Data Manipulation in R

Reading and Munging Data

L. Torgo

ltorgo@dal.ca

Faculty of Computer Science / Institute for Big Data Analytics
Dalhousie University

May, 2021

ACADIA UNIVERSITY

Data Manipulation using dplyr

The Package **dplyr**

- **dplyr** is a package that greatly facilitates manipulating data in R
- It has several interesting features like:
  - Implements the most basic data manipulation operations
  - Is able to handle several data sources (e.g. standard data frames, data bases, etc.)
  - Very fast

Data tables using tbl objects

- **dplyr** defines the class **tbl** that can be seen as a generalization of a data frame, that encapsulates the possibility of the data being store in different sources
- These sources can be:
  - Standard data frames - through **tbl_df** class
  - Tables of relational data bases
  - Etc.

Data sources - internal data frames

- Data frame table (a *tibble*)
  - A wrapper for a local R data frame
  - Main advantage is printing

```
library(dplyr)
data(iris)
ir <- as_tibble(iris)
ir

## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length
##           <dbl>       <dbl>        <dbl>
##  1          5.1         3.5          1.4
##  2          4.9         3            1.4
##  3          4.7         3.2          1.3
##  4          4.6         3.1          1.5
##  5          5           3.6          1.4
##  6          5.4         3.9          1.7
##  7          4.6         3.4          1.4
##  8          5           3.4          1.5
##  9          4.4         2.9          1.4
## 10          4.9         3.1          1.5
## # ... with 140 more rows, and 2 more
## #   variables: Petal.Width <dbl>,
## #   Species <fct>
```


Data sources - relational databases

What you need to install?

- Package **dbplyr**
  - Translations of **dplyr** statements to vendor-specific SQL
- Package **DBI**
  - General interface to relational databases backends
- Specific package to interface your target database backend
  - **RMariaDB** for MySQL or MariaDB
  - **RPostgreSQL** for Postgres and Redshift
  - **RSQLite** for SQLite
  - **odbc** for connecting to several databases using the *odbc* protocol
  - **bigrquery** for Google's BigQuery

Data sources - relational databases

A small example

```
library(dplyr)
dbConn <- DBI::dbConnect(RMariaDB::MariaDB(),
            dbname="myDatabase"
            host="myorganization.com",
            user="myUser",
            password=rstudioapi::askForPassword("DB password"))

sensors <- tbl(dbConn,"sensor_values")
```

- After this initialization the object `sensors` can be used as any other **dplyr** data source.
- **dplyr** will be very conservative never pulling data from the database unless specifically told to do so.

The basic operations

- **filter** - show only a subset of the rows
- **select** - show only a subset of the columns
- **arrange** - reorder the rows
- **mutate** - add new columns
- **summarise** - summarise the values of a column

The structure of the basic operations

- First argument is a data frame table
- Remaining arguments describe what to do with the data
- Return an object of the same type as the first argument (except summarise)
- Never change the object in the first argument

Filtering rows

`filter(data, cond1, cond2, ...)` corresponds to the rows of `data` that satisfy ALL indicated conditions.

```
filter(ir,Sepal.Length > 6,Sepal.Width > 3.5)
```

```
## # A tibble: 3 x 5
##   Sepal.Length Sepal.Width Petal.Length
##          <dbl>       <dbl>        <dbl>
## 1          7.2         3.6          6.1
## 2          7.7         3.8          6.7
## 3          7.9         3.8          6.4
## # ... with 2 more variables:
## #   Petal.Width <dbl>, Species <fct>
```

```
filter(ir,Sepal.Length > 7.7 | Sepal.Length < 4.4)
```

```
## # A tibble: 2 x 5
##   Sepal.Length Sepal.Width Petal.Length
##          <dbl>       <dbl>        <dbl>
## 1          4.3         3            1.1
## 2          7.9         3.8          6.4
## # ... with 2 more variables:
## #   Petal.Width <dbl>, Species <fct>
```

Ordering rows

`arrange(data, col1, col2, ...)` re-arranges the rows of `data` by ordering them by `col1`, then by `col2`, etc.

```
arrange(ir,Species,Petal.Width)

## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length
##           <dbl>       <dbl>        <dbl>
##  1          4.9         3.1          1.5
##  2          4.8         3            1.4
##  3          4.3         3            1.1
##  4          5.2         4.1          1.5
##  5          4.9         3.6          1.4
##  6          5.1         3.5          1.4
##  7          4.9         3            1.4
##  8          4.7         3.2          1.3
##  9          4.6         3.1          1.5
## 10          5           3.6          1.4
## # ... with 140 more rows, and 2 more
## #   variables: Petal.Width <dbl>,
## #   Species <fct>
```

Ordering rows - 2

```
arrange(ir,desc(Sepal.Width),Petal.Length)

## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length
##           <dbl>       <dbl>        <dbl>
##  1          5.7         4.4          1.5
##  2          5.5         4.2          1.4
##  3          5.2         4.1          1.5
##  4          5.8         4            1.2
##  5          5.4         3.9          1.3
##  6          5.4         3.9          1.7
##  7          5.1         3.8          1.5
##  8          5.1         3.8          1.6
##  9          5.7         3.8          1.7
## 10          5.1         3.8          1.9
## # ... with 140 more rows, and 2 more
## #   variables: Petal.Width <dbl>,
## #   Species <fct>
```

Selecting columns

`select(data, col1, col2, ...)` shows the values of columns `col1`, `col2`, etc. of `data`

```
select(ir,Sepal.Length,Species)

## # A tibble: 150 x 2
##    Sepal.Length Species
##           <dbl> <fct>
##  1          5.1 setosa
##  2          4.9 setosa
##  3          4.7 setosa
##  4          4.6 setosa
##  5          5   setosa
##  6          5.4 setosa
##  7          4.6 setosa
##  8          5   setosa
##  9          4.4 setosa
## 10          4.9 setosa
## # ... with 140 more rows
```

Selecting columns - 2

```
select(ir,-(Sepal.Length:Sepal.Width))

## # A tibble: 150 x 3
##    Petal.Length Petal.Width Species
##           <dbl>       <dbl> <fct>
##  1          1.4         0.2 setosa
##  2          1.4         0.2 setosa
##  3          1.3         0.2 setosa
##  4          1.5         0.2 setosa
##  5          1.4         0.2 setosa
##  6          1.7         0.4 setosa
##  7          1.4         0.3 setosa
##  8          1.5         0.2 setosa
##  9          1.4         0.2 setosa
## 10          1.5         0.1 setosa
## # ... with 140 more rows
```

Selecting columns - 3

```
select(ir,starts_with("Sepal"))
```

```
## # A tibble: 150 x 2
##    Sepal.Length Sepal.Width
##           <dbl>       <dbl>
##  1          5.1         3.5
##  2          4.9         3
##  3          4.7         3.2
##  4          4.6         3.1
##  5          5           3.6
##  6          5.4         3.9
##  7          4.6         3.4
##  8          5           3.4
##  9          4.4         2.9
## 10          4.9         3.1
## # ... with 140 more rows
```

Adding new columns

`mutate(data, newcol1, newcol2, ...)` adds the new columns `newcol1`, `newcol2`, etc.

```
mutate(ir,sr=Sepal.Length/Sepal.Width,pr=Petal.Length/Petal.Width,rat=sr/pr)
```

```
## # A tibble: 150 x 8
##    Sepal.Length Sepal.Width Petal.Length
##           <dbl>       <dbl>        <dbl>
##  1          5.1         3.5          1.4
##  2          4.9         3            1.4
##  3          4.7         3.2          1.3
##  4          4.6         3.1          1.5
##  5          5           3.6          1.4
##  6          5.4         3.9          1.7
##  7          4.6         3.4          1.4
##  8          5           3.4          1.5
##  9          4.4         2.9          1.4
## 10          4.9         3.1          1.5
## # ... with 140 more rows, and 5 more
## #   variables: Petal.Width <dbl>,
## #   Species <fct>, sr <dbl>, pr <dbl>,
## #   rat <dbl>
```

NOTE: It does not change the original data!

Several Operations

```
select(filter(ir,Petal.Width > 2.3),Sepal.Length,Species)
```

```
## # A tibble: 6 x 2
##   Sepal.Length Species
##          <dbl> <fct>
## 1          6.3 virginica
## 2          7.2 virginica
## 3          5.8 virginica
## 4          6.3 virginica
## 5          6.7 virginica
## 6          6.7 virginica
```

Several Operations (cont.)

Function composition can become hard to understand...

```
arrange(
    select(
        filter(
            mutate(ir,sr=Sepal.Length/Sepal.Width),
            sr > 1.6),
        Sepal.Length,Species),
    Species,desc(Sepal.Length))
```

```
## # A tibble: 103 x 2
##    Sepal.Length Species
##           <dbl> <fct>
##  1          5   setosa
##  2          4.9 setosa
##  3          4.5 setosa
##  4          7   versicolor
##  5          6.9 versicolor
##  6          6.8 versicolor
##  7          6.7 versicolor
##  8          6.7 versicolor
##  9          6.7 versicolor
## 10          6.6 versicolor
## # ... with 93 more rows
```

The Chaining Operator as Alternative

```
mutate(ir,sr=Sepal.Length/Sepal.Width) %>% filter(sr > 1.6) %>%
    select(Sepal.Length,Species) %>% arrange(Species,desc(Sepal.Length))

## # A tibble: 103 x 2
##    Sepal.Length Species
##           <dbl> <fct>
##  1          5   setosa
##  2          4.9 setosa
##  3          4.5 setosa
##  4          7   versicolor
##  5          6.9 versicolor
##  6          6.8 versicolor
##  7          6.7 versicolor
##  8          6.7 versicolor
##  9          6.7 versicolor
## 10          6.6 versicolor
## # ... with 93 more rows
```

Summarizing a set of rows

`summarise(data, sumF1, sumF2, ...)` summarises the rows in `data` using the provided functions

```
summarise(ir,avgPL= mean(Petal.Length),varSW = var(Sepal.Width))

## # A tibble: 1 x 2
##   avgPL varSW
##   <dbl> <dbl>
## 1  3.76 0.190
```

Forming sub-groups of rows

`group_by(data, crit1, crit2, ...)` creates groups of rows of `data` according to the indicated criteria, applied one over the other (in case of draws)

```
sps <- group_by(ir,Species)
sps

## # A tibble: 150 x 5
## # Groups:   Species [3]
##    Sepal.Length Sepal.Width Petal.Length
##           <dbl>       <dbl>        <dbl>
##  1          5.1         3.5          1.4
##  2          4.9         3            1.4
##  3          4.7         3.2          1.3
##  4          4.6         3.1          1.5
##  5          5           3.6          1.4
##  6          5.4         3.9          1.7
##  7          4.6         3.4          1.4
##  8          5           3.4          1.5
##  9          4.4         2.9          1.4
## 10          4.9         3.1          1.5
## # ... with 140 more rows, and 2 more
## #   variables: Petal.Width <dbl>,
## #   Species <fct>
```

Summarization over groups

```
group_by(ir,Species) %>% summarise(mPL=mean(Petal.Length))

## `summarise()` ungrouping output (override with `.groups` argument)

## # A tibble: 3 x 2
##   Species      mPL
##   <fct>      <dbl>
## 1 setosa      1.46
## 2 versicolor  4.26
## 3 virginica   5.55
```

Counting

```
group_by(ir,Species) %>% tally()
```

```
## # A tibble: 3 x 2
##   Species        n
##   <fct>      <int>
## 1 setosa        50
## 2 versicolor    50
## 3 virginica     50
```

```
group_by(ir,Species) %>% summarise(n=n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 2
##   Species        n
##   <fct>      <int>
## 1 setosa        50
## 2 versicolor    50
## 3 virginica     50
```

```
count(ir,Species)
```

```
## # A tibble: 3 x 2
##   Species        n
##   <fct>      <int>
## 1 setosa        50
## 2 versicolor    50
## 3 virginica     50
```


Tops

```
group_by(ir,Species) %>% arrange(desc(Petal.Length)) %>% slice(1:2)
```

```
## # A tibble: 6 x 5
## # Groups:   Species [3]
##   Sepal.Length Sepal.Width Petal.Length
##          <dbl>       <dbl>        <dbl>
## 1          4.8         3.4          1.9
## 2          5.1         3.8          1.9
## 3          6           2.7          5.1
## 4          6.7         3            5
## 5          7.7         2.6          6.9
## 6          7.7         3.8          6.7
## # ... with 2 more variables:
## #   Petal.Width <dbl>, Species <fct>
```

```
group_by(ir,Species) %>% top_n(2,Petal.Length) # note the slight difference!
```

```
## # A tibble: 7 x 5
## # Groups:   Species [3]
##   Sepal.Length Sepal.Width Petal.Length
##          <dbl>       <dbl>        <dbl>
## 1          4.8         3.4          1.9
## 2          5.1         3.8          1.9
## 3          6.7         3            5
## 4          6           2.7          5.1
## 5          7.7         3.8          6.7
## 6          7.7         2.6          6.9
## 7          7.7         2.8          6.7
## # ... with 2 more variables:
## #   Petal.Width <dbl>, Species <fct>
```

Bottoms

```
group_by(ir,Species) %>% arrange(Sepal.Width) %>% slice(1:2)

## # A tibble: 6 x 5
## # Groups:   Species [3]
##   Sepal.Length Sepal.Width Petal.Length
##          <dbl>       <dbl>        <dbl>
## 1          4.5         2.3          1.3
## 2          4.4         2.9          1.4
## 3          5           2            3.5
## 4          6           2.2          4
## 5          6           2.2          5
## 6          4.9         2.5          4.5
## # ... with 2 more variables:
## #   Petal.Width <dbl>, Species <fct>

group_by(ir,Species) %>% top_n(-2,Sepal.Width)  # note the slight difference!

## # A tibble: 10 x 5
## # Groups:   Species [3]
##    Sepal.Length Sepal.Width Petal.Length
##           <dbl>       <dbl>        <dbl>
##  1          4.4         2.9          1.4
##  2          4.5         2.3          1.3
##  3          5           2            3.5
##  4          6           2.2          4
##  5          6.2         2.2          4.5
##  6          4.9         2.5          4.5
##  7          6.7         2.5          5.8
##  8          5.7         2.5          5
##  9          6           2.2          5
## # ... with 2 more variables:
## #   Petal.Width <dbl>, Species <fct>
```


Combining Data

- Functions `bind_rows()` and `bind_cols()` can be used to glue tables row- and column-wise, respectively
- Functions `left_join()`, `right_join()`, `inner_join()` and `full_join()` allow joining data from different tables that share some common information (keys)

A Good Cheat Sheet to Have Around

RStudio maintains a series of very nice cheat sheets that are handy when you forget some things. They can be found at this URL:

`https://www.rstudio.com/resources/cheatsheets/`

In particular there is one about **dplyr** at this URL:

`https://github.com/rstudio/cheatsheets/raw/master/source/pdfs/data-transformation-cheatsheet.pdf`

Hands On Data Manipulation with dplyr

Package **mlbench** (an extra package that you need to install) contains several data sets (from UCI repository). Load the Ozone data set and check its help page to understand what this data is about. Answer these questions:

1. Create a data frame table with the data for easier manipulation
2. What is the average Humidity per Month?
3. Show the information of Mondays
4. For each combination of *Month* and *Day of the Week* obtain the maximum and minimum temperature at the *Sand* location