R basics

Table of contents

1 Setup

1.1 data.frame

None needed: this is part of Base R.

1.2 tibble

Either `library(tibble)` or (preferably) `library(tidyverse)`. The latter also brings in important libraries such as `dplyr`.

Documentation is fairly extensive, in the Tidyverse style:

- A website.
- A function reference.
- A chapter in R for Data Science.

1.3 data.table

1.4 Polars

1.5 DuckDB

1.6 Arrow

2 Creating

2.1 data.frame

```
# create the column vectors
languages <- c("Fortran", "R", "Python", "Julia")
created <- c(1957, 1993, 1991, 2012)
has.syllabus <- c(FALSE, TRUE, TRUE, TRUE)

# join columns to create the dataframe
df <- data.frame(languages, created, has.syllabus)
df
#>   languages created has.syllabus
#> 1   Fortran    1957        FALSE
#> 2         R    1993         TRUE
```

```
#> 3    Python    1991         TRUE
#> 4     Julia    2012         TRUE

# look at the structure
str(df)
#> 'data.frame':    4 obs. of  3 variables:
 #> $ languages   : chr  "Fortran" "R" "Python" "Julia"
 #> $ created     : num  1957 1993 1991 2012
 #> $ has.syllabus: logi  FALSE TRUE TRUE TRUE
```

2.2 tibble

```
# column vectors are same as for data.frame
library(tibble)
tbl <- tibble(languages, created, has.syllabus)
tbl
  # A tibble: 4 × 3
#>   languages created has.syllabus
#>   <chr>       <dbl> <lgl>
#> 1 Fortran      1957 FALSE
#> 2 R            1993 TRUE
#> 3 Python       1991 TRUE
#> 4 Julia        2012 TRUE

str(tbl)
#> tibble [4 × 3] (S3: tbl_df/tbl/data.frame)
#>  $ languages   : chr [1:4] "Fortran" "R" "Python" "Julia"
#>  $ created     : num [1:4] 1957 1993 1991 2012
#>  $ has.syllabus: logi [1:4] FALSE TRUE TRUE TRUE
```

2.3 data.table

2.4 Polars

2.5 DuckDB

2.6 Arrow

3 Importing data

3.1 data.frame

3.2 tibble

3.3 data.table

3.4 Polars

3.5 DuckDB

3.6 Arrow

4 Subsetting

4.1 data.frame & tibble

Dataframes, including tibbles, can be treated as lists of column vectors, so list indexing recovers a specified column.

```
tbl
# A tibble: 4 × 3
#>   languages created has.syllabus
#>   <chr>       <dbl> <lgl>
#> 1 Fortran      1957 FALSE
#> 2 R            1993 TRUE
#> 3 Python       1991 TRUE
#> 4 Julia        2012 TRUE

tbl$created
#> [1] 1957 1993 1991 2012
```

A dataframe can also be indexed with matrix-style indexing.

```
tbl[c(2, 4), 1:2]
  # A tibble: 2 × 2
#>   languages created
#>   <chr>       <dbl>
#> 1 R            1993
#> 2 Julia        2012
```

Using `dplyr`, get a single column with `pull()` with the name or sequential number (negative numbers to count right-to-left).

```
tbl |> pull(created)
#> [1] 1957 1993 1991 2012
```

This is the same result as `tbl$created`, but using a pipeline-friendly function.

To get multiple columns, the appropriate function is `select()`, which is highly versatile. Get (or drop) columns based on properties of their name or type.

```
# Range with position and/or name
tbl |> select(1:created)
  # A tibble: 4 × 2
#>   languages created
#>   <chr>       <dbl>
#> 1 Fortran      1957
#> 2 R            1993
#> 3 Python       1991
#> 4 Julia        2012

# Exclude a column
tbl |> select(!created)
  # A tibble: 4 × 2
#>   languages has.syllabus
#>   <chr>     <lgl>
#> 1 Fortran   FALSE
#> 2 R         TRUE
#> 3 Python    TRUE
#> 4 Julia     TRUE

# Use type of column
tbl |> select(where(is.numeric))
  # A tibble: 4 × 1
#>   created
#>     <dbl>
#> 1    1957
#> 2    1993
#> 3    1991
#> 4    2012
```

Multiple criteria are allowed, using Boolean operators `&`, `|` and `!` (and, or not).

Column names that are *valid R identifiers* do not need quotes within a `select()`. Invalid names (e.g. those including spaces) can be enclosed in backticks, though renaming them might be better.

The `select()` function can work with a range of helper functions to pick column names: `starts_with`, `contains`, `num_range` and various others. `matches` allows full RegEx matching. See the documentation for details.

```
# limit display to top 3 rows of non-list columns
starwars |>
  select(!where(is.list)) |>
  head(3)
  # A tibble: 3 × 11
#>   name           height  mass hair_color skin_color  eye_color birth_year sex   gender
#>   <chr>           <int> <dbl> <chr>      <chr>       <chr>          <dbl> <chr> <chr>
#> 1 Luke Skywalker    172    77 blond      fair        blue            19 male  masculin
#> 2 C-3PO             167    75 NA         gold        yellow         112 none  masculin
#> 3 R2-D2              96    32 NA         white, blue red             33 none  masculin

# pick a subset of columns
starwars |>
  select(name | ends_with("color")) |>
  head(5)
  # A tibble: 5 × 4
#>   name           hair_color skin_color  eye_color
#>   <chr>          <chr>      <chr>       <chr>
#> 1 Luke Skywalker blond      fair        blue
#> 2 C-3PO          NA         gold        yellow
#> 3 R2-D2          NA         white, blue red
#> 4 Darth Vader    none       white       yellow
#> 5 Leia Organa    brown      light       brown
```

Get rows matching some criteria with `filter()`, or exclude them with `filter_out()`.

```
starwars |>
  select(name:mass) |>
  filter(between(height, 150, 165) & !is.na(mass))
  # A tibble: 4 × 3
#>   name               height  mass
#>   <chr>               <int> <dbl>
#> 1 Leia Organa           150    49
#> 2 Beru Whitesun Lars    165    75
#> 3 Nien Nunb             160    68
#> 4 Ben Quadinaros        163    65
```

Filter criteria can be arbitrarily complex, but always based on row *contents*.

If row *numbers* are known, we can use a variety of `slice()` functions.

```
starwars |>
  select(name | homeworld) |>
  slice(20:25)
  # A tibble: 6 × 2
#>   name             homeworld
#>   <chr>            <chr>
#> 1 Palpatine        Naboo
#> 2 Boba Fett        Kamino
#> 3 IG-88            NA
```

```
#> 4 Bossk            Trandosha
#> 5 Lando Calrissian Socorro
#> 6 Lobot            Bespin

# random sample of rows
starwars |>
  select(name | homeworld) |>
  slice_sample(n = 4)
  # A tibble: 4 × 2
#>   name            homeworld
#>   <chr>           <chr>
#> 1 Shaak Ti        Shili
#> 2 Luminara Unduli Mirial
#> 3 Grievous        Kalee
#> 4 Palpatine       Naboo
```

To remove duplicate rows, use `distinct()`.

4.2 data.table

4.3 Polars

4.4 DuckDB

4.5 Arrow

5 Sorting

5.1 data.frame & tibble

`arrange()` sorts rows by the values in one or more columns.

```
tbl
  # A tibble: 4 × 3
#>   languages created has.syllabus
#>   <chr>       <dbl> <lgl>
#> 1 Fortran      1957 FALSE
#> 2 R            1993 TRUE
#> 3 Python       1991 TRUE
#> 4 Julia        2012 TRUE

tbl |> arrange(languages)
  # A tibble: 4 × 3
#>   languages created has.syllabus
#>   <chr>       <dbl> <lgl>
#> 1 Fortran      1957 FALSE
#> 2 Julia        2012 TRUE
#> 3 Python       1991 TRUE
#> 4 R            1993 TRUE
```

5.2 data.table

5.3 Polars

5.4 DuckDB

5.5 Arrow

6 Modifying

6.1 data.frame & tibble

Column names can be changed with `rename(newname = oldname)`, or `rename_with()` to apply a function. A typical use would be cleaning up imported names to make them easier to work with in R, by removing whitespace and forcing a consistent format for related names.

Note the syntax within `rename()`. The *contents* of column `oldname` are *bound* to name `newname`, hence the order.

Column order can be changed with `relocate()`. Specified column(s) are moved to the left-most position(s) by default, but a `.before` or `.after` argument can be used for finer positioning.

```
sw <- starwars |> select(name:species) |> slice(1:4)
sw
  # A tibble: 4 × 11
#>   name           height  mass hair_color skin_color  eye_color birth_year sex   gender
#>   <chr>           <int> <dbl> <chr>      <chr>       <chr>          <dbl> <chr> <chr>
#> 1 Luke Skywalker    172    77 blond      fair        blue            19   male  masculin
#> 2 C-3PO             167    75 NA         gold        yellow         112   none  masculin
#> 3 R2-D2              96    32 NA         white, blue red             33   none  masculin
#> 4 Darth Vader       202   136 none       white       yellow          41.9 male  masculin

sw |> relocate(c(species, homeworld), .after = name)
  # A tibble: 4 × 11
#>   name           species homeworld height  mass hair_color skin_color  eye_color birth_y
#>   <chr>          <chr>   <chr>      <int> <dbl> <chr>      <chr>       <chr>          <d
#> 1 Luke Skywalker Human   Tatooine     172    77 blond      fair        blue            1
#> 2 C-3PO          Droid   Tatooine     167    75 NA         gold        yellow         11
#> 3 R2-D2          Droid   Naboo         96    32 NA         white, blue red             3
#> 4 Darth Vader    Human   Tatooine     202   136 none       white       yellow          4
```

For bigger changes, `mutate()` lets you:

- Create new columns that are functions of existing columns.
- Replace an existing column, by creating a new column with the same name.
- Delete a column, by setting its value to `NULL`

```
starwars |>
  select(c(name, species, height, mass)) |>
  mutate(BMI = mass / (height / 100)^2) |>
  head(4)
  # A tibble: 4 × 5
#>   name           species height  mass   BMI
#>   <chr>          <chr>    <int> <dbl> <dbl>
#> 1 Luke Skywalker Human      172    77  26.0
#> 2 C-3PO          Droid      167    75  26.9
#> 3 R2-D2          Droid       96    32  34.7
#> 4 Darth Vader    Human      202   136  33.3
```

When you want to operate on a subset of the columns with functions such as `mutate()`, the `select() |> mutate()` sequence in the above example is one option. Only the selected columns will be in the result.

Alternatively, it can be convenient to use `pick()` *within* the `mutate()` call:

```
starwars |> mutate(pick(c(name, species, height, mass)), BMI = mass / (height / 100)^2) |>
When you want to operate on a subset of the columns with functions such as `mutate()`, the
Only the selected columns will be in the result.

Alternatively, it can be convenient to use `pick()` _within_ the `mutate()` call:

```R
starwars |> mutate(pick(c(name, species, height, mass)), BMI = mass / (height / 100)^2) |>
    # A tibble: 4 × 15
#>   name         height  mass hair_color skin_color eye_color birth_year sex   gender home
#>   <chr>         <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>  <chr
#> 1 Luke Skywal…    172    77 blond      fair       blue            19   male  mascu… Tato
#> 2 C-3PO           167    75 NA         gold       yellow         112   none  mascu… Tato
#> 3 R2-D2            96    32 NA         white, bl… red             33   none  mascu… Nabo
#> 4 Darth Vader     202   136 none       white      yellow          41.9 male  mascu… Tato
   #   2 more variables: starships <list>, BMI <dbl>
```
```

Only the `pick`ed columns are used in the mutation, but all columns are returned.

6.2 data.table

6.3 Polars

6.4 DuckDB

6.5 Arrow