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;                    new_data <- old_data
  set old_data;
run;
```

```
data new_data (keep=id);          new_data <- old_data %>%
  set old_data (drop=job_title) ;   select(-job_title) %>%
run;                                select(id)
```

```
data new_data (drop= temp: );     new_data <- old_data %>%
  set old_data;                     select( -starts_with("temp")
run;
```
C.f. contains( ) , ends_with( )

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

Conditional filtering

```
data new_data;                    new_data <- old_data %>%
  set old_data;                     filter(Sex == "M")
  if Sex = "M";
run;
```

```
data new_data;                    new_data <- old_data %>%
  set old_data;                     filter(year %in% c(2010,2011,2012))
  if year in (2010,2011,2012);
run;
```

```
data new_data;                    new_data <- old_data %>%
  set old_data;                     group_by( id ) %>%
  by id ;                           slice(1)
  if first.id ;
run;
```
Could use slice(n( )) for last

```
data new_data;                    new_data <- old_data %>%
  set old_data;                     filter(dob > as.Date("1990-04-25"))
  if dob > "25APR1990"d;
run;
```

New variables, conditional editing

```
data new_data;                    new_data <- old_data %>%
  set old_data;                     mutate(total_income = wages + benefits)
  total_income = wages + benefits ;
run;
```

```
data new_data;                    new_data <- old_data %>%
  set old_data;                     mutate(full_time = if_else(hours > 30 , "Y" , "N"))
  if hours > 30 then full_time = "Y";
  else full_time = "N";
run;
```

```
data new_data;                    new_data <- old_data %>%
  set old_data;                     mutate(weather = case_when(
  if temp > 20 then weather = "Warm";   temp > 20 ~ "Warm",
  else if temp > 10 then weather = "Mild";  temp > 10 ~"Mild",
  else weather = "Cold";                TRUE ~ "Cold" ) )
run;
```

Counting and Summarising

```
proc freq data = old_data ;       old_data %>%
  table job_type ;                  count( job_type )
run;
```
For percent, add:
*%>% mutate(percent = n*100/sum(n))*

```
proc freq data = old_data ;       old_data %>%
  table job_type*region ;           count( job_type , region )
run;
```

```
proc summary data = old_data nway ;   new_data <- old_data %>%
  class job_type region ;              group_by( job_type , region ) %>%
  output out = new_data ;              summarise( Count = n( ) )
run;
```
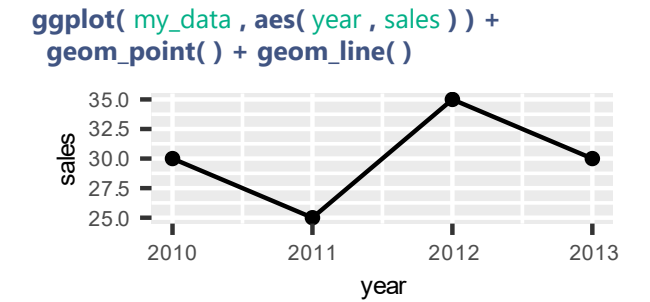*Equivalent without nway not trivially produced*

```
proc summary data = old_data nway ;   new_data <- old_data %>%
  class job_type region ;              group_by( job_type , region ) %>%
  var salary ;                         summarise( total_salaries = sum( salary ) ,
  output out = new_data                  Count = n( ) )
    sum( salary ) = total_salaries ;
run;
```
Lots of summary functions in both languages
Swap summarise( ) for mutate( ) to add summary data to original data

Combining datasets

```
data new_data ;                   new_data <- bind_rows( data_1 , data_2 )
  set data_1 data_2 ;
run;
```
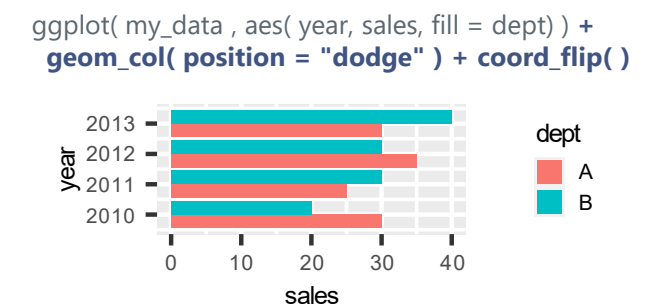*C.f. rbind( ) which produces error if columns are not identical*

```
data new_data ;                   new_data <- left_join( data_1 , data_2 , by = "id")
  merge data_1 (in= in_1) data_2 ;
  by id ;
  if in_1 ;
run;
```
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( )
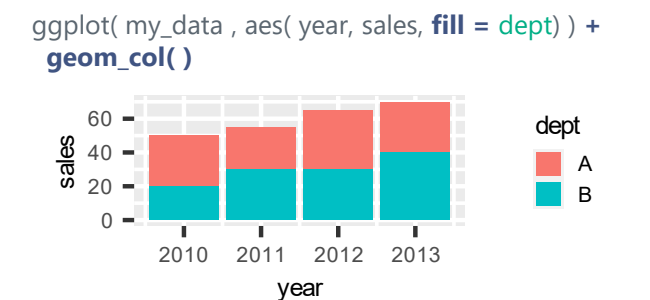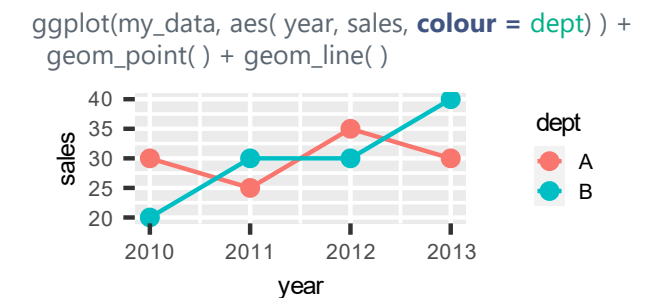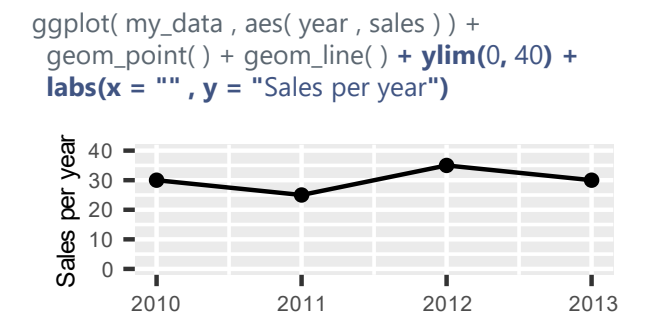
Note 'colour' for lines & points, 'fill' for shapes

```
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

```
proc sort data=old_data out=new_data;          new_data <- old_data %>%
  by id descending income ;                         arrange( id , desc( income ) )
run;
```

```
proc sort data=old_data nodup;                 old_data <- old_data %>%
  by id job_type;                                  arrange( id , job_type)) %>%
run;                                             distinct( )
```
Note nodup relies on adjacency of duplicate rows, distinct( ) does not

---

```
proc sort data=old_data nodupkey;             old_data <- old_data %>%
  by id ;                                         arrange( id ) %>%
run;                                            group_by( id ) %>%
                                                slice( 1 )
```

```
data new_data;                                 new_data <- old_data %>%
  set old_data;                                   group_by( id ) %>%
  by id descending income ;                       slice(which.max( income ))
  if first.id ;                                                      C.f.which.min( )
run;
```
Swap to preserve duplicate maxima: … slice_max(income)
Alternatively: … filter(income==max(income))

```
data new_data;                                 new_data <- old_data %>%
  set old_data;                                   mutate( prev_id = lag( id , 1 ))
  prev_id= lag( id );
run;
```
C.f. lead( ) for subsequent rows

```
data new_data;                                 new_data <- old_data %>%
  set old_data;                                   group_by( id ) %>%
  by id;                                          mutate( counter = row_number( ) )
  counter +1 ;
  if first.id then counter = 1;
run;
```

Converting and Rounding

```
data new_data;                                 new_data <- old_data %>%
  set old_data ;                                  mutate(num_var = as.numeric("5" )) %>%
  num_var = input("5" , 8. );                     mutate(text_var = as.character( 5 ))
  text_var = put( 5 , 8. );
run;
```

```
data new_data  ;                               new_data <- old_data %>%
  set old_data;                                   mutate(nearest_5 = round(x/5)*5) %>%
  nearest_5 = round( x , 5 )                       mutate(two_decimals = round( x , digits = 2)
  two_decimals = round( x , 0.01)
run;
```

Creating functions to modify datasets

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

String Manipulation

```
data new_data;                                 new_data <- old_data %>%
  set old_data;                                   filter( str_detect( job_title , "Health" ))
  if find( job_title , "Health" );
run;
```

```
data new_data;                                 new_data <- old_data %>%
  set old_data;                                   mutate( substring = str_sub( big_string , 3 , 6 ))
  substring = substr( big_string , 3 , 4 );
run;
```
Returns characters 3 to 6. Note SAS uses <start>, <length>, R uses <start>, <end>

```
data new_data;                                 new_data <- old_data %>%
  set old_data;                                   mutate( address = str_replace_all( address , "Street" , "St" ))
  address = tranwrd( address , "Street" , "St" );
run;
```
C.f. str_replace( ) for first instance of pattern only

```
data new_data;                                 new_data <- old_data %>%
  set old_data;                                   mutate( house_number = str_extract( address , "\\d+" ))
  house_number = compress( address , , "dk" );
run;
```
Wide range of regexps in both languages, this example extracts digits only

Transpose/Pivot

```
proc transpose data=long_data out=wide_data;   wide_data <- long_data %>%
  by student ;                                     pivot_wider(names_from = subject , values_from = grade)
  id subject ;
  var grade ;
run;
```
Add NOTSORTED if long_data is not sorted by student

```
proc transpose data=wide_data                  long_data <- wide_data %>%
  out=long_data(rename=(col1=grade)) name=subject;  pivot_longer(c(English, Irish, Maths) ,
  by student ;                                      names_to = "subject", values_to = "grade")
  var English Irish Maths;
run;
```

File operations

```
Operate in 'Work' library.                     Operate in a particular 'working directory' (identify using getwd( ) )
Use libname to define file locations           Move to other locations using setwd( )
```

```
libname library_name "file_location";          saveRDS(data_in_use , file="file_location/saved_data.rds")
data library_name.saved_data;                  or
  set data_in_use;                             setwd("file_location")
run;                                           saveRDS( data_in_use , file = "saved_data.rds")
```

```
libname library_name "file_location";          data_in_use <- readRDS("file_location/saved_data.rds" )
data data_in_use ;                             or
  set library_name.saved_data ;               setwd("file_location")
run;                                           data_in_use <- readRDS("saved_data.rds")
```

```
proc export data = my_data                     write_csv(my_data , "my_file.csv")
  outfile = "my_file.csv" dbms = csv replace;
run;
```

```
proc import datafile = "my_file.csv"           my_data <- read_csv("my_file.csv")
  out = my_data dbms = csv;
run;
```
Both examples assume column headers in csv file