The **dplyr** package Cheat Sheet

Intro

Dplyr is a powerful and fast R programming package used to manipulate datasets. It makes working with data easier by constraining the options and providing simple functions to use.

Selecting columns & filtering rows

select() - Subset a dataframe by its columns.

filter() - Extract rows from a dataframe

mutate() - Creates new columns based on existing ones

transmute() - Creates new columns ignoring the old ones

summarise() - Collapses given existing values to a single-row summary.

Pipe operator (%>%)

The pipe operator **%>%** is used to take the output of one function and use it in the next. This new syntax is useful because it's easier to read, write and tells the story of your analysis.

```
# Instead of this
mtcars_gears <- group_by(mtcars, gear)
# You can use pipe %>% and do
mtcars %>% group_by(gear)
```

Arrange and Editing Data

arrange() - Sorts/reorders the rows in the data frame by the value of given columns

relocate() - Rearranging the position of a set of columns.

desc() - Orders a column in descending order, useful with arrange()

rename() - Changes the name of individual columns, order and attributes are preserved.

Slicing the Data

slice() - Makes it possible to select rows based off their location

slice_head() - Without specifying this function extracts the first row

slice_tail() - Filters out the last row in the data frame

slice_sample() - Randomly selects rows

slice_min() - Selects the rows based on the lowest values of the variable

slice_max() - Selects the certain row in which the values are the highest

Detailed Summary of the Data

summarize_all() - Requires an argument that affects every variable and summarizes all columns

summarize_if() - Requires two arguments and returns a boolean value for conditional mapping

dplyr::first() - first value **dplyr:: last()** - last value

dplyr::nth() - nth value **dplyr::n()** - no. values
dplyr::n_distinct() - no. unique values

IQR() - Interquartile range
min() - min value
median() - median
sd() - standard deviation

max() - max value
mean() - mean
var() - variance

RStudio®

Not Applicable Values

Below are examples of functions that are used in order to replace or manipulate values that are NA (Not Available). This is helpful as it allows data to be stored in the position of where results would have been.

na_if() - Converts the value to NA

is.na() - Checks to find NA values and returns true or false for each value in a data set.

replace_na() - Used to replace NAs with specific values

drop_na() - Drops any rows that contain NA's

na.omit() - Deletes the NA values in your dataset

Advantages of dplyr

RPostgreSQL was much slower than our current package dplyr as it has the most important parts of its code written in Rcpp. This is a package that integrates R with C++ to accelerate computations.

Grouping data

group_by() - Splits the data into groups for further manipulation

ungroup() - Gives ungrouped data

Inside summarise() or mutate() these can be used to return info on current group:

cur_data() - Not including grouping variables, gives current data

cur_data_all() - Gives all current data

cur_group() - Returns group keys (help you to identify a row and the relationship between tables). Each grouping variable gets a tibble w/ one row and column

cur_group_id() - the current group gets a unique numeric identifier

cur_group_rows() - Row indices for the current group are returned

(for use in across() only)
cur_column() - The name of the current column is returned

Group-Data and the following functions return data on all groups

group_data() - Returns grouping structure
group_keys() - Returns grouping data only
group_rows() - Returns just location

group_indices() - Returns integer vector = length of data

group_vars()- Returns the names of grouping variables

groups() - Returns symbols corresponding to the names

group_size() - Size of each group is given

n_groups - The total number of groups is returned

Other Package integration

To use Tidyr(used for making code neater) with dplyr you need to take advantage of some of the reshaping functions.
Pivot_wider()/_longer() - These can be used in order to create new columns allowing for the structure of the table to be manipulated from tidyr to dplyr and vice versa

Ggplot can also be used to visualise the data used in dplyr(with ggplot2 having a similar function to the pipe which can be engineered over easier than dplyr)

Joins: (continued)

by = c specifies columns to be joined at e.g. right_join(x, y, b = "A")

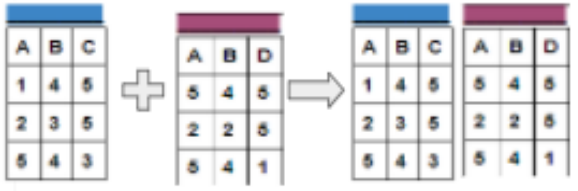
semi_join(x, y, by = "A",...) - x which match y

anti_join(x,y, by ="A",...) - x w/ no match w/ y

Joining Tables

bind_cols() - Tables are pasted next to each other

bind_cols(...) - tables given side by side as one table



bind_rows() - Gives tables joined underneath each other.

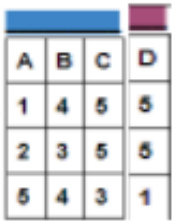
bind_rows(..., id = "A") - tables joined underneath as one table. id = allows you to add column names originally used to a column bind_rows

intersect(x, y,...) - Rows in both x and y

set_diff(x, y,...) - Rows only in x

union(x, y...) - Rows in x or y

left_join(x, y, by = "A",...) - Join at y to x



right_join(x, y, by = "A",...) - Join at x to y

inner_join(x, y, by = "A",...) - Only join rows which match

full_join(x, y, by ="A",...) - Join all data

suffix= adding a number to columns with the same name to differentiate them
e.g. right_join(x, y, b = "A", suffix = "1", "2")