FDA Work Example

Shawn Garbett

2020-05-07

FDA Table Tranforms using tangram

This is an example project showing how tangram can meet the needs of a fictional FDA report. It's based on some actual work that is currently embargoed. From clinical trial data many tables of various summaries are required. So the goal is to demonstrate the work required to create a series of consistent tables using tangram for a project.

This example shows a few interesting things:

- One can add *any* additional arguments to a tangram transform. The point of this library is flexibility and extensibility.
- Further, the statistical tests are *not* defined by the library. This is being provided by the user.
- It doesn't use the more targeted `cell_*` functions like `cell_chi_test`, since the formatting of these was not the desired.
- This shows the need for some additional care around P-values. Fancy display of p-values is complicated and since this is targetted at statistical users I'm going to spend some time on this.
- It's not a simple single table transform, and there are really 3 subcategories of table information that produce the full table.
- Short cut in bundle definition is demonstrated in which the column type is ignored since in this case all columns specified are factors. This was known a priori based on the incoming data.
- Full passing of formatting information is not done as this wasn't required.

This work will most likely evolve into a tranform bundle inside the library that is available for anyone to use. To do this requires more work and consideration of every possible cross product of types that could be thrown at this transform and proper treatment of format specifiers. For the current effort this is not required and the format of the data/transform is constrained heavily by the problem at hand.

I hope that over time transforms that are useful to a broader audience get defined and encourage submissions to add to this library.

Random Data

First some random data to work with (real data is confidential).

```
# Make up some data
N    <- 10000
d1   <- data.frame(
  id        = 1:N,
  procedure = sample(c("A","B","C","D","E","F","G",NA),
                     N,
                     replace=TRUE,
                     prob=c(rep(0.14,7), 0.02)),
  category  = sample(c("D","E", NA),     N, replace=TRUE, prob=c(0.49, 0.49, 0.02)),
  prior     = pmin(rpois(N, 1), 5),
  modality  = sample(c("X","Y","Z", NA), N, replace=TRUE, prob=c(0.33, 0.33, 0.33, 0.01)),
```

```
  albumin   = rnorm(N, 3.5, 0.4)
)

map_procedure_cat <- c(
  "Incisional",
  "Parastomal",
  "Incisional and Parastomal",
  "Epigastric (primary hernia)",
  "Umbilical (primary hernia)",
  "Spigelian (primary hernia)",
  "Lumbar (primary hernia)"
)

d1$prior            <- factor(d1$prior, levels=0:5, labels=c("0", "1", "2", "3", "4", "5+"))

d1$procedure        <- factor(d1$procedure, labels=map_procedure_cat)
label(d1$prior)     <- "Number of prior hernia repairs (among recurrent)"
label(d1$category)  <- "Primary or recurrent"
label(d1$procedure) <- "Ventral hernia procedure"

d1$albumin[sample(1:N,100)] <- NA
label(d1$albumin)   <- "Albumin"
units(d1$albumin)   <- "g/dL"


# Add a binary coded side effect variable
d1$side_effect     <- sample(c(TRUE, FALSE, NA), N, replace=TRUE, prob=c(0.49, 0.49, 0.02))
d1$reported_side_effects <- sample(1:256, N, replace=TRUE)
d1$reported_side_effects[!d1$side_effect] <- NA
label(d1$reported_side_effects) <- "Reported Side Effects"
```

A function that performs the appropriate statistical test and returns a table cell is very helpful. In this case determination of the appropriate ^{2} test over pairs of columns in a grid of numbers.

```
chiTests <- function(grid)
{
  lapply(2:dim(grid)[2], FUN=function(i){
    consider <- grid[,c(1, i)]
    if(min(consider) >= 1)
    {
      test   <- suppressWarnings(chisq.test(consider, correct=FALSE))
      stat   <- unname(test$statistic) * (sum(consider)-1) / sum(consider)
      cell(render_f(pchisq(stat, test$parameter, lower.tail=FALSE), 3), reference=1)
    }
    else
    {
      cell(render_f(fisher.test(consider, simulate.p.value=TRUE, B=1e7)$p.value, 3), reference=2)
    }
  })
}
```

In this example we are only considering N categories as row and M categories as column. The resulting table is (N+1) X (2M). More specifically a row for the name of the row variable and row for each category present in the row. There is a column for the count (percentage) of each column variable, then pair-wise

comparisons with the first category following by the count of missing. Statistical tests only appear in the first row.

The additional logical argument `display_percent` is specified to turn on and off the display of percents. By default it's TRUE and additional arguments passed to `tangram` are pushed down into these transforms so one is free to define *any* additional variables being passed in and out of transforms.

Further this example seeks to avoid use of the `%>%` operator for instructional purposes, unlike the original example of using table_builder operators.

```
fda_cat_by_cat <- function(tb, row, col, display_percent=TRUE, ...)
{
  grid   <- table(row$data, col$data)

  tests  <- chiTests(grid)
  colN   <- lapply(colnames(grid), function(cat) cell_n(sum(grid[,cat]), subcol=cat))
  rowlbl <- lapply(rownames(grid), function(x) paste("  ", x))
  versus <- paste(colnames(grid)[1], "vs.", colnames(grid)[2:length(colnames(grid))])

  # Now construct the table by add rows to each column
  tb <- col_header(tb, colnames(grid), versus, "Missing")
  tb <- col_header(tb, colN, rep("P-value", length(versus)), "")
  tb <- row_header(tb, derive_label(row))

  for(nm in rowlbl) tb <- row_header(tb, nm)

  for(colnm in colnames(grid))
  {
    denom <- sum(grid[,colnm])
    tb <- add_row(tb, "")

    for(rownm in rownames(grid))
    {
      numer <- grid[rownm, colnm]
      x     <- if(display_percent) paste0(numer, " (", render_f(100*numer/denom, 1), ")") else
                                  as.character(numer)

      tb    <- add_row(tb, cell(x, subcol = colnm, subrow = rownm))
    }

    tb <- new_col(tb)
  }

  tb <- add_col(tb, tests)
  tb <- add_col(tb, length(row$data)-sum(grid))
}
```

Using this any variables that are factors can be used now to generate a table and render to HTML5.

```
tangram("modality ~ procedure + category + prior",
        d1, "tbl1", caption="FDA Table 1",
        transforms=fda_cat_by_cat,
        style="nejm",
        relsize=-1)
```

Table 1: FDA Table 1

Table 1: FDA Table 1

	X	Y	Z	X vs. Y	X vs. Z	Missing
	3297	3294	3151	P-value	P-value	
Ventral hernia procedure				0.256	0.065	258
Incisional	509 (15.4)	487 (14.8)	436 (13.8)			
Parastomal	414 (12.6)	460 (14.0)	463 (14.7)			
Incisional and Parastomal	472 (14.3)	467 (14.2)	474 (15.0)			
Epigastric (primary hernia)	536 (16.3)	472 (14.3)	461 (14.6)			
Umbilical (primary hernia)	450 (13.6)	473 (14.4)	424 (13.5)			
Spigelian (primary hernia)	464 (14.1)	472 (14.3)	446 (14.2)			
Lumbar (primary hernia)	452 (13.7)	463 (14.1)	447 (14.2)			
Primary or recurrent				0.145	0.305	284
D	1619 (49.5)	1683 (51.2)	1602 (50.7)			
E	1655 (50.5)	1601 (48.8)	1556 (49.3)			
Number of prior hernia repairs (among recurrent)				0.818	0.004	85
0	1211 (36.2)	1239 (37.0)	1205 (37.4)			
1	1235 (36.9)	1226 (36.6)	1178 (36.5)			
2	652 (19.5)	629 (18.8)	560 (17.4)			
3	171 (5.1)	189 (5.6)	225 (7.0)			
4	61 (1.8)	53 (1.6)	42 (1.3)			
5+	13 (0.4)	13 (0.4)	13 (0.4)			

Next it is necessary to allow for row variables that are continuous. We begin with the helper function that creates cells for the tests given the data for a row (x) and column (y). In this case we make no distribution assumption about the continuous variable and apply a Wilcoxon rank sum test.

```
wilcoxTests <- function(x, y)
{
  lvls <- levels(y)

  lapply(2:length(lvls), FUN=function(i){
    test <- wilcox.test(x[y==lvls[1]], x[y==lvls[i]])
    cell(render_f(test$p.value, 3), reference=3)
  })
}
```

Similarly we create a table builder for continuous by M category summaries. The resulting table is (4) X (2M). There is a row for the row variable name, and the mean, median and standard deviation. Column's are the same as above.

```
fda_cont_by_cat <- function(tb, row, col, ...)
{
  datar          <- row$data
  datac          <- col$data

  lvls           <- levels(datac)

  colN   <- lapply(lvls, function(cat)
```

```
    cell_n(length(datac[datac == cat & !is.na(datac)]), subcol=cat))
  versus <- paste(lvls[1], "vs.", lvls[2:length(lvls)])

  # Now construct the table by add rows to each column
  tb <- col_header(tb, lvls, versus, "Missing")
  tb <- col_header(tb, colN, rep("P-value", length(versus)), "")
  tb <- row_header(tb, derive_label(row))

  for(nm in c("Mean", "Median", "SD")) tb <- row_header(tb, paste0("  ",nm))

  # Summary
  for(colnm in lvls)
  {
    d     <- datar[datac == colnm & !is.na(datac)]
    tb    <- add_row(tb, "")
    tb    <- add_row(tb, render_f(mean(d, na.rm=TRUE),   row$format))
    tb    <- add_row(tb, render_f(median(d, na.rm=TRUE), row$format))
    tb    <- add_row(tb, render_f(sd(d, na.rm=TRUE),     row$format))

    tb    <- new_col(tb)
  }

  # Tests
  tests <- wilcoxTests(datar, datac)
  tb <- add_col(tb, tests)
  tb <- add_col(tb, length(datar)-sum(!is.na(datar) & !is.na(datac)))

  tb
}
```

This step bundles the two together and based on type of variable decides which transform to apply. We use the hmisc type determination function as a quick guide. Note that some transforms are unsupported as we there was no requirement to provide those cross product tables of variables.

Further we add some descriptive footnotes.

```
unsupported <- function(tb, row, col) stop("unsupported type", row$value, "X", col$value)
fda <- list(
  Type = hmisc_data_type,
  Numerical   = list(
                  Numerical   = unsupported,
                  Categorical = fda_cont_by_cat
            ),
  Categorical = list(
                  Numerical   = unsupported,
                  Categorical = fda_cat_by_cat
            ),
  Footnote    = "Count (Percent) format. ^1^ ^2^ minus one. ^2^ Fisher exact. ^3^ Wilcoxon rank sum"
)
```

Now a rendering with two forms of information is possible.

```
tangram(modality ~ procedure + category + prior + albumin,
        d1, "tbl2", caption="FDA Table 2", style="nejm",
        transforms=fda,
```

```
        relsize=-1)
```

Table 2: FDA Table 2

Table 2: FDA Table 2						
	X	**Y**	**Z**	**X vs. Y**	**X vs. Z**	**Missing**
	3297	3294	3151	P-value	P-value	
Ventral hernia procedure				0.256	0.065	258
Incisional	509 (15.4)	487 (14.8)	436 (13.8)			
Parastomal	414 (12.6)	460 (14.0)	463 (14.7)			
Incisional and Parastomal	472 (14.3)	467 (14.2)	474 (15.0)			
Epigastric (primary hernia)	536 (16.3)	472 (14.3)	461 (14.6)			
Umbilical (primary hernia)	450 (13.6)	473 (14.4)	424 (13.5)			
Spigelian (primary hernia)	464 (14.1)	472 (14.3)	446 (14.2)			
Lumbar (primary hernia)	452 (13.7)	463 (14.1)	447 (14.2)			
Primary or recurrent				0.145	0.305	284
D	1619 (49.5)	1683 (51.2)	1602 (50.7)			
E	1655 (50.5)	1601 (48.8)	1556 (49.3)			
Number of prior hernia repairs (among recurrent)				0.818	0.004	85
0	1211 (36.2)	1239 (37.0)	1205 (37.4)			
1	1235 (36.9)	1226 (36.6)	1178 (36.5)			
2	652 (19.5)	629 (18.8)	560 (17.4)			
3	171 (5.1)	189 (5.6)	225 (7.0)			
4	61 (1.8)	53 (1.6)	42 (1.3)			
5+	13 (0.4)	13 (0.4)	13 (0.4)			
Albumin ***g/dL***				0.535	0.380	183
Mean	3.50	3.49	3.51			
Median	3.50	3.50	3.51			
SD	0.40	0.40	0.40			

Count (Percent) format. [1] [2] minus one. [2] Fisher exact. [3] Wilcoxon rank sum

A tricky binary coded varible for reported side effects needs treatment. In this instance we only want the category in which side effects appeary, i.e. only those individuals with side effects is to be reported. The variable contains a binary number in which each bit represents a different side effect reported.

I have chosen to handle this in the formula syntax with the * operator for now. I have debated adding the traditional | denoting nested models to the formula syntax, but at present even handling the * properly is complicated and incomplete.

Secondly as mentioned above additional variables are passed down to the transform which can make use of them. This is useful now for passing in a binary transform table (but it would be used for all transforms if multiple existed in a table, further refinement of list of lists could be used if needed).

The basic approach is to expand the data into a long form, then pass to original cat X cat function using the display_percent logical to turn that off.

```
side_effect_key = list(
  "Repetative Uttering of Wut?",
  "Excessive Sweating",
  "Hairy Navel",
  "Breaking Voice",
  "Beiber Fever",
  "Swiftaphila",
  "Akward Elbows",
  "Veruca"
)
```

```
fda_binary <- function(tb, row, col, binary_key=list(), ...)
{
  inside    <- row$right$data   # Grouped inside the right hand side of '*' assuming logical
  inside[is.na(inside)] <- FALSE
  datar     <- row$left$data[inside]  # Data to further group
  datac     <- col$data[inside]

  # Expand for counting
  x         <- rep(datar, each=length(binary_key))
  y         <- rep(datac, each=length(binary_key))
  key       <- rep(1:length(binary_key), length(datar))
  present   <- bitwAnd(x, 2^(key-1)) > 0

  # Filter down
  x         <- factor(sapply(key[present], function(x) binary_key[[x]]))
  y         <- y[present]

  rname     <- paste0(row$left$name(), " N=", sum(inside))

  fda_cat_by_cat(tb, list(data=x, name=function() rname), list(data=y, name=col$name),
                 display_percent=FALSE)
}

fda_data_type <- function(x, category_threshold=NA)
{
  if(is.categorical(x,category_threshold))  "Categorical" else
  if(is.numeric(x))                         "Numerical"   else
  stop(paste("Unsupported class/type - ",class(x), typeof(x)))
}

# Note the second dimension isn't present, it only determines function call by type of Row
# If provided a list of types to functions for each argument a cross product of types
# determines the functional transform. But this is a nice short cut provided.
fda <- list(
  Type        = fda_data_type,
  Numerical   = fda_cont_by_cat,
  Categorical = fda_cat_by_cat,
  ASTMultiply = fda_binary,
  Footnote    = "Count (Percent) format. ^1^ ^2^ minus one. ^2^ Fisher exact. ^3^ Wilcoxon rank sum"
)
```

Now we have 3 different pieces completed.

```
tangram(modality ~ procedure + category + prior + albumin + reported_side_effects*side_effect,
        d1, "tbl3", transforms=fda,  binary_key=side_effect_key,
        style="nejm", caption="FDA Table 3",
        relsize=-1)
```

Table 3: FDA Table 3

Table 3: FDA Table 3

	X	Y	Z	X vs. Y	X vs. Z	Missing
	3297	3294	3151	P-value	P-value	
Ventral hernia procedure				0.256	0.065	258
Incisional	509 (15.4)	487 (14.8)	436 (13.8)			
Parastomal	414 (12.6)	460 (14.0)	463 (14.7)			
Incisional and Parastomal	472 (14.3)	467 (14.2)	474 (15.0)			
Epigastric (primary hernia)	536 (16.3)	472 (14.3)	461 (14.6)			
Umbilical (primary hernia)	450 (13.6)	473 (14.4)	424 (13.5)			
Spigelian (primary hernia)	464 (14.1)	472 (14.3)	446 (14.2)			
Lumbar (primary hernia)	452 (13.7)	463 (14.1)	447 (14.2)			
Primary or recurrent				0.145	0.305	284
D	1619 (49.5)	1683 (51.2)	1602 (50.7)			
E	1655 (50.5)	1601 (48.8)	1556 (49.3)			
Number of prior hernia repairs (among recurrent)				0.818	0.004	85
0	1211 (36.2)	1239 (37.0)	1205 (37.4)			
1	1235 (36.9)	1226 (36.6)	1178 (36.5)			
2	652 (19.5)	629 (18.8)	560 (17.4)			
3	171 (5.1)	189 (5.6)	225 (7.0)			
4	61 (1.8)	53 (1.6)	42 (1.3)			
5+	13 (0.4)	13 (0.4)	13 (0.4)			
Albumin ***g/dL***				0.535	0.380	183
Mean	3.50	3.49	3.51			
Median	3.50	3.50	3.51			
SD	0.40	0.40	0.40			
Reported Side Effects N=4830				0.818	0.786	180
Akward Elbows	784	801	745			
Beiber Fever	813	808	746			
Breaking Voice	788	828	776			
Excessive Sweating	833	799	766			
Hairy Navel	805	815	740			
Repetative Uttering of Wut?	840	818	820			
Swiftaphila	797	802	769			
Veruca	852	802	752			

Count (Percent) format. [1] [2] minus one. [2] Fisher exact. [3] Wilcoxon rank sum

And the requested table tranforms for FDA work is complete.

Here is the final version of the code:

```
  ##################################################################
 # Statistical tests as table cells
#
chiTests <- function(grid)
{
  lapply(2:dim(grid)[2], FUN=function(i){
    consider <- grid[,c(1, i)]
    if(min(consider) >= 1)
    {
```

```
      test   <- suppressWarnings(chisq.test(consider, correct=FALSE))
      stat   <- unname(test$statistic) * (sum(consider)-1) / sum(consider)
      cell(render_f(pchisq(stat, test$parameter, lower.tail=FALSE), 3), reference=1)
    }
    else
    {
      cell(render_f(fisher.test(consider, simulate.p.value=TRUE, B=1e7)$p.value, 3), reference=2)
    }
  })
}

wilcoxTests <- function(x, y)
{
  lvls <- levels(y)

  lapply(2:length(lvls), FUN=function(i){
    test <- wilcox.test(x[y==lvls[1]], x[y==lvls[i]])
    cell(render_f(test$p.value, 3), reference=3)
  })
}

  ##################################################################
 # Row/Column from abstract syntax tree transforms to tables
#
fda_cat_by_cat <- function(tb, row, col, display_percent=TRUE, ...)
{
  grid   <- table(row$data, col$data)

  tests  <- chiTests(grid)
  colN   <- lapply(colnames(grid), function(cat) cell_n(sum(grid[,cat]), subcol=cat))
  rowlbl <- lapply(rownames(grid), function(x) paste("  ", x))
  versus <- paste(colnames(grid)[1], "vs.", colnames(grid)[2:length(colnames(grid))])

  # Now construct the table by add rows to each column
  tb <- col_header(tb, colnames(grid), versus, "Missing")
  tb <- col_header(tb, colN, rep("P-value", length(versus)), "")
  tb <- row_header(tb, derive_label(row))

  for(nm in rowlbl) tb <- row_header(tb, nm)

  for(colnm in colnames(grid))
  {
    denom <- sum(grid[,colnm])
    tb <- add_row(tb, "")

    for(rownm in rownames(grid))
    {
      numer <- grid[rownm, colnm]
      x     <- if(display_percent) paste0(numer, " (", render_f(100*numer/denom, 1), ")") else
                                  as.character(numer)

      tb    <- add_row(tb, cell(x, subcol = colnm, subrow = rownm))
    }
```

```
    tb <- new_col(tb)
  }

  tb <- add_col(tb, tests)
  tb <- add_col(tb, length(row$data)-sum(grid))
}

fda_cont_by_cat <- function(tb, row, col, ...)
{
  datar          <- row$data
  datac          <- col$data

  lvls           <- levels(datac)

  colN   <- lapply(lvls, function(cat)
    cell_n(length(datac[datac == cat & !is.na(datac)]), subcol=cat))
  versus <- paste(lvls[1], "vs.", lvls[2:length(lvls)])

  # Now construct the table by add rows to each column
  tb <- col_header(tb, lvls, versus, "Missing")
  tb <- col_header(tb, colN, rep("P-value", length(versus)), "")
  tb <- row_header(tb, derive_label(row))

  for(nm in c("Mean", "Median", "SD")) tb <- row_header(tb, paste0("  ",nm))

  # Summary
  for(colnm in lvls)
  {
    d     <- datar[datac == colnm & !is.na(datac)]
    tb    <- add_row(tb, "")
    tb    <- add_row(tb, render_f(mean(d, na.rm=TRUE),   row$format))
    tb    <- add_row(tb, render_f(median(d, na.rm=TRUE), row$format))
    tb    <- add_row(tb, render_f(sd(d, na.rm=TRUE),     row$format))

    tb    <- new_col(tb)
  }

  # Tests
  tests <- wilcoxTests(datar, datac)
  tb <- add_col(tb, tests)
  tb <- add_col(tb, length(datar)-sum(!is.na(datar) & !is.na(datac)))

  tb
}

fda_binary <- function(tb, row, col, binary_key=list(), ...)
{
  inside    <- row$right$data   # Grouped inside the right hand side of '*' assuming logical
  inside[is.na(inside)] <- FALSE
  datar     <- row$left$data[inside]  # Data to further group
  datac     <- col$data[inside]

  # Expand for counting
```

```
  x         <- rep(datar, each=length(binary_key))
  y         <- rep(datac, each=length(binary_key))
  key       <- rep(1:length(binary_key), length(datar))
  present   <- bitwAnd(x, 2^(key-1)) > 0

  # Filter down
  x         <- factor(sapply(key[present], function(x) binary_key[[x]]))
  y         <- y[present]

  rname     <- paste0(row$left$name(), " N=", sum(inside))

  fda_cat_by_cat(tb, list(data=x, name=function() rname), list(data=y, name=col$name),
                 display_percent=FALSE)
}

  ################################################################
 # Data typing function to use with above
#
fda_data_type <- function(x, category_threshold=NA)
{
  if(is.categorical(x,category_threshold))  "Categorical" else
  if(is.numeric(x))                         "Numerical"   else
  stop(paste("Unsupported class/type - ",class(x), typeof(x)))
}

  ################################################################
 #
# Bring it all together into a useable bundle of transforms
fda <- list(
  Type        = fda_data_type,
  Numerical   = fda_cont_by_cat,
  Categorical = fda_cat_by_cat,
  ASTMultiply = fda_binary,
  Footnote    = "Count (Percent) format. ^1^ ^2^ minus one. ^2^ Fisher exact. ^3^ Wilcoxon rank sum"
)
```

IRR Tables

For some DSMB reports to the FDA I created incident relative ratios transforms and they've been working great.

The problem here is that we need the number of events and the exposure and a group variable. I chose to use the * operator in the following format `group ~ exposure*(event1+event2)` for my formula. The * distributes before executing the transforms and results in `group ~ exposure*event1 + exposure*event2` calling the transform twice once for each event.

Let's create some example data. A mixed group with a period of observation recording observed burps, and rude comments. In this set, the males have a higher rate of public burping.

```
N <- 1000
manners <- data.frame(gender  =sample(c("M", "F"), N, replace=TRUE),
                      observed=rexp(N)/365,
                      rude    =rpois(N, 0.1))

manners$burps <- rpois(N, 0.1) + ifelse(manners$gender == "M", rpois(N, 0.2), 0)
```

Now, a reusable transform for this purpose is created. Eventually this will become a core transform available to any user, but it requires more work to handle different possibilities of data sets. This example is targetted towards knowing exactly the form of the data coming in.

```
# A custom transform to create IRR tables from exposure / event data.
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:Hmisc':
##
##     src, summarize
```

```
## The following object is masked from 'package:tangram':
##
##     add_row
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
irr <- function(table, row, column, ...)
{
    events   <- row$right$data
    exposure <- row$left$data

    data     <- data.frame(group  = column$data,
                           events = events,
                           expose = exposure)

    data     <- subset(data, !is.na(events) & !is.na(expose) & expose > 0)
    est      <- "-"
    pval     <- "-"

    tryCatch(
    {
      # Now put in epitools format
      etd <- data %>% group_by(group) %>% summarize(
                incidents = sum(events),
                exposure  = sum(expose)
              )
      # Run midp analysis using epitools
      x <- rateratio.midp(etd$incidents, etd$exposure)

      fmt <- if(x$measure[2,1] < 1000) 2 else "%5.3g"

      est <- paste0(render_f(x$measure[2,1], fmt), " (",
                    render_f(x$measure[2,2], fmt), "--",
                    render_f(x$measure[2,3], fmt), ")")
      pval <- if(x$p.value[2,1] <= 0.0005) "<0.001" else render_f(x$p.value[2,1], 3)
    },
    error = function(e) {}
```

```
    )


    table %>%
    col_header("N", "At Least 1", "Total",
               "At-Risk Time", "Incidence Rate", "Incidence Rate Ratio",
               "P Value^1^") %>%
    col_header("", "no. of participants", "no.",
               "person-yr", "events/100 person-yr", "", "") %>%
    row_header(derive_label(row$right)) %>%
    add_col("", "", "", "", "") %>%
    add_col(est, pval) %>%
    table_apply(levels(data$group), function(tbl, category)
    {
        selector  <- data$group == category
        #n         <- sum(column$data == category, na.rm=TRUE)
        n         <- length(data$group[selector])
        total_exp <- sum(data$expose[selector])
        total_evt <- sum(data$events[selector])

        tbl %>%
        row_header(paste0("  ", category)) %>%
        new_row() %>%
        add_col(if(total_exp>0) n else "-",
                if(total_exp>0) sum(data$events[selector] > 0) else "-",
                if(total_exp>0) sum(data$events[selector]) else "-",
                if(total_exp>0) render_f(total_exp*100, 2) else "-",
                if(total_exp>0) render_f(total_evt/total_exp,2) else "-") %>%
        add_col("", "")
    })
}

irr.footnote="^1^IRR and *p*-values calculated by the number of adverse events over exposure via median-
```

And for the result:

```
tangram(gender ~ observed*(burps + rude),
        manners, "tbl4", transform=irr,
        style="nejm", footnote=irr.footnote, caption="FDA IRR Table")
```

Table 4: FDA IRR Table

Table 4: FDA IRR Table

	N	At Least 1	Total	At-Risk Time	Incidence Rate	Incidence Rate Ratio	P Value[1]
		no. of participants	no.	person-yr	events/100 person-yr		
burps						4.21 (3.00–6.08)	<0.001
F	487	34	38	131.89	28.81		
M	513	137	170	139.64	121.75		
rude						0.92 (0.61–1.40)	0.709
F	487	42	45	131.89	34.12		
M	513	43	44	139.64	31.51		

[1]IRR and *p*-values calculated by the number of adverse events over exposure via median-unbiased estimation (mid-p Rothman 2008).

Summary

This is now a reusable IRR table generator that calculates rate ratio by median-unbiased estimation (mid-p). See Kenneth J. Rothman, Sander Greenland, and Timothy Lash (2008), Modern Epidemiology, Lippincott-Raven Publishers.

A personal or department library of preferred analysis and tables can easily be shared via the tangram interface, creating consistent and repeatble analysis across many reports. More importantly the analysis and the final format rendering are completely separate choices.