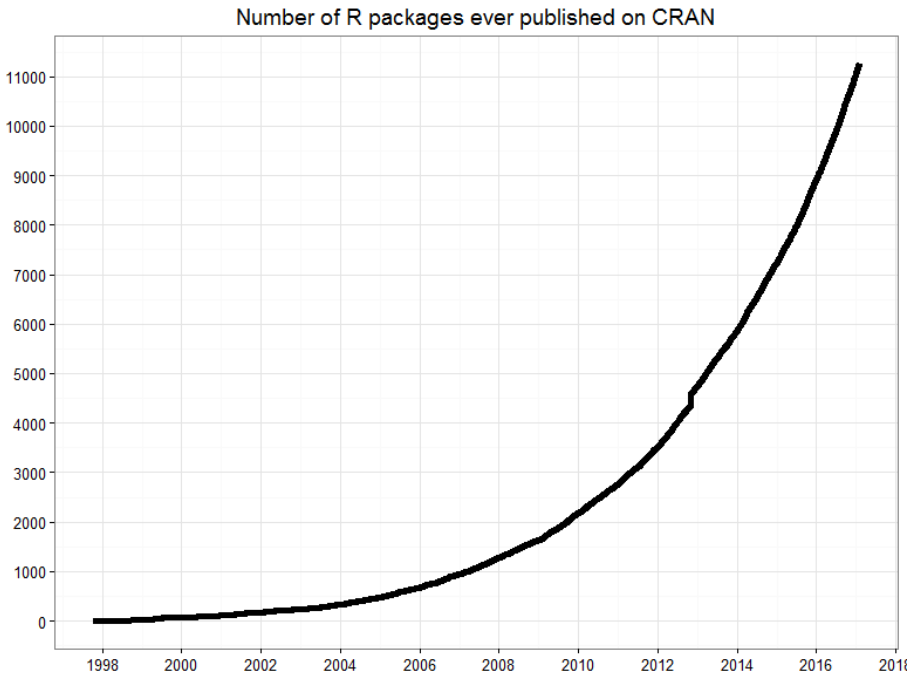
1. A very quick introduction to R
2. The tidyverse
3. A real world example
4. Wrap up, questions, & learning more

Appendix has links to learn more.

Brief history of R

- First there was S by John Chambers, et al
  - 1976: first internal production use at Bell Labs
  - 1980: first distribution outside of Bell Labs
- Then R by Ross Ihaka & Robert Gentleman at University of Auckland
  - 1995: the initial release,
  - 2000: the V1.0 “production” release
- RStudio & Hadley Wickham
  - 2011: the initial V0.92 release of RStudio IDE
  - 2017: R for Data Science published

Growth of R



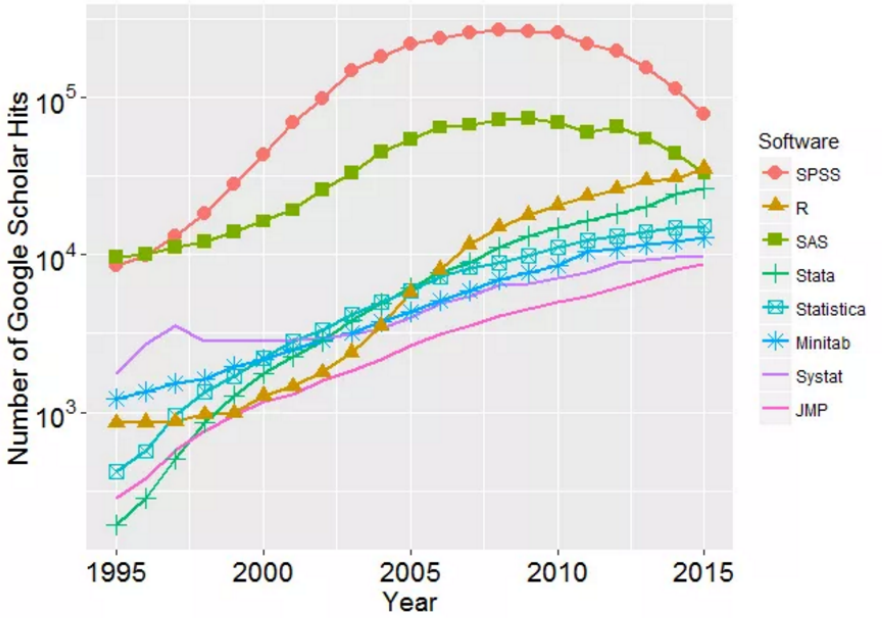


Figure 2f. A logarithmic view of the number of scholarly articles found in each year by Google Scholar. This combines the previous two figures into one by compressing the y-axis with a base 10 logarithm.

Big Ideas of R

- “R is a free software environment for statistical computing and graphics” (www.r-project.org)
- Base language is functional, object orientated (sort of), does vector arithmetic, missing data handled with NA’s
- Methods produce a result object (not printed output). Generic functions, like print(), specific to object type.
- R ecosystem from the beginning to support rich development.
- 10456 packages on CRAN (17Apr17)

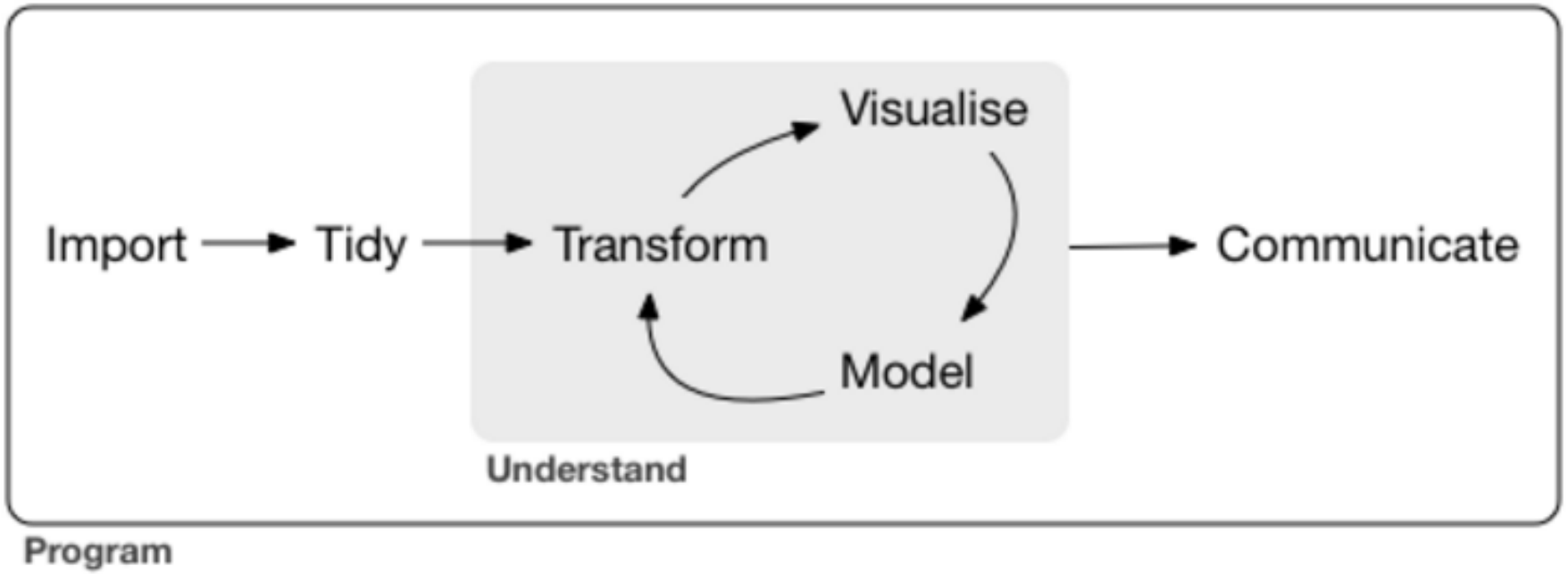
The dataframe in R

- Columns of same data type:
  - Character, integer, real, logical, Date, POSIX (timestamp with optional time zone)
  - But mix data types in dataframe
- Rows across columns
- Same idea as a SQL table, Excel sheet (with typed columns), CSV or tab delimited text files, etc.
- RAM resident (but with good workarounds for big data)

The tidyverse - Outline

- Hadley's data science workflow
- Packages in the tidyverse
  - Packages that are friends of the tidyverse
- dplyr package – the data wrangling workhorse
- Tidy & untidy data
- tidyr package – getting tidy
- Quick look at other packages & functions

Hadley's Data Science Workflow



This and most figures in this section are from *R for Data Science* by Garrett Grolemund and Hadley Wickham at [r4ds.had.co.nz](r4ds.had.co.nz)

install.packages("tidyverse")

core:

- **ggplot2**, for data visualisation.
- **dplyr**, for data manipulation.
- **tidyr**, for data tidying.
- **readr**, for data import.
- **purrr**, for functional programming.
- **tibble**, for tibbles, a modern re-imagining of data frames.

friends for data manipulation:

- **hms**, for times.
- **stringr**, for strings.
- **lubridate**, for date/times.
- **forcats**, for factors.

friends for data import:

- **DBI**, for databases.
- **haven**, for SPSS, SAS and Stata files.
- **httr**, for web apis.
- **jsonlite** for JSON.
- **readxl**, for .xls and .xlsx files.
- **rvest**, for web scraping.
- **xml2**, for XML.

friends for modeling:

- **modelr**, for simple modelling within a pipeline
- **broom**, for turning models into tidy data

Loading tidyverse Packages

`library(tidyverse)`

- loads the core packages

The friends must be loaded independently, eg

- `library(stringr)`
- `library(lubridate)`
- Etc.

dplyr Data Wrangling Verbs

- filter( ) – rows by their values
- arrange( ) – reorder rows
- select( ) – pick and/or exclude columns
- mutate( ) – create new columns
- summarize( ) – collapse rows with summaries

And any of above may scoped over a group of rows with:

- group_by( ) – define groups based on categorical variables

When working with data in RDMS' (eg Redshift):

- collect( ) – build SQL for the prior verbs, send to RDBMS, and pull down result set while maintaining data types.

Using dplyr Functions

Arguments:

- 1st is the data frame to work on
- Following arguments define what to do on which named columns in the data frame

Result is always another data frame:

```
# starting with myDF0 - observations across the world
myDF1 <- filter(myDF0, country == "US")
myDF <- summarize(myDF1, Count = n())

# or, without intermediate dataframes
myDF <- summarize(
  filter(myDF0, country == "US"),
  Count = n())
```

Pipes to the Rescue!

%>%

This is a pipe.

Takes the result of the LHS and pushes into the first argument of the RHS:

```
# or, with pipes
myDF <- myDF0 %>%
  filter(country == "US") %>%
  summarize(Count = n())
```

dplyr Example (1 of 2)

Business question 2: by monthly cohort how many and what percentage of paying subscribers are currently subscribed?

Words

Group the subsubscriber_summary rows having positive RTD by monthly cohort. Calculate:

1. Number of subscriber starts as the number of rows,
2. Number still subscribed as the number with is_subscribed being TRUE, and
3. Percentage that are still subscribed from 1) and 2).

Arrange in monthly cohort order.

SQL

```
SELECT ss.cohort_yymm,
       COUNT(*) AS Number_Starts,
       SUM(ss.is_subscribed::INT) AS Number_Subscribed,
       (100.0 * SUM(ss.is_subscribed::INT) /
         COUNT(*))::NUMERIC(10, 1) AS Per cent_Subscribed
  FROM subscriber_summary AS
 WHERE ss.revenue_to_date > 0
 GROUP BY 1
 ORDER BY 1;
```

Figure 4: Solution to business question 2

From Jim's recent paper [A Data Structure for Customer Insights](#) in Applied Marketing Analytics

dplyr Example (2 of 2)

```
library(tidyverse)

bq2 <- subscriber_summary %>%
  filter(revenue_to_date > 0) %>%
  group_by(cohort_yymm) %>%
  summarize(Number_Starts = n(),
            Number_Subscribed = sum(is_Subscribed),
            Percent_Subscribed = 100 * Number_Subscribed / Number_Starts) %>%
  arrange(cohort_yymm)
```

Observations:

- Fewer lines than SQL (by 1)
- A more natural sequence: start with data source, end with order
- Reference prior calculations by name!

Hadley's 3 Rules for Being Tidy

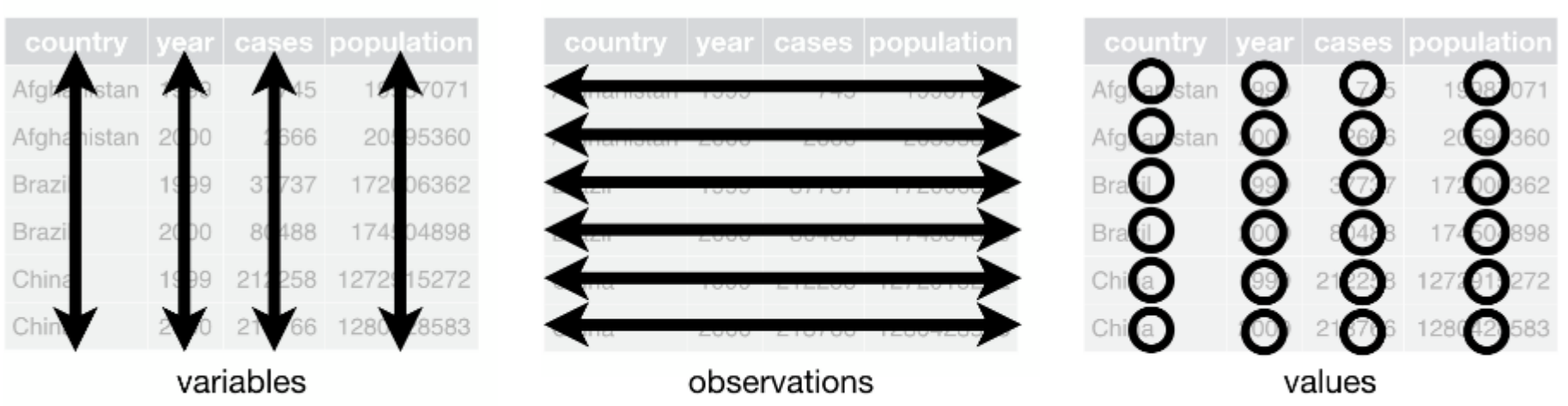


Figure 12.1: Following three rules makes a dataset tidy: variables are in columns, observations are in rows, and values are in cells.

Note that any two conditions imply the third.

If values are not in cells, where are they?

Tidy Case Study

The widest,

un-tidyist,

worst data file,

ever!

Which came in as

an _____ file,

of course.

Tidy Case Study

A process flow survey broken down into Stages, Process Groups, and individual Tasks coded as <stage>. <process group>. <task>

For each we have # cycles to pass QC and time per cycle.

The first ~50 rows (out of 3400) & 775 columns:

The first few data columns:

1.1.1 Time	1.1.1 Cycle	1.1.1 Total	1.1.2 Time	1.1.2 Cycle	1.1.2 Total	1.1.3 Time	1.1.3 Cycle	1.1.3 Total	1.1.4 Time	1.1.4 Cycle	1.1.4 Total	1.1 One Cycle Total	1.1 Total	1.2.1 Time	1.2.1 Cycle
10	100	1000	10	100	1000	10	100	1000	10	100	1000	40	4000	10	100
									15	1	15	15	15		
15	1	15	15	1	15	15	1	15	15	1	15	60	60	10	1

The last few data columns:

11.1.1 Time	11.1.1 Cycle	11.1.1 Total	11.1.2 Time	11.1.2 Cycle	11.1.2 Total	11.1.3 Time	11.1.3 Cycle	11.1.3 Total	11.1 One Cycle Total	11.1 Total	11 One Cycle Total	11 Total	One Cycle Total Time in minutes	Total Time in minutes
10	10	100	10	2	20	10	20	200	30	320	30	320	1530	14650
30	3	90	20	3	60	30	1	30	80	180	80	180	605	1660
5	1	5	5	1	5	5	1	5	15	15	15	15	1450	1455

Load & Basic Cleanup

l sheet; Drop Columns, Fix Column Names; Split into Details & Measures

```
## Read Sheet & Basic Cleanup

```{r ReadSheet}
InputFile <- "../DataIn/Transformed - MTC_DataExport_201607181050 - 0822-1.xlsx"
mtc_tc0 <- read_excel(InputFile)
# Drop PII data & all "Total" columns
mtc_tc <- mtc_tc0 %>%
  select(-(2:6), -contains("Total"))
# Give the unique identifier column a name
colnames(mtc_tc)[1] <- "ID"
# split off respondents' properties (replacing spaces in column names with underscores)
mtc_tc_details <- mtc_tc %>%
  select(1:14)
colnames(mtc_tc_details) <- str_replace_all(colnames(mtc_tc_details), fixed(" "), "_")
# from their responses (left padding single digit Stage in column names with zero)
mtc_tc_measures <- mtc_tc %>%
  select(-(2:14)) %>%
  mutate_at(vars(-ID), as.integer)
colnames(mtc_tc_measures) <- ifelse(str_detect(colnames(mtc_tc_measures), "^[1-9]\\."),
                                    paste0("0", colnames(mtc_tc_measures)),
                                    colnames(mtc_tc_measures))
```
```

en shot from the R Notebook showing the “ReadSheet” code chunk.

Gather our Columns to be Tidy

In this example, from R4DS, the observations for the years are in individual columns. We want a column for the year and a second column for the number of cases. “year” and “cases” are the key/value pair.

Figure 12.2: Gathering `table4` into a tidy form.

In the final result, the gathered columns are dropped, and we get new `key` and `value` columns.

Tidying up (1 of 4)

Prior step left us with:

```
> mtc_tc_measures
# A tibble: 3,411 × 449
      ID `01.1.1 Time` `01.1.1 Cycle` `01.1.2 Time` `01.1.2 Cycle`
   <dbl>         <int>          <int>         <int>          <int>
1      2            10              1            30              2
2      4            NA             NA            NA             NA
3      5
4      6
5      7                                                       NA
6      8                                            60          2
7     10                                            NA         NA
8     12            NA             NA            NA             NA
9     13            NA             NA            NA             NA
10    14            60              1            NA             NA
# ... with 3,401 more rows, and 444 more variables: `01.1.3 Time` <int>,
#   `01.1.3 Cycle` <int>, `01.1.4 Time` <int>, `01.1.4 Cycle` <int>, `01.2.1
#   Time` <int>, `01.2.1 Cycle` <int>, `01.2.2 Time` <int>, `01.2.2
```

Column name becomes key: "TaskMetric"

Data becomes value: "Value"

Now: 1) gather data columns into key/value pairs called TaskMetric & Value; 2) separate TaskMetric into keys Task & Metric

Tidying up (2 of 4)

```
```{r tidyUp}
mtc_tc_tidy0 <- mtc_tc_measures %>%
  gather(TaskMetric, Value, -ID) %>%
  separate(TaskMetric, c("Task", "Metric"), sep = " ")
```

```
> mtc_tc_tidy0 %>% arrange(ID, Task, Metric)
# A tibble: 1,528,128 × 4
      ID   Task Metric Value
   <dbl>  <chr>  <chr> <int>
1      2 01.1.1  Cycle     1
2      2 01.1.1   Time    10
3      2 01.1.2  Cycle     2
4      2 01.1.2   Time    30
5      2 01.1.3  Cycle     3
6      2 01.1.3   Time    20
7      2 01.1.4  Cycle     4
8      2 01.1.4   Time   120
9      2 01.2.1  Cycle    NA
10     2 01.2.1   Time    NA
# ... with 1,528,118 more rows
```

Include "Tot Task Time" which is just product:
Cycle * Time
```

Tidying up (3 of 4)

First make a new tibble mtc_tc_tt which is just the total task time for each Task for each ID:

```
91  # get Total Time = Cycle * Time
92  mtc_tc_tt <- mtc_tc_tidy0 %>%
93    group_by(ID, Task) %>%
94    summarise(Value = prod(Value)) %>%
95    mutate(Metric = "Tot Task Time") %>%
96    select(ID, Task, Metric, Value)
```

```
> mtc_tc_tt
Source: local data frame [764,064 x 4]
Groups: ID [3,411]

      ID   Task         Metric Value
   <dbl>  <chr>          <chr> <dbl>
1      2 01.1.1 Tot Task Time    10
2      2 01.1.2 Tot Task Time    60
3      2 01.1.3 Tot Task Time    60
4      2 01.1.4 Tot Task Time   480
```

Add, bind these rows to mtc_tc_tidy0

Tidying up (4 of 4)

Finally: 1) bind in Total Task Time rows; 2) Split out Stage & Process Group from Task; 3) Sort; & 4) Select columns for resulting data set.

```
 97  mtc_tc_tidy <- mtc_tc_tidy0 %>%
 98    bind_rows(mtc_tc_tt) %>%
 99    mutate(Stage = as.integer(str_sub(Task, 1, 2)),
100           Process_Group = as.numeric(str_sub(Task, 1, 4))) %>%
101    arrange(ID, Stage, Process_Group, Task, Metric) %>%
102    select(ID, Stage, Process_Group, Task, Metric, Value)
```

```
> mtc_tc_tidy
# A tibble: 2,292,192 × 6
      ID Stage Process_Group   Task         Metric Value
   <dbl> <int>         <dbl>  <chr>          <chr> <dbl>
1      2     1           1.1 01.1.1          Cycle     1
2      2     1           1.1 01.1.1           Time    10
3      2     1           1.1 01.1.1 Tot Task Time    10
4      2     1           1.1 01.1.2          Cycle     2
5      2     1           1.1 01.1.2           Time    30
6      2     1           1.1 01.1.2 Tot Task Time    60
```

Now that we are tidy…

It is trivial to do rollups like:

```
112  mtc_Stage_rollup <- mtc_tc_tidy_dtls %>%
113    group_by(Stage, Metric) %>%
114    summarise(Num_Values = sum(!is.na(Value)),
115              Avg_Value = mean(Value, na.rm = TRUE),
116              Median_Value = median(Value, na.rm = TRUE))
```

```
> mtc_Stage_rollup
Source: local data frame [33 x 5]
Groups: Stage [?]

   Stage         Metric Num_Values Avg_Value Median_Value
   <int>          <chr>      <int>     <dbl>        <dbl>
1      1          Cycle      34586  2.608512            1
2      1           Time      34586 39.514399           15
3      1 Tot Task Time      34586 80.960851           30
4      2          Cycle      40741  2.504185            1
5      2           Time      40741 32.304951           15
6      2 Tot Task Time      40741 64.518642           20
```

Much More to tidyverse!

- 522 pages in print edition of R for Data Science
- Including workflow suggestions (RStudio):
  - Projects, R Markdown, R Notebooks, Git/Github
- Visualization for exploring & communication
  - ggplot2, forcats, …
- Data in RDMS'
  - RPostgreSQL, DB, using dplyr against Redshift, …
- Modeling in the tidyverse
  - modelr & broom
- Friends of tidyverse
  - stringr, lubridate, & from other contributors like: tidytext, …

What We Covered

- A quick introduction to R
- Tidyverse overview and some examples
- Tidying up a really messy Excel data set.
- Wrap up, questions, & learning more

Questions?
Comments?
Now is the time!

Learning More

- Jim's Learning R and RStudio
  - With many links to learn more
- Three dplyr examples from Berkeley R Beginners Meetup
  - US Bureau of Labor Statistics – Wide to Tidy
  - Sessionization of Web Events
  - Waiting for BART – A Simulation
- All of Jim's prior talks www.ds4ci.org/archives
  - Mostly about or using R
- Contact: Jim@DS4CI.org

Data Wrangling in the Tidyverse

21st Century R

DS Portugal Meetup, at Farfetch, Porto, Portugal
April 19, 2017

Jim Porzak
Data Science for Customer Insights

Outline

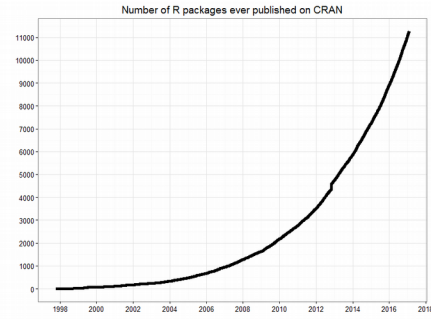
1. A very quick introduction to R
2. The tidyverse
3. A real world example
4. Wrap up, questions, & learning more

Appendix has links to learn more.

Brief history of R

- First there was S by John Chambers, et al
  - 1976: first internal production use at Bell Labs
  - 1980: first distribution outside of Bell Labs
- Then R by Ross Ihaka & Robert Gentleman at University of Auckland
  - 1995: the initial release,
  - 2000: the V1.0 “production” release
- RStudio & Hadley Wickham
  - 2011: the initial V0.92 release of RStudio IDE
  - 2017: R for Data Science published

Growth of R



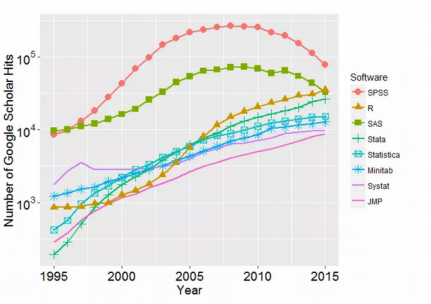


Figure 2f. A logarithmic view of the number of scholarly articles found in each year by Google Scholar. This combines the previous two figures into one by compressing the y-axis with a base 10 logarithm.

Big Ideas of R

- “R is a free software environment for statistical computing and graphics” (www.r-project.org)
- Base language is functional, object orientated (sort of), does vector arithmetic, missing data handled with NA’s
- Methods produce a result object (not printed output). Generic functions, like print(), specific to object type.
- R ecosystem from the beginning to support rich development.
- 10456 packages on CRAN (17Apr17)

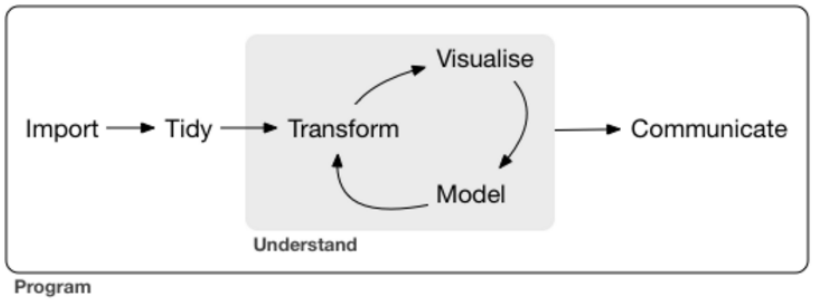
The dataframe in R

- Columns of same data type:
  - Character, integer, real, logical, Date, POSIX (timestamp with optional time zone)
  - But mix data types in dataframe
- Rows across columns
- Same idea as a SQL table, Excel sheet (with typed columns), CSV or tab delimited text files, etc.
- RAM resident (but with good workarounds for big data)

The tidyverse - Outline

- Hadley's data science workflow
- Packages in the tidyverse
  - Packages that are friends of the tidyverse
- dplyr package – the data wrangling workhorse
- Tidy & untidy data
- tidyr package – getting tidy
- Quick look at other packages & functions

Hadley's Data Science Workflow



This and most figures in this section are from *R for Data Science* by Garrett Grolemund and Hadley Wickham at [r4ds.had.co.nz](r4ds.had.co.nz)

install.packages("tidyverse")

core:

- **ggplot2**, for data visualisation.
- **dplyr**, for data manipulation.
- **tidyr**, for data tidying.
- **readr**, for data import.
- **purrr**, for functional programming.
- **tibble**, for tibbles, a modern re-imagining of data frames.

friends for data manipulation:

- **hms**, for times.
- **stringr**, for strings.
- **lubridate**, for date/times.
- **forcats**, for factors.

friends for data import:

- **DBI**, for databases.
- **haven**, for SPSS, SAS and Stata files.
- **httr**, for web apis.
- **jsonlite** for JSON.
- **readxl**, for .xls and .xlsx files.
- **rvest**, for web scraping.
- **xml2**, for XML.

friends for modeling:

- **modelr**, for simple modelling within a pipeline
- **broom**, for turning models into tidy data

Loading tidyverse Packages

`library(tidyverse)`

- loads the core packages

The friends must be loaded independently, eg

- `library(stringr)`
- `library(lubridate)`
- Etc.

dplyr Data Wrangling Verbs

- filter( ) – rows by their values
- arrange( ) – reorder rows
- select( ) – pick and/or exclude columns
- mutate( ) – create new columns
- summarize( ) – collapse rows with summaries

And any of above may scoped over a group of rows with:

- group_by( ) – define groups based on categorical variables

When working with data in RDMS' (eg Redshift):

- collect( ) – build SQL for the prior verbs, send to RDBMS, and pull down result set while maintaining data types.

Using dplyr Functions

Arguments:

- 1st is the data frame to work on
- Following arguments define what to do on which named columns in the data frame

Result is always another data frame:

```
# starting with myDF0 - observations across the world
myDF1 <- filter(myDF0, country == "US")
myDF <- summarize(myDF1, Count = n())

# or, without intermediate dataframes
myDF <- summarize(
  filter(myDF0, country == "US"),
  Count = n())
```

Pipes to the Rescue!

%>%

This is a pipe.

Takes the result of the LHS and pushes into the first argument of the RHS:

```
# or, with pipes
myDF <- myDF0 %>%
  filter(country == "US") %>%
  summarize(count = n())
```

dplyr Example (1 of 2)

Business question 2: by monthly cohort how many and what percentage of paying subscribers are currently subscribed?

Words

```
Group the subsubscriber_summary rows having positive RTD by
monthly cohort. Calculate:

  1. Number of subscriber starts as the number of rows,
  2. Number still subscribed as the number with is_subscribed
     being TRUE, and
  3. Percentage that are still subscribed from 1) and 2).

Arrange in monthly cohort order.
```

SQL

```
SELECT ss.cohort_yymm,
       COUNT(*) AS Number_Starts,
       SUM(ss.is_subscribed::INT) AS Number_Subscribed,
       (100.0 * SUM(ss.is_subscribed::INT) /
         COUNT(*))::NUMERIC(10, 1) AS Per cent_Subscribed
  FROM subscriber_summary AS
 WHERE ss.revenue_to_date > 0
 GROUP BY 1
 ORDER BY 1;
```

Figure 4: Solution to business question 2

From Jim's recent paper A Data Structure for Customer Insights in Applied Marketing Analytics

dplyr Example (2 of 2)

```
library(tidyverse)

bq2 <- subscriber_summary %>%
  filter(revenue_to_date > 0) %>%
  group_by(cohort_yymm) %>%
  summarize(Number_Starts = n(),
            Number_Subscribed = sum(is_Subscribed),
            Percent_Subscribed = 100 * Number_Subscribed / Number_Starts) %>%
  arrange(cohort_yymm)
```

Observations:

- Fewer lines than SQL (by 1)
- A more natural sequence: start with data source, end with order
- Reference prior calculations by name!

Hadley's 3 Rules for Being Tidy

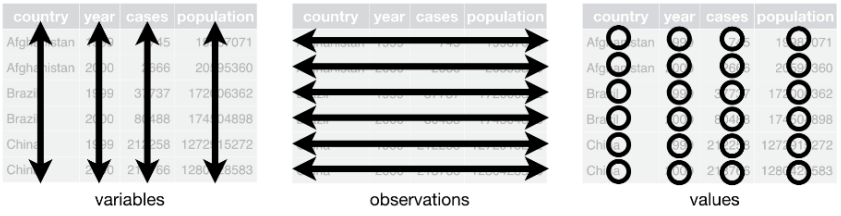


Figure 12.1: Following three rules makes a dataset tidy: variables are in columns, observations are in rows, and values are in cells.

Note that any two conditions imply the third.

If values are not in cells, where are they?

Tidy Case Study

The widest,
un-tidyist,
worst data file,
ever!

Which came in as
an _____ file,
of course.

Tidy Case Study

A process flow survey broken down into Stages, Process Groups, and individual Tasks coded as <stage>. <process group>. <task>

For each we have # cycles to pass QC and time per cycle.

The first ~50 rows (out of 3400) & 775 columns:

The first few data columns:

1.1.1 Time	1.1.1 Cycle	1.1.1 Total	1.1.2 Time	1.1.2 Cycle	1.1.2 Total	1.1.3 Time	1.1.3 Cycle	1.1.3 Total	1.1.4 Time	1.1.4 Cycle	1.1.4 Total	1.1 One Cycle Total	1.1 Total	1.2.1 Time	1.2.1 Cycle
10	100	1000	10	100	1000	10	100	1000	10	100	1000	40	4000	10	100
									15	1	15	15	15		
15	1	15	15	1	15	15	1	15	15	1	15	60	60	10	1

The last few data columns:

11.1.1 Time	11.1.1 Cycle	11.1.1 Total	11.1.2 Time	11.1.2 Cycle	11.1.2 Total	11.1.3 Time	11.1.3 Cycle	11.1.3 Total	11.1 One Cycle Total	11.1 Total	11 One Cycle Total	11 Total	One Cycle Total Time in minutes	Total Time in minutes
10	10	100	10	2	20	10	20	200	30	320	30	320	1530	14650
30	3	90	20	3	60	30	1	30	80	180	80	180	605	1660
5	1	5	5	1	5	5	1	5	15	15	15	15	1450	1455

Load & Basic Cleanup

l sheet; Drop Columns, Fix Column Names; Split into Details & Measures

```
## Read Sheet & Basic Cleanup

```{r ReadSheet}
InputFile <- "../DataIn/Transformed - MTC_DataExport_201607181050 - 0822-1.xlsx"
mtc_tc0 <- read_excel(InputFile)
# Drop PII data & all "Total" columns
mtc_tc <- mtc_tc0 %>%
  select(-(2:6), -contains("Total"))
# Give the unique identifier column a name
colnames(mtc_tc)[1] <- "ID"
# split off respondents' properties (replacing spaces in column names with underscores)
mtc_tc_details <- mtc_tc %>%
  select(1:14)
colnames(mtc_tc_details) <- str_replace_all(colnames(mtc_tc_details), fixed(" "), "_")
# from their responses (left padding single digit Stage in column names with zero)
mtc_tc_measures <- mtc_tc %>%
  select(-(2:14)) %>%
  mutate_at(vars(-ID), as.integer)
colnames(mtc_tc_measures) <- ifelse(str_detect(colnames(mtc_tc_measures), "^[1-9]\\."),
                                    paste0("0", colnames(mtc_tc_measures)),
                                    colnames(mtc_tc_measures))
```
```

en shot from the R Notebook showing the “ReadSheet” code chunk.

Gather our Columns to be Tidy

In this example, from R4DS, the observations for the years are in individual columns. We want a column for the year and a second column for the number of cases. “year” and “cases” are the key/value pair.

country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

table4

Figure 12.2: Gathering `table4` into a tidy form.

In the final result, the gathered columns are dropped, and we get new `key` and `value` columns.

Tidying up (1 of 4)

Prior step left us with:

```
> mtc_tc_measures
# A tibble: 3,411 × 449
      ID `01.1.1 Time` `01.1.1 Cycle` `01.1.2 Time` `01.1.2 Cycle`
   <dbl>         <int>          <int>         <int>          <int>
1      2            10              1            30              2
2      4            NA             NA            NA             NA
3      5
4      6
5      7                                                       NA
6      8                                                       60              2
7     10                                                       NA             NA
8     12            NA             NA            NA             NA
9     13            NA             NA            NA             NA
10    14            60              1            NA             NA
# ... with 3,401 more rows, and 444 more variables: `01.1.3 Time` <int>,
#   `01.1.3 Cycle` <int>, `01.1.4 Time` <int>, `01.1.4 Cycle` <int>, `01.2.1
#   Time` <int>, `01.2.1 Cycle` <int>, `01.2.2 Time` <int>, `01.2.2
```

Now: 1) gather data columns into key/value pairs called TaskMetric & Value; 2) separate TaskMetric into keys Task & Metric

Tidying up (2 of 4)

```
```{r tidyUp}
mtc_tc_tidy0 <- mtc_tc_measures %>%
  gather(TaskMetric, Value, -ID) %>%
  separate(TaskMetric, c("Task", "Metric"), sep = " ")
```

```
> mtc_tc_tidy0 %>% arrange(ID, Task, Metric)
# A tibble: 1,528,128 × 4
      ID   Task Metric Value
   <dbl>  <chr>  <chr> <int>
1      2 01.1.1  Cycle     1
2      2 01.1.1   Time    10
3      2 01.1.2  Cycle     2
4      2 01.1.2   Time    30
5      2 01.1.3  Cycle     3
6      2 01.1.3   Time    20
7      2 01.1.4  Cycle     4
8      2 01.1.4   Time   120
9      2 01.2.1  Cycle    NA
10     2 01.2.1   Time    NA
# ... with 1,528,118 more rows
```

Include "Tot Task Time" which is just product: Cycle * Time
```

Tidying up (3 of 4)

First make a new tibble mtc_tc_tt which is just the total task time for each Task for each ID:

```
91  # get Total Time = Cycle * Time
92  mtc_tc_tt <- mtc_tc_tidy0 %>%
93    group_by(ID, Task) %>%
94    summarise(Value = prod(Value)) %>%
95    mutate(Metric = "Tot Task Time") %>%
96    select(ID, Task, Metric, Value)
```

```
> mtc_tc_tt
Source: local data frame [764,064 x 4]
Groups: ID [3,411]

      ID   Task        Metric Value
   <dbl>  <chr>         <chr> <dbl>
1      2 01.1.1 Tot Task Time    10
2      2 01.1.2 Tot Task Time    60
3      2 01.1.3 Tot Task Time    60
4      2 01.1.4 Tot Task Time   480
```

Add, bind these rows to mtc_tc_tidy0

Tidying up (4 of 4)

Finally: 1) bind in Total Task Time rows; 2) Split out Stage & Process Group from Task; 3) Sort; & 4) Select columns for resulting data set.

```
 97  mtc_tc_tidy <- mtc_tc_tidy0 %>%
 98    bind_rows(mtc_tc_tt) %>%
 99    mutate(Stage = as.integer(str_sub(Task, 1, 2)),
100           Process_Group = as.numeric(str_sub(Task, 1, 4))) %>%
101    arrange(ID, Stage, Process_Group, Task, Metric) %>%
102    select(ID, Stage, Process_Group, Task, Metric, Value)
```

```
> mtc_tc_tidy
# A tibble: 2,292,192 × 6
      ID Stage Process_Group   Task        Metric Value
   <dbl> <int>         <dbl>  <chr>         <chr> <dbl>
1      2     1           1.1 01.1.1         Cycle     1
2      2     1           1.1 01.1.1          Time    10
3      2     1           1.1 01.1.1 Tot Task Time    10
4      2     1           1.1 01.1.2         Cycle     2
5      2     1           1.1 01.1.2          Time    30
6      2     1           1.1 01.1.2 Tot Task Time    60
```

Now that we are tidy…

It is trivial to do rollups like:

```
112  mtc_Stage_rollup <- mtc_tc_tidy_dtls %>%
113    group_by(Stage, Metric) %>%
114    summarise(Num_Values = sum(!is.na(Value)),
115              Avg_Value = mean(Value, na.rm = TRUE),
116              Median_Value = median(Value, na.rm = TRUE))
```

```
> mtc_Stage_rollup
Source: local data frame [33 x 5]
Groups: Stage [?]

   Stage        Metric Num_Values Avg_Value Median_Value
   <int>         <chr>      <int>     <dbl>        <dbl>
1      1         Cycle      34586  2.608512            1
2      1          Time      34586 39.514399           15
3      1 Tot Task Time      34586 80.960851           30
4      2         Cycle      40741  2.504185            1
5      2          Time      40741 32.304951           15
6      2 Tot Task Time      40741 64.518642           20
```

Much More to tidyverse!

- 522 pages in print edition of R for Data Science
- Including workflow suggestions (RStudio):
  - Projects, R Markdown, R Notebooks, Git/Github
- Visualization for exploring & communication
  - ggplot2, forcats, …
- Data in RDMS'
  - RPostgreSQL, DB, using dplyr against Redshift, …
- Modeling in the tidyverse
  - modelr & broom
- Friends of tidyverse
  - stringr, lubridate, & from other contributors like: tidytext, …

What We Covered

- A quick introduction to R
- Tidyverse overview and some examples
- Tidying up a really messy Excel data set.
- Wrap up, questions, & learning more

Questions?
Comments?
Now is the time!

Learning More

- Jim's Learning R and RStudio
  - With many links to learn more
- Three dplyr examples from Berkeley R Beginners Meetup
  - US Bureau of Labor Statistics – Wide to Tidy
  - Sessionization of Web Events
  - Waiting for BART – A Simulation
- All of Jim's prior talks www.ds4ci.org/archives
  - Mostly about or using R
- Contact: Jim@DS4CI.org