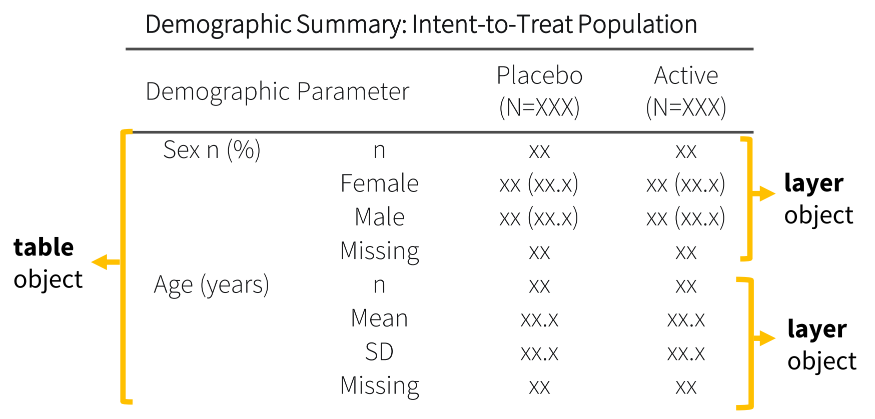
Building Clinical Safety Summaries with Tplyr: : **CHEAT SHEET**

```
Tplyr
row_label2  var1_Placebo
<chr>       <chr>
<65         14 (  16.3%)
>80         25 (  29.1%)
65-80       42 (  48.8%)
            86 (100.0%)
```

'Tplyr' contains intuitive functions that build upon one another to create summary tables, which eliminates the redundancy of programming all while remaining flexible enough to conform to varying standards.

Table Components



Demographic Summary: Intent-to-Treat Population

Demographic Parameter		Placebo (N=XXX)	Active (N=XXX)
Sex n (%)	n	xx	xx
	Female	xx (xx.x)	xx (xx.x)
	Male	xx (xx.x)	xx (xx.x)
	Missing	xx	xx
Age (years)	n	xx	xx
	Mean	xx.x	xx.x
	SD	xx.x	xx.x
	Missing	xx	xx

The output of **tplyr_layer**() objects will be stacked to create the **tplyr_table**() object.

Table Level Settings

TABLE FUNCTION

tplyr_table(target, treat_var, where=TRUE, cols=vars()) - applies logic at the table level. *t <- tplyr_table(adsl, TRT01P, where=SAFFL=='Y', cols=RACE)*

Parameter	Description
target	dataset used to perform summaries
treat_var	variable used to distinguish treatment groups
where=	subset applied to table level
cols=	grouping variable(s) used to create columns on the display (Note: this is in addition to treat_var)

ADDING TREATMENT GROUPS

add_treat_grps(table, …) – Create new treatment groups by combining existing treatment groups from the values within treat_var. *add_treat_grps(t, 'Treated'=c("High", "Low"))*

add_total_group(table, group_name="Total") - Abstraction of add_treat_grps() to create a group for total. *add_total_group(t)*

ADDING A POPULATION DATASET

If target does not include the entire necessary population, the **population functions** can provide population information.

set_pop_data(table, pop_data) - Specifies a population dataset. *set_pop_data(t, adsl)*

set_pop_treat_var(table, pop_treat_var) - Specifies a treatment variable from the population dataset. If not called, uses treat_var from tplyr_table(). *set_pop_treat_var(t, TRT01A)*

set_pop_where(obj, where) - Specifies a population subset. If not called, uses where= from tplyr_table(). *set_pop_where(t, SAFFL=="Y")*

Building the Table

ADDING LAYERS TO A TABLE

add_layer(parent, layer, name=NULL) Constructs the layer within the call to the function.
add_layer(t, group_count(SEX, by="Sex n (%)"))

Parameter	Description
parent	the tplyr_table() object
layer	contains the group_type() function call and any modifier functions to create the layer
name=	specifies the layers name within the tplyr_table() object's layer container

add_layers(parent, …) Attaches layers that have already been constructed. *add_layers(t, l1, l2)*

Parameter	Description
parent	the tplyr_table() object
…	specifies the layer objects that will be attached to the tplyr_table() object

PROCESSING THE DATA

Constructing a tplyr_table() object or a tplyr_layer() object constructs the metadata necessary to generate a table but does not process the actual data. To generate the data and perform the summaries use the **build**() function.

t %>%
build()

Layer Level Settings

group_<type>(parent, target_var, by=vars(), where=TRUE, …) - family of functions used to create layers.

The types of layers are count, shift, and desc (descriptive statistics).

Parameter	Description
parent	the tplyr_table() object
target_var	variable(s) on which the summary is performed
by=	variable(s) or value(s) used as grouping variable(s) and represented as row label(s)
where=	subset applied to layer level (Note: this is in addition to any subset applied at the table level)

COUNT LAYERS

group_count() - Specifies that a layer will be created to count occurrences and/or their proportions.

Sex n (%)	F	53 (61.6%)
	M	33 (38.4%)

group_count(t, SEX, by="Sex n (%)")

When calculating **nested counts** use dplyr::vars() to specify 2 variables for target_var.

SOC 1		21 (24.4%)
	AE 1	13 (15.1%)
	AE 2	8 (9.3%)

group_count(t, vars(AEBODSYS,AEDECOD))

CALCULATING PERCENTAGES

set_denoms_by(x, …) - Specifies variable(s) to use to calculate percentages. If not called, uses treat_var and cols= from tplyr_table(). *set_denoms_by(x, TRTA, PARAM, AVISIT)*

set_denom_ignore(e, …) - Specifies values of target_var to exclude from percentage calculation. *set_denom_ignore(e, "NA")*

DISTINCT VS EVENT COUNTS

set_distinct_by(e, distinct_by) - Specifies variable(s) to use to calculate distinct occurrences. *set_distinct_by(e, USUBJID)*

ADDING A 'TOTAL' ROW

Sex n (%)	F	53 (61.6%)
	M	33 (38.4%)
	Total	86 (100.0%)

add_total_row(e) - Adds a row with the total count within by= from group_<type>() and treat_var and cols= from tplyr_table(). *add_total_row(e)*

Sex n (%)	F	53 (61.6%)
	M	33 (38.4%)
	All	86 (100.0%)

set_total_row_label(e, total_row_label) - Specifies a row label for the total row. If not called, default text will be "Total". *set_total_row_label(e, "All")*

atorus
Your Complete Data Solution

Building Clinical Safety Summaries with Tplyr: : CHEAT SHEET

COUNT LAYERS (CONTINUED)

SETTING FORMATTING

set_format_strings() and **f_str()** are used to specify the occurrence and proportion variables and how they will be presented. The user uses x's to specify how the numbers will be displayed.

SOC 1		21 (24.4%) [26]
	AE 1	13 (15.1%) [15]
	AE 2	8 (9.3%) [11]

set_format_strings(e, f_str('xx (xx.x%) [xx]', distinct, distinct_pct,n))

Sex n (%)	F	53 : 61.6%
	M	33 : 38.4%

set_format_strings(e, f_str('xx : xx.x%', n, pct))

MISSING COUNTS PRESENTATION

HIGH	12 (38.7%)
LOW	17 (54.8%)
MISSING	2

set_missing_count(e, f_str, string="NA") - Sets the display of missing values.
set_missing_count(t, f_str('xx', n), string=c(MISSING="NA")

SHIFT LAYERS

group_shift() - Specifies a shift layer will be created to count occurrences and their proportions from one state to another.

PARAM	**VISIT**		L	N	H
PARAM 1	VISIT 1	L	0	0	1
		N	3	12	0
		H	0	7	2

group_shift(t, vars(row=BNRIND, column=ANRIND), by=vars(PARAM,AVISIT))

group_shift() is largely an abstraction of a count layer. The function can be used with **set_denoms_by()**, **set_format_strings()**, and **f_str()**.

DESCRIPTIVE STATISTICS LAYERS

group_desc() - Specifies a layer will be created to perform summaries on continuous variables.

Age (years)	n	86
	Mean (SD)	75.2 (8.59)
	Median	76.0
	Q1, Q3	69.2, 81.8
	Min, Max	52, 89
	Missing	0

group_desc(t, AGE, by="Age (years)"))

CUSTOM SUMMARIES

Custom summaries allow any function to be used in a descriptive statistics layer.

set_custom_summaries(e, …) – Allows user to define custom summaries that will be performed in dplyr::summarize. Use .var as the variable name being summarized.
set_custom_summaries(geo_mean=exp(sum(log(.var[.var>0]),na.rm=TRUE/length(.var)))(e)

BUILT-IN SUMMARIES

Description	Variable Name
N	n
Mean	mean
Standard Deviation	sd
Median	median
Variance	variance
Minimum	min
Maximum	max
Interquartile Range	iqr
Q1	q1
Q3	q3
Missing	missing

FORMATTING AND PERFORMING SUMMARIES

set_format_strings() and **f_str()** are used to specify the summaries that will be performed and how they will be presented.

n	86
Mean (SD)	75.21 (8.590)
Q1	69.2
Q3	81.8

set_format_strings(e,
"n" = f_str("xx", n),
"Mean (SD)" = f_str("xx.xx, (xx.xxx)", mean, sd)
"Q1" = f_str(xx.x, q1)
"Q3" = f_str(xx.x, q3))

- On the left side of the equal sign the user inputs text that becomes the row label.
- On the right side the user uses x's to specify how the numbers will be displayed and lists the descriptive statistic summaries that will be performed.

The empty parameter of **f_str()** specifies what to display if an element or elements in a cell produced NA values.

n	1
SD	NA

set_format_strings(e,
"n" = f_str("xx", n),
"SD" = f_str("xx.xx", sd, empty="NA")

AUTO PRECISION

Auto precision is used to format numeric summaries based on the precision of the data collected.

set_format_strings(
'Mean (SD)'=f_str('a.a+1 (a.a+2)',mean,sd), cap=c(int=3,dec=2))

- Use a instead of x (only 1 a is needed on each side of the decimal)
- Use a+n where n is the number of additional spaces you wish to add
- Use the cap parameter to cap the length allotted for integers and decimals

Sorting

Ordering helpers are columns added into 'Tplyr' tables.

SORTING THE LAYERS

Layers are indexed using the variable **ord_layer_index** by the order in which they were added to the table using add_layer() or add_layers().

SORTING THE BY VARIABLES

Each by variable gets an ord_layer_<n> column. The order variables will calculate based on the first applicable method:

- Use a matching variable name suffixed by *N* from the dataset if available (i.e. RACE and RACE*N*)
- Use factor levels if variable is a factor
- Use alphanumeric sorting of variable values

SORTING DESCRIPTIVE STATISTICS LAYER RESULTS

Descriptive statistics layers get an ord_layer_<n> column based on the order in which the f_str() objects are created through set_format_strings()

SORTING COUNT LAYER RESULTS

Count layers get an ord_layer_<n> column based on the sort method specified in **set_order_count_method()**.

set_order_count_method("byfactor") - Use factor levels. If variable is not a factor, alphanumeric sorting will be used. This is the default method and set_order_count_method() does not need to be called.

set_order_count_method("byvarn") - Use a matching variable name suffixed by *N* from the dataset if available (i.e. RACE and RACE*N*)

set_order_count_method("bycount") – Sort based on counts in a particular column. Requires the use of additional helper functions:

- **set_ordering_cols**(e, …) – Specifies the treat_var and cols= value(s) from tplyr_table() to determine the column from which the ordering should be based. *set_ordering_cols("High","WHITE")*
- **set_result_order_var**(e, result_order_var) – Specifies the occurrence or proportion variable on which the ordering should be based. *set_result_order_var(n)*

One More Thing

To get the **underlying raw calculations** the following function is used instead of build().

get_numeric_data(x, layer=NULL, where=TRUE, …) – Provides access to the un-formatted numeric data for each layer.
get_numeric_data(t)