

# 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

<pre>data new_data;   set old_data; run;</pre>	<pre>new_data &lt;- old_data</pre>
<pre>data new_data (keep=id);   set old_data (drop=job_title); run;</pre>	<pre>new_data &lt;- old_data %&gt;%   select(-job_title) %&gt;%   select(id)</pre>
<pre>data new_data (drop= temp: );   set old_data; run;</pre>	<pre>new_data &lt;- old_data %&gt;%   select( -starts_with("temp") )</pre> <p><i>C.f. contains( ), ends_with( )</i></p>
<pre>data new_data;   set old_data;   rename old_name = new_name; run;</pre>	<pre>new_data &lt;- old_data %&gt;%   rename(new_name = old_name)</pre> <p><i>Note order differs</i></p>

## Conditional filtering

<pre>data new_data;   set old_data;   if Sex = "M"; run;</pre>	<pre>new_data &lt;- old_data %&gt;%   filter(Sex == "M")</pre>
<pre>data new_data;   set old_data;   if year in (2010,2011,2012); run;</pre>	<pre>new_data &lt;- old_data %&gt;%   filter(year %in%   c(2010,2011,2012))</pre>
<pre>data new_data;   set old_data;   by id;   if first.id; run;</pre>	<pre>new_data &lt;- old_data %&gt;%   group_by( id ) %&gt;%   slice(1)</pre> <p><i>Could use slice(n()) for last</i></p>
<pre>data new_data;   set old_data;   if dob &gt; "25APR1990"d; run;</pre>	<pre>new_data &lt;- old_data %&gt;%   filter(dob &gt; as.Date("1990-04-25"))</pre>

## New variables, conditional editing

<pre>data new_data;   set old_data;   total_income = wages + benefits; run;</pre>	<pre>new_data &lt;- old_data %&gt;%   mutate(total_income = wages + benefits)</pre>
<pre>data new_data;   set old_data;   if hours &gt; 30 then full_time = "Y";   else full_time = "N"; run;</pre>	<pre>new_data &lt;- old_data %&gt;%   mutate(full_time = if_else(hours &gt; 30, "Y", "N"))</pre>
<pre>data new_data;   set old_data;   if temp &gt; 20 then weather = "Warm";   else if temp &gt; 10 then weather = "Mild";   else weather = "Cold"; run;</pre>	<pre>new_data &lt;- old_data %&gt;%   mutate(weather = case_when(     temp &gt; 20 ~ "Warm",     temp &gt; 10 ~ "Mild",     TRUE ~ "Cold" ))</pre>

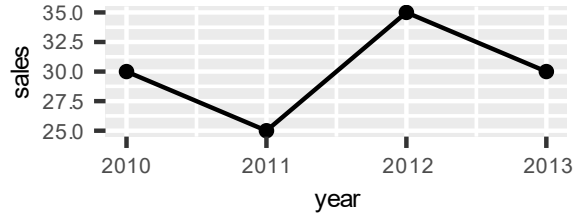
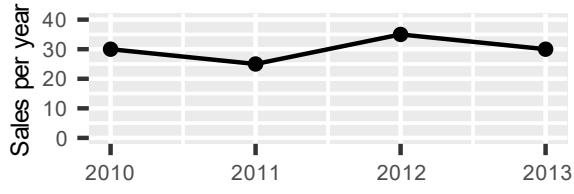
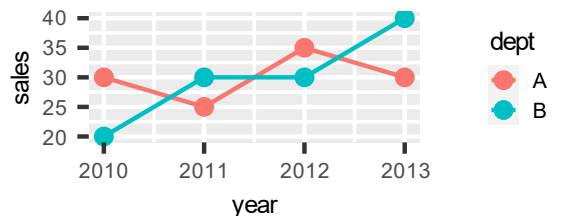
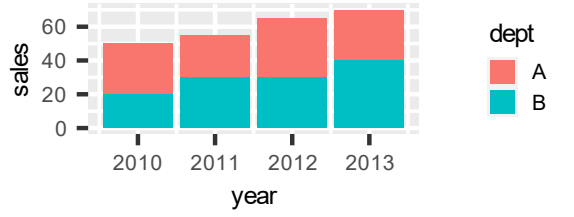
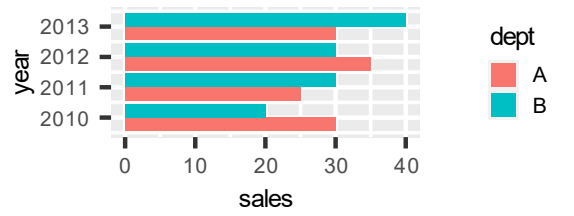
## Counting and Summarising

<pre>proc freq data = old_data ;   table job_type ; run;</pre>	<pre>old_data %&gt;%   count( job_type )</pre> <p><i>For percent, add:</i> <code>%&gt;% mutate(percent = n*100/sum(n))</code></p>
<pre>proc freq data = old_data ;   table job_type*region ; run;</pre>	<pre>old_data %&gt;%   count( job_type , region )</pre>
<pre>proc summary data = old_data nway ;   class job_type region ;   output out = new_data ; run;</pre>	<pre>new_data &lt;- old_data %&gt;%   group_by( job_type , region ) %&gt;%   summarise( Count = n( ) )</pre> <p><i>Equivalent without nway not trivially produced</i></p>
<pre>proc summary data = old_data nway ;   class job_type region ;   var salary ;   output out = new_data   sum( salary ) = total_salaries ; run;</pre>	<pre>new_data &lt;- old_data %&gt;%   group_by( job_type , region ) %&gt;%   summarise( total_salaries = sum( salary ) ,     Count = n( ) )</pre> <p><i>Lots of summary functions in both languages</i> <i>Swap summarise( ) for mutate( ) to add summary data to original data</i></p>

## Combining datasets

<pre>data new_data ;   set data_1 data_2 ; run;</pre>	<pre>new_data &lt;- bind_rows( data_1 , data_2 )</pre> <p><i>C.f. rbind( ) which produces error if columns are not identical</i></p>
<pre>data new_data ;   merge data_1 (in= in_1) data_2 ;   by id ;   if in_1 ; run;</pre>	<pre>new_data &lt;- left_join( data_1 , data_2 , by = "id" )</pre> <p><i>C.f. full_join( ) , right_join( ) , inner_join( )</i></p>

## Some plotting in R

<pre>ggplot( my_data , aes( year , sales ) ) +   geom_point( ) + geom_line( )</pre>	
<pre>ggplot( my_data , aes( year , sales ) ) +   geom_point( ) + geom_line( ) + ylim(0, 40) +   labs(x = "" , y = "Sales per year")</pre>	
<pre>ggplot(my_data, aes( year, sales, colour = dept )) +   geom_point( ) + geom_line( )</pre>	
<pre>ggplot( my_data , aes( year, sales, fill = dept ) ) +   geom_col( )</pre>	
<pre>ggplot( my_data , aes( year, sales, fill = dept ) ) +   geom_col( position = "dodge" ) + coord_flip( )</pre>	

*Note 'colour' for lines & points, 'fill' for shapes*  
*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> <i>Swap to preserve duplicate maxima: ... slice.max( income )</i> <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. Use <b>libname</b> to define file locations</p>	<p>Operate in a particular 'working directory' (identify using <b>getwd( )</b> ) 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>save(data_in_use , file="file_location/saved_data.rda") or setwd("file_location") save( data_in_use , file = "saved_data.rda")</pre>
<pre>libname library_name "file_location"; data data_in_use ;   set library_name.saved_data ; run;</pre>	<pre>load("file_location/saved_data.rda" ) or setwd("file_location") load("saved_data.rda")</pre> <p><i>save( ) can store multiple data frames in a single .rda file, load( ) will restore all of these</i></p>
<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>