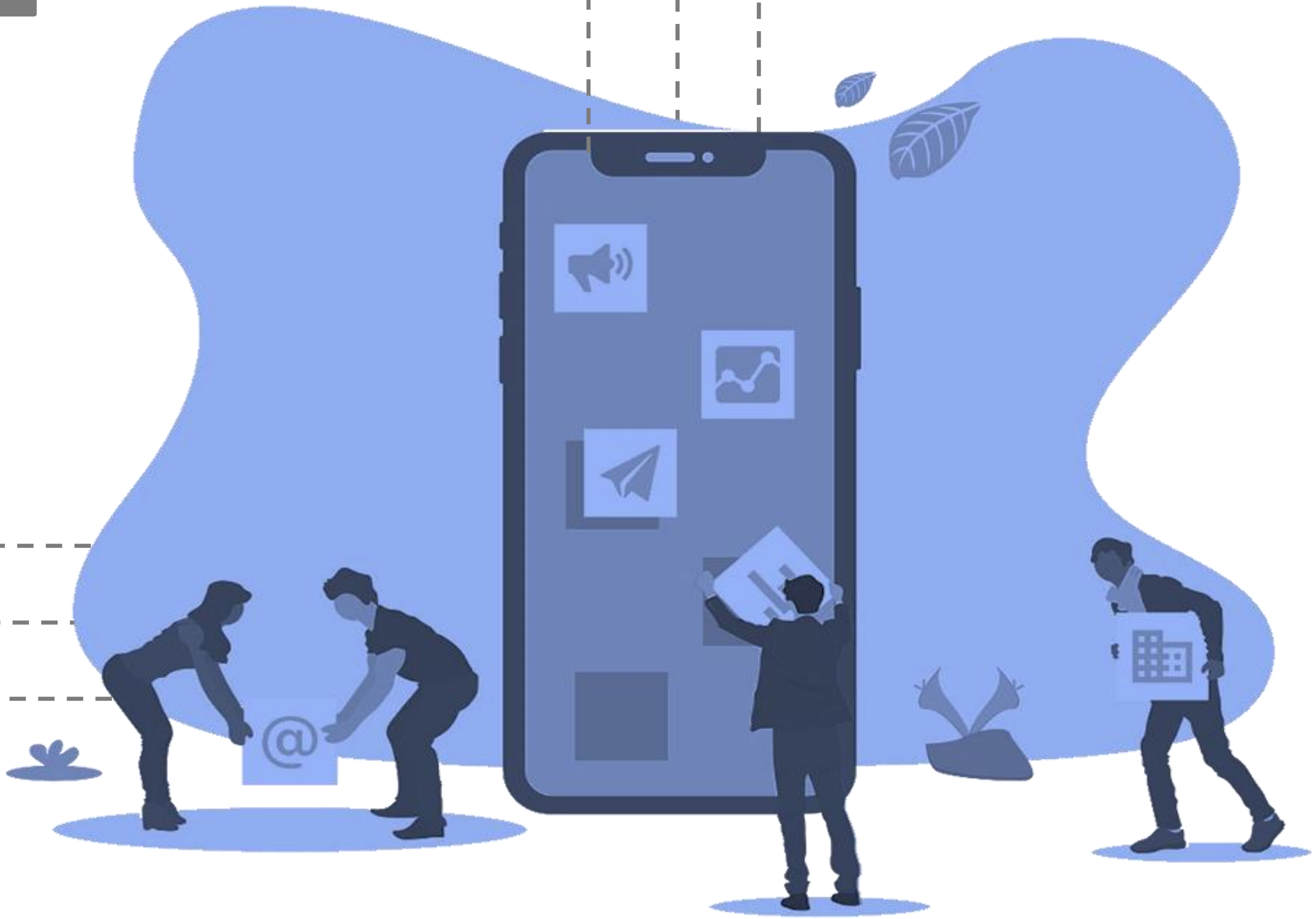
The Data Science Track



Prepared By: R. Daynolo

INTRODUCTION TO DATA SCIENCE WITH R

20. Managing and Reshaping of Data

CREATING NEW VARIABLES

Why create new variables?

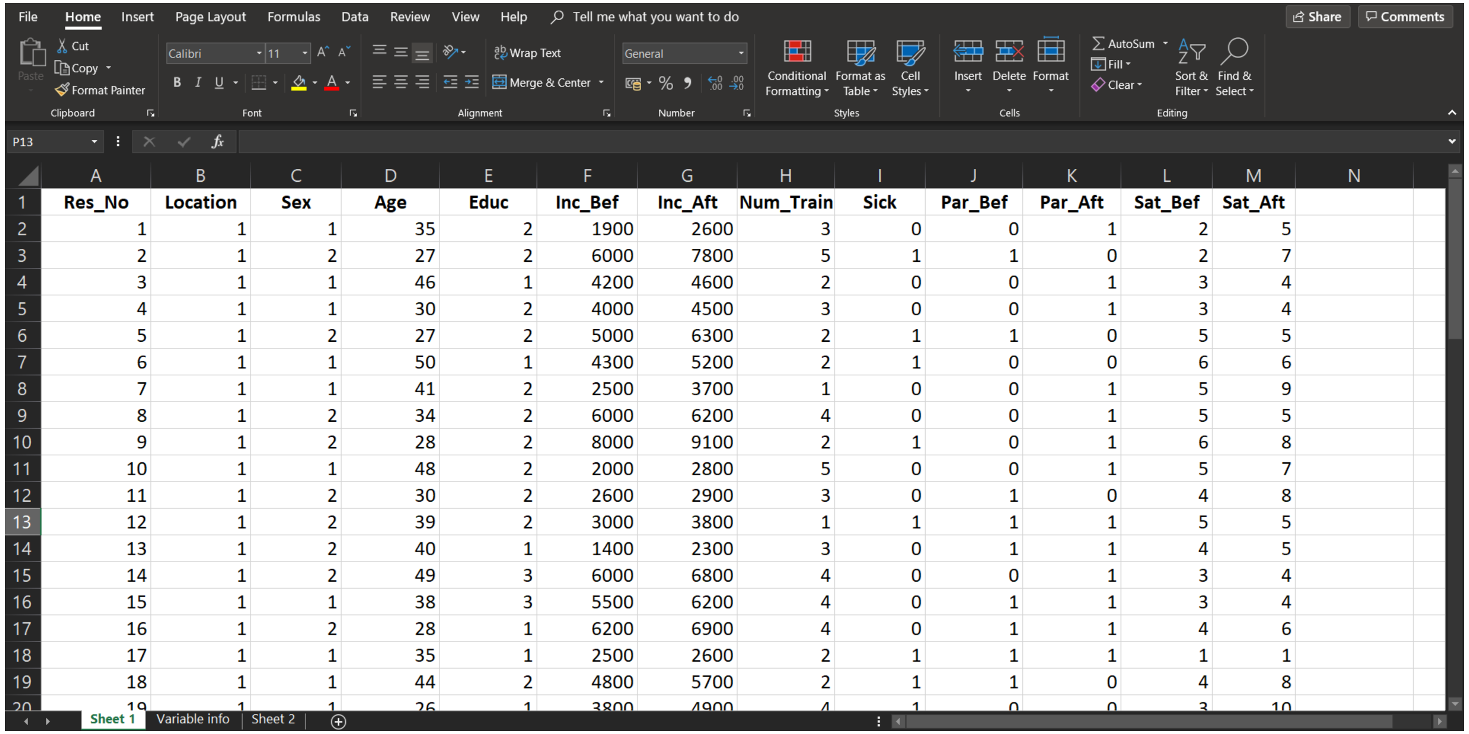
- Often the raw data won't have a value you are looking for
- You will need to transform the data to get the values you would like
- Usually you will add those values to the data frames you are working with
- Common variables to create
    - Missingness indicators
    - "Cutting up" quantitative variables
    - Applying transformations

COMMON TRANSFORMS

- **`abs(x)`** absolute value
- **`sqrt(x)`** square root
- **`ceiling(x)`** ceiling(3.475) is 4
- **`floor(x)`** floor(3.475) is 3
- **`round(x, digits=n)`** round(3.475, digits=2) is 3.48
- **`signif(x, digits=n)`** signif(3.475, digits=2) is 3.5
- **`cos(x)`**, **`sin(x)`** etc
- **`log(x)`** natural logarithm
- **`log2(x)`**, **`log10(x)`** other common logs
- **`exp(x)`** exponentiating x

RESHAPING DATA

The goal is tidy data



Res_No	Location	Sex	Age	Educ	Inc_Bef	Inc_Aft	Num_Train	Sick	Par_Bef	Par_Aft	Sat_Bef	Sat_Aft
1	1	1	35	2	1900	2600	3	0	0	1	2	5
2	1	2	27	2	6000	7800	5	1	1	0	2	7
3	1	1	46	1	4200	4600	2	0	0	1	3	4
4	1	1	30	2	4000	4500	3	0	0	1	3	4
5	1	2	27	2	5000	6300	2	1	1	0	5	5
6	1	1	50	1	4300	5200	2	1	0	0	6	6
7	1	1	41	2	2500	3700	1	0	0	1	5	9
8	1	2	34	2	6000	6200	4	0	0	1	5	5
9	1	2	28	2	8000	9100	2	1	0	1	6	8
10	1	1	48	2	2000	2800	5	0	0	1	5	7
11	1	2	30	2	2600	2900	3	0	1	0	4	8
12	1	2	39	2	3000	3800	1	1	1	1	5	5
13	1	2	40	1	1400	2300	3	0	1	1	4	5
14	1	2	49	3	6000	6800	4	0	0	1	3	4
15	1	1	38	3	5500	6200	4	0	1	1	3	4
16	1	2	28	1	6200	6900	4	0	1	1	4	6
17	1	1	35	1	2500	2600	2	1	1	1	1	1
18	1	1	44	2	4800	5700	2	1	1	0	4	8
19	1	1	26	1	3800	4900	4	1	0	0	3	10

1. Each variable forms a column
2. Each observation forms a row
3. Each table/file stores data about one kind of observation (e.g. people)

START WITH RESHAPING

```
> library(reshape2)
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

ANALYSIS STRUCTURE ALGORITHM PROCESS PROGRAMMING SOLVING KNOWLEDGE

MELTING DATA FRAMES

```r
mtcars$carname <- rownames(mtcars)
carMelt <- melt(mtcars, id=c("carname", "gear", "cyl"),
                measure.vars = c("mpg","hp"))
```

```
> head(carMelt, n = 3)
        carname gear cyl variable value
1     Mazda RX4    4   6      mpg  21.0
2 Mazda RX4 Wag    4   6      mpg  21.0
3    Datsun 710    4   4      mpg  22.8
>
> tail(carMelt, n = 3)
         carname gear cyl variable value
62  Ferrari Dino    5   6       hp   175
63 Maserati Bora    5   8       hp   335
64    Volvo 142E    4   4       hp   109
```

ANALYSIS STRUCTURE ALGORITHM PROCESS PROGRAMMING SOLVING KNOWLEDGE

CASTING DATA FRAMES

```
> cylData <- dcast(carMelt, cyl ~ variable)
Aggregation function missing: defaulting to length
> cylData
  cyl mpg hp
1   4  11 11
2   6   7  7
3   8  14 14
```

```
> cylData <- dcast(carMelt, cyl ~ variable, mean)
> cylData
  cyl      mpg        hp
1   4 26.66364  82.63636
2   6 19.74286 122.28571
3   8 15.10000 209.21429
```

AVERAGING VALUES

```
> head(InsectSprays)
  count spray
1    10     A
2     7     A
3    20     A
4    14     A
5    14     A
6    12     A
```

```
> tapply(InsectSprays$count, InsectSprays$spray, sum)
  A   B   C   D   E   F 
174 184  25  59  42 200 
```

ANOTHER WAY - SPLIT

```
> spIns <- split(InsectSprays$count, InsectSprays$spray)
> spIns
$A
 [1] 10  7 20 14 14 12 10 23 17 20 14
[12] 13

$B
 [1] 11 17 21 11 16 14 17 17 19 21  7
[12] 13

$C
 [1] 0 1 7 2 3 1 2 1 3 0 1 4

$D
 [1]  3  5 12  6  4  3  5  5  5  5  2
[12]  4

$E
```

ANOTHER WAY - APPLY

```
> sprCount <- lapply(spIns, sum)
> sprCount
$A
[1] 174

$B
[1] 184

$C
[1] 25

$D
[1] 59

$E
```

ANOTHER WAY - COMBINE

```
> unlist(sprCount)
  A   B   C   D   E   F 
174 184  25  59  42 200 
```

```
> sapply(spIns, sum)
  A   B   C   D   E   F 
174 184  25  59  42 200 
```

ANOTHER WAY – PLYR PACKAGE

```
> library(plyr)
> ddply(InsectSprays, .(spray), summarize, sum = sum(count))
  spray sum
1     A 174
2     B 184
3     C  25
4     D  59
5     E  42
6     F 200
```

CREATING A NEW VARIABLE

```
spraySums <- ddply(InsectSprays, .(spray), summarize,
                   sum = ave(count, FUN=sum))
```

```
> dim(spraySums)
[1] 72  2
> head(spraySums)
  spray sum
1     A 174
2     A 174
3     A 174
4     A 174
5     A 174
6     A 174
```

MORE INFORMATION

- A tutorial from the developer of plyr
  http://plyr.had.co.nz/09-user/
- A nice reshape tutorial
  https://www.slideshare.net/jeffreybreen/reshaping-data-in-r
- A good plyr primer
  https://www.r-bloggers.com/a-quick-primer-on-split-apply-combine-problems/
- See also the functions
  - `acast` – for casting as multi-dimensional arrays
  - `arrange` – for faster reordering without using order() commands
  - `mutate` – adding new variables

MANAGING DATA FRAMES WITH DPLYR

`dplyr`

The data frame is a key data structure in statistics and in R
- There is one observation per row
- Each column represents a variable or measure or characteristic
- Primary implementation that you will use is the default R implementation
- Other implementations, particularly relational databases systems

MANAGING DATA FRAMES WITH DPLYR

`dplyr`

- Developed by Hadley Wickham of Rstudio
- An optimized and distilled version of plyr package (also by Hadley)
- Does not provide any “new” functionality per se, but greatly simplifies existing functionality in R
- Provides a “grammar” (in particular, verbs) for data manipulation
- Is very fast, as many key operations are coded in C++

DPLYR VERBS

- `select`: return a subset of the columns of the data frame
- `filter`: extract a subset of rows from a data frame based on logical conditions
- `arrange`: reorder rows of a data frame
- `rename`: rename variables in a data frame
- `mutate`: add new variables/columns or transform existing variables
- `summarise / summarize`: generate summary statistics of different variables in the data frame, possibly within strata

There is also a handy `print` method that prevents you from printing a lot of data to the console.

DPLYR PROPERTIES

- The first argument is a data frame.
- The subsequent arguments describe what to do with it, and you can refer to columns in the data frame directly without using the $ operator (just use the names).
- The result is a new data frame
- Data frames must be properly formatted and annotated for this to all be useful

SELECT

```
> chicago <- readRDS("chicago.rds")
> dim(chicago)
[1] 6940    8
> head(select(chicago, 1:5))
  city tmpd   dptp       date pm25tmean2
1 chic 31.5 31.500 1987-01-01         NA
2 chic 33.0 29.875 1987-01-02         NA
3 chic 33.0 27.375 1987-01-03         NA
4 chic 29.0 28.625 1987-01-04         NA
5 chic 32.0 28.875 1987-01-05         NA
6 chic 40.0 35.125 1987-01-06         NA
```

SELECT

```
> names(chicago)[1:3]
[1] "city" "tmpd" "dptp"
> head(select(chicago, city:dptp))
  city tmpd   dptp
1 chic 31.5 31.500
2 chic 33.0 29.875
3 chic 33.0 27.375
4 chic 29.0 28.625
5 chic 32.0 28.875
6 chic 40.0 35.125
```

SELECT

In `dplyr` you can do

```
> head(select(chicago, -(city:dptp)))
        date pm25tmean2 pm10tmean2 o3tmean2 no2tmean2
1 1987-01-01         NA   34.00000 4.250000  19.98810
2 1987-01-02         NA         NA 3.304348  23.19099
3 1987-01-03         NA   34.16667 3.333333  23.81548
4 1987-01-04         NA   47.00000 4.375000  30.43452
5 1987-01-05         NA         NA 4.750000  30.33333
6 1987-01-06         NA   48.00000 5.833333  25.77233
```

SELECT

Equivalent base R

```
> i <- match("city", names(chicago))
> j <- match("dptp", names(chicago))
> head(chicago[, -(i:j)])
        date pm25tmean2 pm10tmean2 o3tmean2 no2tmean2
1 1987-01-01         NA   34.00000 4.250000  19.98810
2 1987-01-02         NA         NA 3.304348  23.19099
3 1987-01-03         NA   34.16667 3.333333  23.81548
4 1987-01-04         NA   47.00000 4.375000  30.43452
5 1987-01-05         NA         NA 4.750000  30.33333
6 1987-01-06         NA   48.00000 5.833333  25.77233
```

FILTER

```
> chicago.f <- filter(chicago, pm25tmean2>30)
> head(select(chicago.f, 1:3, pm25tmean2),10)
   city tmpd dptp pm25tmean2
1  chic   23 21.9      38.10
2  chic   28 25.8      33.95
3  chic   55 51.3      39.40
4  chic   59 53.7      35.40
5  chic   57 52.0      33.30
6  chic   57 56.0      32.10
7  chic   75 65.8      56.50
8  chic   61 59.0      33.80
9  chic   73 60.3      30.30
10 chic   78 67.1      41.40
```

FILTER

```
> chicago.f <- filter(chicago, pm25tmean2>30 & tmpd>80)
> head(select(chicago.f, 1:3, pm25tmean2),10)
   city tmpd dptp pm25tmean2
1  chic   81 71.2    39.6000
2  chic   81 70.4    31.5000
3  chic   82 72.2    32.3000
4  chic   84 72.9    43.7000
5  chic   85 72.6    38.8375
6  chic   84 72.6    38.2000
7  chic   82 67.4    33.0000
8  chic   82 63.5    42.5000
9  chic   81 70.4    33.1000
10 chic   82 66.2    38.8500
```

ANALYSIS STRUCTURE ALGORITHM PROCESS PROGRAMMING SOLVING KNOWLEDGE

ARRANGE

Reordering rows of a data frame (while preserving corresponding order of other columns) is normally a pain to do in R.

```
> chicago <- arrange(chicago, date)
> head(select(chicago, date, pm25tmean2), 3)
        date pm25tmean2
1 1987-01-01         NA
2 1987-01-02         NA
3 1987-01-03         NA
> tail(select(chicago, date, pm25tmean2), 3)
           date pm25tmean2
6938 2005-12-29    7.45000
6939 2005-12-30   15.05714
6940 2005-12-31   15.00000
```

ANALYSIS STRUCTURE ALGORITHM PROCESS PROGRAMMING SOLVING KNOWLEDGE

ARRANGE

Columns can be arranged in descending order too.

```
> chicago <- arrange(chicago, desc(date))
> head(select(chicago, date, pm25tmean2), 3)
        date pm25tmean2
1 2005-12-31   15.00000
2 2005-12-30   15.05714
3 2005-12-29    7.45000
> tail(select(chicago, date, pm25tmean2), 3)
           date pm25tmean2
6938 1987-01-03         NA
6939 1987-01-02         NA
6940 1987-01-01         NA
```

RENAME

Renaming a variable in a data frame in R is surprisingly hard to do!

```
> head(chicago[, 1:5], 3)
  city tmpd dptp       date pm25tmean2
1 chic   35 30.1 2005-12-31   15.00000
2 chic   36 31.0 2005-12-30   15.05714
3 chic   35 29.4 2005-12-29    7.45000
> chicago <- rename(chicago, dewpoint = dptp,
+                   pm25 = pm25tmean2)
> head(chicago[, 1:5], 3)
  city tmpd dewpoint       date     pm25
1 chic   35     30.1 2005-12-31 15.00000
2 chic   36     31.0 2005-12-30 15.05714
3 chic   35     29.4 2005-12-29  7.45000
```

MUTATE

```
> chicago <- mutate(chicago, pm25detrend = pm25 - mean(pm25, na.rm=TRUE))
> head(select(chicago, pm25, pm25detrend))
      pm25 pm25detrend
1 15.00000   -1.230958
2 15.05714   -1.173815
3  7.45000   -8.780958
4 17.75000    1.519042
5 23.56000    7.329042
6  8.40000   -7.830958
```

GROUP_BY

Generating summary statistics by stratum

```
chicago <- mutate(chicago,
                  tempcat = factor(1 * (tmpd >80),
                                   labels = c("cold", "hot")))
hotcold <- group_by(chicago, tempcat)
summarize(hotcold, pm25 = mean(pm25, na.rm = TRUE),
          o3 = max(o3tmean2),
          no2 = median(no2tmean2))
```

```
# A tibble: 3 x 4
  tempcat   pm25      o3    no2
  <fct>    <dbl>   <dbl>  <dbl>
1 cold    15.978 66.588  24.549
2 hot     26.481 62.970  24.939
3 NA      47.738  9.4167 37.444
```

ANALYSIS STRUCTURE ALGORITHM PROCESS PROGRAMMING SOLVING KNOWLEDGE

GROUP_BY

```
chicago <- mutate(chicago,
                  year = as.POSIXlt(date)$year + 1900)
years <- group_by(chicago, year)
summarize(years, pm25 = mean(pm25, na.rm = TRUE),
          o3 = max(o3tmean2, na.rm = TRUE),
          no2 = median(no2tmean2, na.rm = TRUE))
```

```
# A tibble: 19 x 4
   year  pm25     o3    no2
  <dbl> <dbl>  <dbl>  <dbl>
1  1987   NaN 62.970 23.494
2  1988   NaN 61.677 24.523
3  1989   NaN 59.727 26.141
4  1990   NaN 52.229 22.596
5  1991   NaN 63.104 21.382
6  1992   NaN 50.829 24.789
```

%>%

```r
chicago %>% mutate(month = as.POSIXlt(date)$mon + 1) %>%
    group_by(month) %>%
    summarize(pm25 = mean(pm25, na.rm = TRUE),
              o3 = max(o3tmean2, na.rm = TRUE),
              no2 = median(no2tmean2, na.rm = TRUE))
```

```
# A tibble: 12 x 4
   month   pm25     o3    no2
   <dbl>  <dbl>  <dbl>  <dbl>
 1     1 17.770 28.222 25.354
 2     2 20.375 37.375 26.780
 3     3 17.408 39.050 26.770
 4     4 13.859 47.949 25.031
 5     5 14.074 52.75  24.222
 6     6 15.865 66.588 25.011
```

MERGING DATA

plos.org create account sign in

PLOS | ONE

PUBLISH ABOUT BROWSE SEARCH

advanced search

OPEN ACCESS PEER-REVIEWED

RESEARCH ARTICLE

158 Save	15 Citation
24,835 View	91 Share

Cooperation between Referees and Authors Increases Peer Review Accuracy

Jeffrey T. Leek, Margaret A. Taub, Fernando J. Pineda

Published: November 9, 2011 • https://doi.org/10.1371/journal.pone.0026895

Article | Authors | Metrics | Comments | Media Coverage

Download PDF

Print Share

Abstract
Introduction
Results
Discussion
Materials and Methods
Author Contributions
References

Reader Comments (1)
Media Coverage (2)

Abstract

Peer review is fundamentally a cooperative process between scientists in a community who agree to review each other's work in an unbiased fashion. Peer review is the foundation for decisions concerning publication in journals, awarding of grants, and academic promotion. Here we perform a laboratory study of open and closed peer review based on an online game. We show that when reviewer behavior was made public under open review, reviewers were rewarded for refereeing and formed significantly more cooperative interactions (13% increase in cooperation, P=0.018). We also show that referees and authors who participated in cooperative interactions had an 11% higher reviewing accuracy rate (P=0.016). Our results suggest that increasing cooperation in the peer review process can lead to a decreased risk of reviewing errors.

Check for updates

Figures

https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0026895

ANALYSIS STRUCTURE ALGORITHM PROCESS PROGRAMMING SOLVING KNOWLEDGE

MERGING DATA

```
> head(reviews)
  id solution_id reviewer_id      start       stop time_left accept
1  1           3          27 1304095698 1304095758      1754      1
2  2           4          22 1304095188 1304095206      2306      1
3  3           5          28 1304095276 1304095320      2192      1
4  4           1          26 1304095267 1304095423      2089      1
5  5          10          29 1304095456 1304095469      2043      1
6  6           2          29 1304095471 1304095513      1999      1
```

```
> head(solutions)
  id problem_id subject_id      start       stop time_left answer
1  1        156         29 1304095119 1304095169      2343      B
2  2        269         25 1304095119 1304095183      2329      C
3  3         34         22 1304095127 1304095146      2366      C
4  4         19         23 1304095127 1304095150      2362      D
5  5        605         26 1304095127 1304095167      2345      A
6  6        384         27 1304095131 1304095270      2242      C
```

ANALYSIS STRUCTURE ALGORITHM PROCESS PROGRAMMING SOLVING KNOWLEDGE

MERGING DATA – merge()

- Merges data frames
- Important parameter: `x, y, by, by.x, by.y, all`

```
> names(reviews)
[1] "id"          "solution_id" "reviewer_id" "start"
[5] "stop"        "time_left"   "accept"
> names(solutions)
[1] "id"         "problem_id" "subject_id" "start"
[5] "stop"       "time_left"  "answer"
```

MERGING DATA – merge()

```
mergedData <- merge(reviews, solutions, by.x = "solution_id",
                    by.y = "id", all = TRUE)
```

```
> head(mergedData)
  solution_id id reviewer_id    start.x     stop.x time_left.x accept
1           1  4          26 1304095267 1304095423        2089      1
2           2  6          29 1304095471 1304095513        1999      1
3           3  1          27 1304095698 1304095758        1754      1
4           4  2          22 1304095188 1304095206        2306      1
5           5  3          28 1304095276 1304095320        2192      1
6           6 16          22 1304095303 1304095471        2041      1
  problem_id subject_id    start.y     stop.y time_left.y answer
1        156         29 1304095119 1304095169        2343      B
2        269         25 1304095119 1304095183        2329      C
3         34         22 1304095127 1304095146        2366      C
4         19         23 1304095127 1304095150        2362      D
5        605         26 1304095127 1304095167        2345      A
6        384         27 1304095131 1304095270        2242      C
```

DEFAULT – MERGE ALL COMMON COLUMN NAMES

```
> intersect(names(solutions), names(reviews))
[1] "id"          "start"       "stop"        "time_left"
```

```
> mergedData2 <- merge(reviews, solutions, all = TRUE)
> head(mergedData2)
  id      start       stop time_left solution_id reviewer_id accept
1  1 1304095119 1304095169      2343          NA          NA     NA
2  1 1304095698 1304095758      1754           3          27      1
3  2 1304095119 1304095183      2329          NA          NA     NA
4  2 1304095188 1304095206      2306           4          22      1
5  3 1304095127 1304095146      2366          NA          NA     NA
6  3 1304095276 1304095320      2192           5          28      1
  problem_id subject_id answer
1        156         29      B
2         NA         NA   <NA>
3        269         25      C
4         NA         NA   <NA>
5         34         22      C
6         NA         NA   <NA>
```

USING JOIN IN THE PLYR PACKAGE

- Faster, but less full featured – defaults to left join, see help file for more

```r
library(plyr)
df1 <- data.frame(id = sample(1:10), x = rnorm(10))
df2 <- data.frame(id = sample(1:10), y = rnorm(10))
```

USING JOIN IN THE PLYR PACKAGE

```
> arrange(join(df1, df2), id)
Joining by: id
   id            x           y
1   1  0.627027364 -0.60073051
2   2  0.700967679  0.03386791
3   3 -0.810659859  0.60660141
4   4  0.005873449 -1.31129057
5   5  0.380967414  0.73563731
6   6 -0.170144848  1.60118227
7   7 -1.698512322 -0.14455617
8   8 -0.110217865 -0.96965285
9   9  0.077961005  1.65329231
10 10  0.724992375 -1.94993964
```

ANALYSIS STRUCTURE ALGORITHM PROCESS PROGRAMMING SOLVING KNOWLEDGE

IF YOU HAVE MULTIPLE DATA FRAMES

```r
df1 <- data.frame(id = sample(1:10), x = rnorm(10))
df2 <- data.frame(id = sample(1:10), y = rnorm(10))
df3 <- data.frame(id = sample(1:10), z = rnorm(10))
dfList <- list(df1, df2, df3)
```

IF YOU HAVE MULTIPLE DATA FRAMES

```
> join_all(dfList)
Joining by: id
Joining by: id
   id          x          y           z
1   6  1.3859707 -2.3950090  1.45381210
2   4 -0.5372697  2.2485589  0.58109302
3  10 -0.3127759  0.4397997 -0.01423728
4   7 -1.0128262 -0.1101474 -0.35719004
5   1 -0.5174372  0.1965517  0.67689562
6   9  0.5188877 -0.5958733 -0.02668022
7   5 -1.0822557 -2.3850438 -0.23531571
8   8 -0.3802740  0.2729542 -0.40372999
9   3  0.6611610  0.4494388  0.66859383
10  2  0.3689558 -0.8623506 -0.38679770
```

MORE ON MERGING DATA

- The quick R data merging page
  https://www.statmethods.net/management/merging.html

- plyr information
  http://plyr.had.co.nz/