Study Guide: Data Manipulation with R

Afshine Amidi and Shervine Amidi

August 21, 2020

Main concepts

❒ **File management** – The table below summarizes the useful commands to make sure the working directory is correctly set:

Category	Action	Command
Paths	Change directory to another path	`setwd(path)`
	Get current working directory	`getwd()`
	Join paths	`file.path(path_1, ..., path_n)`
Files	List files and folders in a given directory	`list.files(path, include.dirs = TRUE)`
	Check if path is a file / folder	`file_test('-f', path)`
		`file_test('-d', path)`
	Read / write csv file	`read.csv(path_to_csv_file)`
		`write.csv(df, path_to_csv_file)`

❒ **Chaining** – The symbol `%>%`, also called "pipe", enables to have chained operations and provides better legibility. Here are its different interpretations:

- `f(arg_1, arg_2, ..., arg_n)` is equivalent to `arg_1 %>% f(arg_2, arg_3, ..., arg_n)`, and also to:
  - `arg_1 %>% f(., arg_2, ..., arg_n)`
  - `arg_2 %>% f(arg_1, ., arg_3, ..., arg_n)`
  - `arg_n %>% f(arg_1, ..., arg_n-1, .)`
- A common use of pipe is when a dataframe `df` gets first modified by `some_operation_1`, then `some_operation_2`, until `some_operation_n` in a sequential way. It is done as follows:

R
```
# df gets some_operation_1, then some_operation_2, ...,
# then some_operation_n
df %>%
  some_operation_1 %>%
  some_operation_2 %>%
        ...        %>%
  some_operation_n
```

❒ **Exploring the data** – The table below summarizes the main functions used to get a complete overview of the data:

Category	Action	Command
Look at data	Select columns of interest	`df %>% select(col_list)`
	Remove unwanted columns	`df %>% select(-col_list)`
	Look at n first rows / last rows	`df %>% head(n)` / `df %>% tail(n)`
	Summary statistics of columns	`df %>% summary()`
Data types	Data types of columns	`df %>% str()`
	Number of rows / columns	`df %>% NROW()` / `df %>% NCOL()`

❒ **Data types** – The table below sums up the main data types that can be contained in columns:

Data type	Description	Example
`character`	String-related data	`'teddy bear'`
`factor`	String-related data that can be put in bucket, or ordered	`'high'`
`numeric`	Numerical data	`24.0`
`int`	Numeric data that are integer	`24`
`Date`	Dates	`'2020-01-01'`
`POSIXct`	Timestamps	`'2020-01-01 00:01:00'`

Data preprocessing

❒ **Filtering** – We can filter rows according to some conditions as follows:

R
```
df %>%
  filter(some_col some_operation some_value_or_list_or_col)
```

where `some_operation` is one of the following:

Category	Operation	Command
Basic	Equality / non-equality	`==` / `!=`
	Inequalities	`<`, `<=`, `>=`, `>`
	And / or	`&` / `|`
Advanced	Check for missing value	`is.na()`
	Belonging	`%in% (val_1, ..., val_n)`
	Pattern matching	`%like% 'val'`

Remark: we can filter columns with the `select_if` *command.*

❒ **Changing columns** – The table below summarizes the main column operations:

Action	Command
Add new columns on top of old ones	`df %>% mutate(new_col = operation(other_cols))`
Add new columns and discard old ones	`df %>% transmute(new_col = operation(other_cols))`
Modify several columns in-place	`df %>% mutate_at(vars, funs)`
Modify all columns in-place	`df %>% mutate_all(funs)`
Modify columns fitting a specific condition	`df %>% mutate_if(condition, funs)`
Unite columns	`df %>% unite(new_merged_col, old_cols_list)`
Separate columns	`df %>% separate(col_to_separate, new_cols_list)`

❒ **Conditional column** – A column can take different values with respect to a particular set of conditions with the `case_when()` command as follows:

R
```
case_when(condition_1 ~ value_1,  # If condition_1 then value_1
          condition_2 ~ value_2,  # If condition_2 then value_2
                ...
          TRUE ~ value_n)         # Otherwise, value_n
```

Remark: the `ifelse(condition_if_true, value_true, value_other)` *can be used and is easier to manipulate if there is only one condition.*

❒ **Mathematical operations** – The table below sums up the main mathematical operations that can be performed on columns:

Operation	Command
$\sqrt{x}$	`sqrt(x)`
$\lfloor x \rfloor$	`floor(x)`
$\lceil x \rceil$	`ceiling(x)`

❒ **Datetime conversion** – Fields containing datetime values can be stored in two different `POSIXt` data types:

Action	Command
Converts to datetime with seconds since origin	`as.POSIXct(col, format)`
Converts to datetime with attributes (e.g. time zone)	`as.POSIXlt(col, format)`

where `format` is a string describing the structure of the field and using the commands summarized in the table below:

Category	Command	Description	Example
Year	`'%Y'` / `'%y'`	With / without century	`2020` / `20`
Month	`'%B'` / `'%b'` / `'%m'`	Full / abbreviated / numerical	`August` / `Aug` / `8`
Weekday	`'%A'` / `'%a'`	Full / abbreviated	`Sunday` / `Sun`
	`'%u'` / `'%w'`	Number (1-7) / Number (0-6)	`7` / `0`
Day	`'%d'` / `'%j'`	Of the month / of the year	`09` / `222`
Time	`'%H'` / `'%M'`	Hour / minute	`09` / `40`
Timezone	`'%Z'` / `'%z'`	String / Number of hours from UTC	`EST` / `-0400`

Remark: data frames only accept datetime in `POSIXct` *format.*

❒ **Date properties** – In order to extract a date-related property from a datetime object, the following command is used:

R
```
format(datetime_object, format)
```

where `format` follows the same convention as in the table above.

Data frame transformation

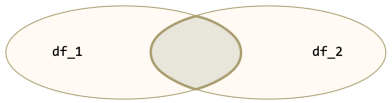
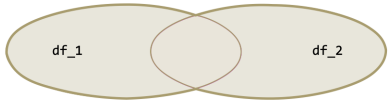
❒ **Merging data frames** – We can merge two data frames by a given field as follows:

R
```
merge(df_1, df_2, join_field, join_type)
```

where `join_field` indicates fields where the join needs to happen:

Case	Fields are equal	Different field names
Command	`by = 'field'`	`by.x = 'field_1', by.y = 'field_2'`

and where `join_type` indicates the join type, and is one of the following:

Join type	Option	Illustration
Inner join	default	
Left join	`all.x = TRUE`	
Right join	`all.y = TRUE`	
Full join	`all = TRUE`	

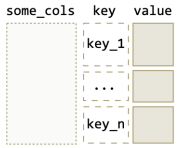
Remark: if the `by` *parameter is not specified, the merge will be a cross join.*

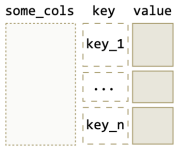
❒ **Concatenation** – The table below summarizes the different ways data frames can be concatenated:

Type	Command	Illustration
Rows	`rbind(df_1, ..., df_n)`	
Columns	`cbind(df_1, ..., df_n)`	

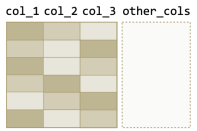
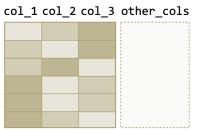
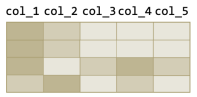
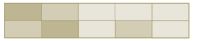
❒ **Common transformations** – The common data frame transformations are summarized in the table below:

Type	Command	Illustration	
		Before	After

Long to wide	`spread(`
	`  df, key = 'key',`
	`  value = 'value'`
	`)`
			

Wide to long	`gather(`
	`  df, key = 'key'`
	`  value = 'value',`
	`  c(key_1, ..., key_n)`
	`)`
			

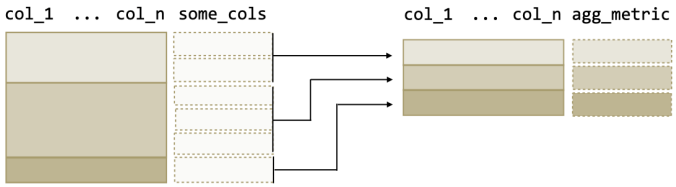
❒ **Row operations** – The following actions are used to make operations on rows of the data frame:

Action	Command	Illustration	
		Before	After
Sort with respect to columns	`df %>%` `  arrange(col_1, ..., col_n)`		
Dropping duplicates	`df %>% unique()`		
Drop rows with at least a null value	`df %>% na.omit()`		

Remark: by default, the `arrange` *command sorts in ascending order. If we want to sort it in descending order, the* `-` *command needs to be used before a column.*

Aggregations

❒ **Grouping data** – Aggregate metrics are computed across groups as follows:



The R command is as follows:

R

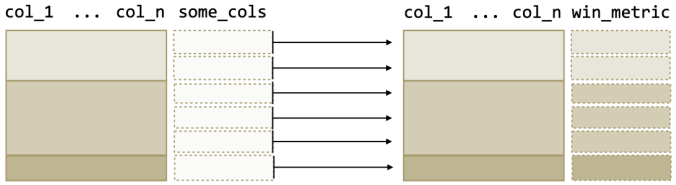
```
df %>%                                                  # Ungrouped data frame
  group_by(col_1, ..., col_n) %>%                       # Group by some columns
  summarize(agg_metric = some_aggregation(some_cols))   # Aggregation step
```

❒ **Aggregate functions** – The table below summarizes the main aggregate functions that can be used in an aggregation query:

Category	Action	Command
Properties	Count of observations	`n()`
Values	Sum of values of observations	`sum()`
	Max / min of values of observations	`max()` / `min()`
	Mean / median of values of observations	`mean()` / `median()`
	Standard deviation / variance across observations	`sd()` / `var()`

Window functions

❒ **Definition** – A window function computes a metric over groups and has the following structure:



The R command is as follows:

R

```
df %>%                                          # Ungrouped data frame
  group_by(col_1, ..., col_n) %>%               # Group by some columns
  mutate(win_metric = window_function(col))     # Window function
```

Remark: applying a window function will not change the initial number of rows of the data frame.

❒ **Row numbering** – The table below summarizes the main commands that rank each row across specified groups, ordered by a specific field:

Join type	Command	Example
`row_number(x)`	Ties are given different ranks	1, 2, 3, 4
`rank(x)`	Ties are given same rank and skip numbers	1, 2.5, 2.5, 4
`dense_rank(x)`	Ties are given same rank and do not skip numbers	1, 2, 2, 3

❒ **Values** – The following window functions allow to keep track of specific types of values with respect to the group:

Command	Description
`first(x)`	Takes the first value of the column
`last(x)`	Takes the last value of the column
`lag(x, n)`	Takes the n^{th} previous value of the column
`lead(x, n)`	Takes the n^{th} following value of the column
`nth(x, n)`	Takes the n^{th} value of the column