

# SAS <-> R :: CHEAT SHEET

## Introduction

This guide aims to familiarise SAS users with R.  
R examples make use of tidyverse collection of packages.

Install tidyverse: `install.packages("tidyverse")`  
Attach tidyverse packages for use: `library(tidyverse)`

R data here in 'data frames', and occasionally vectors (via `c()`)  
Other R structures (lists, matrices...) are not explored here.

Keyboard shortcuts: `<-` Alt + - `%>%` Ctrl + Shift + m

## Datasets; drop, keep & rename variables

```
data new_data;
  set old_data;
run;

new_data <- old_data
```

```
data new_data (keep=id);
  set old_data (drop=job_title);
run;

new_data <- old_data %>%
  select(-job_title) %>%
  select(id)
```

```
data new_data (drop= temp: );
  set old_data;
run;

new_data <- old_data %>%
  select( -starts_with("temp") )
  C.f. contains( ), ends_with( )
```

```
data new_data;
  set old_data;
  rename old_name = new_name;
run;

new_data <- old_data %>%
  rename(new_name = old_name)
  Note order differs
```

## Conditional filtering

```
data new_data;
  set old_data;
  if Sex = "M";
run;

new_data <- old_data %>%
  filter(Sex == "M")
```

```
data new_data;
  set old_data;
  if year in (2010,2011,2012);
run;

new_data <- old_data %>%
  filter(year %in% c(2010,2011,2012))
```

```
data new_data;
  set old_data;
  by id;
  if first.id;
run;

new_data <- old_data %>%
  group_by( id ) %>%
  slice(1)
  Could use slice(n()) for last
```

```
data new_data;
  set old_data;
  if dob > "25APR1990"d;
run;

new_data <- old_data %>%
  filter(dob > as.Date("1990-04-25"))
```

## New variables, conditional editing

```
data new_data;
  set old_data;
  total_income = wages + benefits;
run;
```

```
new_data <- old_data %>%
  mutate(total_income = wages + benefits)
```

```
data new_data;
  set old_data;
  if hours > 30 then full_time = "Y";
  else full_time = "N";
run;
```

```
new_data <- old_data %>%
  mutate(full_time = if_else(hours > 30, "Y", "N"))
```

```
data new_data;
  set old_data;
  if temp > 20 then weather = "Warm";
  else if temp > 10 then weather = "Mild";
  else weather = "Cold";
run;
```

```
new_data <- old_data %>%
  mutate(weather = case_when(
    temp > 20 ~ "Warm",
    temp > 10 ~ "Mild",
    TRUE ~ "Cold" ))
```

## Counting and Summarising

```
proc freq data = old_data ;
  table job_type ;
run;
```

```
old_data %>%
  count( job_type )
  For percent, add:
  %>% mutate(percent = n*100/sum(n))
```

```
proc freq data = old_data ;
  table job_type*region ;
run;
```

```
old_data %>%
  count( job_type , region )
```

```
proc summary data = old_data nway ;
  class job_type region ;
  output out = new_data ;
run;
```

```
new_data <- old_data %>%
  group_by( job_type , region ) %>%
  summarise( Count = n( ) )
```

Equivalent without nway not trivially produced

```
proc summary data = old_data nway ;
  class job_type region ;
  var salary ;
  output out = new_data
  sum( salary ) = total_salaries ;
run;
```

```
new_data <- old_data %>%
  group_by( job_type , region ) %>%
  summarise( total_salaries = sum( salary ) ,
    Count = n( ) )
```

Lots of summary functions in both languages  
Swap summarise( ) for mutate( ) to add summary data to original data

## Combining datasets

```
data new_data ;
  set data_1 data_2 ;
run;
```

```
new_data <- bind_rows( data_1 , data_2 )
```

C.f. rbind( ) which produces error if columns are not identical

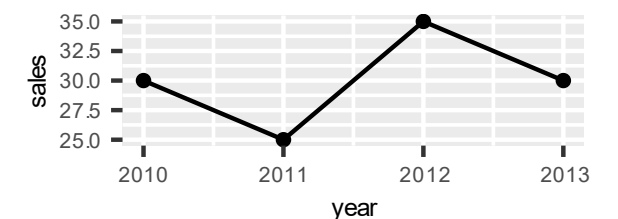
```
data new_data ;
  merge data_1 (in= in_1) data_2 ;
  by id ;
  if in_1 ;
run;
```

```
new_data <- left_join( data_1 , data_2 , by = "id" )
```

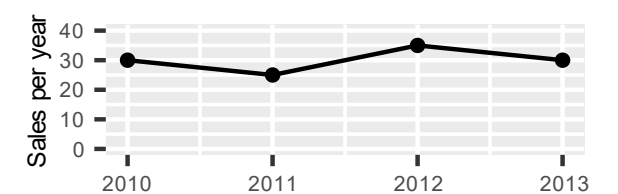
C.f. full\_join( ) , right\_join( ) , inner\_join( )

## Some plotting in R

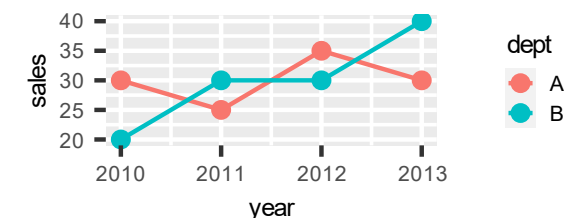
```
ggplot( my_data , aes( year , sales ) ) +
  geom_point( ) + geom_line( )
```



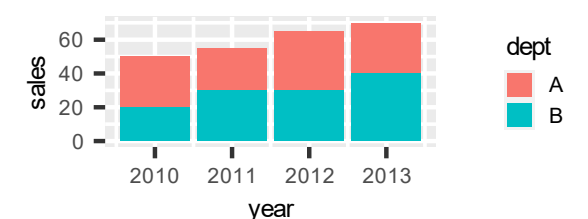
```
ggplot( my_data , aes( year , sales ) ) +
  geom_point( ) + geom_line( ) + ylim(0, 40) +
  labs(x = "" , y = "Sales per year")
```



```
ggplot(my_data, aes( year, sales, colour = dept )) +
  geom_point( ) + geom_line( )
```

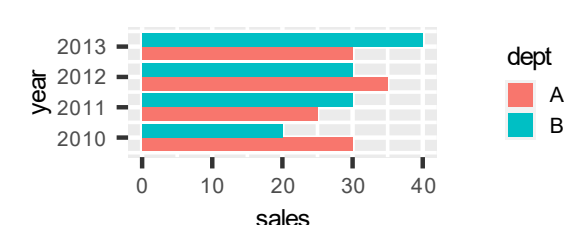


```
ggplot( my_data , aes( year, sales, fill = dept ) ) +
  geom_col( )
```



Note 'colour' for lines & points, 'fill' for shapes

```
ggplot( my_data , aes( year, sales, fill = dept ) ) +
  geom_col( position = "dodge" ) + coord_flip( )
```



C.f. position = "fill" for 100% stacked bars/cols

# Sorting and Row-Wise Operations

|   |  |
|---|--|
| <pre>proc sort data=old_data out=new_data;   by id descending income ; run;</pre>                       | <pre>new_data &lt;- old_data %&gt;%   arrange( id , desc( income ) )</pre>   |
| <pre>proc sort data=old_data nodup;   by id job_type; run;</pre>  | <pre>old_data &lt;- old_data %&gt;%   arrange( id , job_type)) %&gt;%   distinct( )</pre> <p><i>Note nodup relies on adjacency of duplicate rows, distinct( ) does not</i></p>   |
| <pre>proc sort data=old_data nodupkey;   by id ; run;</pre>   | <pre>old_data &lt;- old_data %&gt;%   arrange( id ) %&gt;%   group_by( id ) %&gt;%   slice( 1 )</pre>  |
| <pre>data new_data;   set old_data;   by id descending income ;   if first.id ; run;</pre>              | <pre>new_data &lt;- old_data %&gt;%   group_by( id ) %&gt;%   slice(which.max( income ))</pre> <p><i>C.f. which.min( )</i><br/><i>Swap to preserve duplicate maxima: ... slice_max( income )</i><br/><i>Alternatively: ... filter(income==max(income))</i></p> |
| <pre>data new_data;   set old_data;   prev_id= lag( id ); run;</pre>                                    | <pre>new_data &lt;- old_data %&gt;%   mutate( prev_id = lag( id , 1 ))</pre> <p><i>C.f. lead( ) for subsequent rows</i></p>  |
| <pre>data new_data;   set old_data;   by id;   counter + 1 ;   if first.id then counter = 1; run;</pre> | <pre>new_data &lt;- old_data %&gt;%   group_by( id ) %&gt;%   mutate( counter = row_number( ) )</pre>  |

# Converting and Rounding

|  |  |
|--|--|
| <pre>data new_data;   set old_data ;   num_var = input("5" , 8. );   text_var = put( 5 , 8. ); run;</pre>      | <pre>new_data &lt;- old_data %&gt;%   mutate(num_var = as.numeric("5" )) %&gt;%   mutate(text_var = as.character( 5 ))</pre>       |
| <pre>data new_data ;   set old_data;   nearest_5 = round( x , 5 )   two_decimals = round( x , 0.01) run;</pre> | <pre>new_data &lt;- old_data %&gt;%   mutate(nearest_5 = round(x/5)*5) %&gt;%   mutate(two_decimals = round( x , digits = 2)</pre> |

# Creating functions to modify datasets

|  |   |
|--|---|
| <pre>%macro add_variable(dataset_name); data &amp;dataset_name;   set &amp;dataset_name;   new_variable = 1; run; %mend; %add_variable( my_data );</pre> | <pre>add_variable &lt;- function( dataset_name ){   dataset_name &lt;- dataset_name %&gt;%   mutate(new_variable = 1)   return( dataset_name ) } my_data &lt;- add_variable( my_data )</pre> <p><i>Note SAS can modify within the macro, whereas R creates a copy within the function</i></p> |
|--|---|

# Dealing with strings

|  |  |
|--|--|
| <pre>data new_data;   set old_data;   if find( job_title , "Health" ); run;</pre>                | <pre>new_data &lt;- old_data %&gt;%   filter( str_detect( job_title , "Health" ))</pre>  |
| <pre>data new_data;   set old_data;   if job_title =: "Health" ; run;</pre>                      | <pre>new_data &lt;- old_data %&gt;%   filter( str_detect( job_title , "^Health" ))</pre> <p><i>Use ^ for start of string, \$ for end of string, e.g. "Health\$"</i></p>  |
| <pre>data new_data;   set old_data;   substring = substr( big_string , 3 , 4 ); run;</pre>       | <pre>new_data &lt;- old_data %&gt;%   mutate( substring = str_sub( big_string , 3 , 6 ))</pre> <p><i>Returns characters 3 to 6. Note SAS uses &lt;start&gt;, &lt;length&gt;, R uses &lt;start&gt;, &lt;end&gt;</i></p> |
| <pre>data new_data;   set old_data;   address = tranwrd( address , "Street" , "St" ); run;</pre> | <pre>new_data &lt;- old_data %&gt;%   mutate( address = str_replace_all( address , "Street" , "St" ))</pre> <p><i>C.f. str_replace( ) for first instance of pattern only</i></p>                                       |
| <pre>data new_data;   set old_data;   full_name = catx(" " , first_name , surname ); run;</pre>  | <pre>new_data &lt;- old_data %&gt;%   mutate( full_name = str_c( first_name , surname , sep = " " ))</pre> <p><i>Drop sep = " " for equivalent to cats( ) in SAS</i></p>   |
| <pre>data new_data;   set old_data;   first_word = scan( sentence , 1 ); run;</pre>              | <pre>new_data &lt;- old_data %&gt;%   mutate( first_word = word( sentence , 1 ))</pre> <p><i>R example preserves punctuation at the end of words, SAS doesn't</i></p>  |
| <pre>data new_data;   set old_data;   house_number = compress( address , , "dk" ); run;</pre>    | <pre>new_data &lt;- old_data %&gt;%   mutate( house_number = str_extract( address , "\\d+" ))</pre> <p><i>Wide range of regexps in both languages, this example extracts digits only</i></p>                           |

# File operations

|  |   |
|--|---|
| <p>Operate in 'Work' library.<br/>Use <b>libname</b> to define file locations</p>                        | <p>Operate in a particular 'working directory' (identify using <b>getwd( )</b> )<br/>Move to other locations using <b>setwd( )</b></p>          |
| <pre>libname library_name "file_location"; data library_name.saved_data;   set data_in_use; run;</pre>   | <pre>saveRDS(data_in_use , file="file_location/saved_data.rds") or setwd("file_location") saveRDS( data_in_use , file = "saved_data.rds")</pre> |
| <pre>libname library_name "file_location"; data data_in_use ;   set library_name.saved_data ; run;</pre> | <pre>data_in_use &lt;- readRDS("file_location/saved_data.rds" ) or setwd("file_location") data_in_use &lt;- readRDS("saved_data.rds")</pre>     |
| <pre>proc export data = my_data   outfile = "my_file.csv" dbms = csv replace; run;</pre>                 | <pre>write_csv(my_data , "my_file.csv")</pre>   |
| <pre>proc import datafile = "my_file.csv"   out = my_data dbms = csv; run;</pre>                         | <pre>my_data &lt;- read_csv("my_file.csv")</pre> <p><i>Both examples assume column headers in csv file</i></p>                                  |