Data Science in R

Section: Tidy Data Manipulation

Prof. Dr. Nicolas Pröllochs
University of Giessen
nicolas.proellochs@wi.jlug.de

Today's Lecture

1. **Tidy Data Manipulation:** Learn how to manipulate data using the `dplyr` R-package
2. **Pipe Operator:** Learn how increase code readability using pipes
3. **Joins:** Learn how to efficiently join separate datasets in R

Motivation: data manipulation in base R

- Filtering observation

```
df[df$var1 == 1 & df$var2 == 0, ]
```

- Selecting columns

```
df[, c("var1", "var2")]
```

Data manipulation in base R is tedious!

‘Tidy’ data analysis

- 50-80% of data analysis is spent on **cleaning and preparing data**
- The **R-package `dplyr`** was developed to make **data cleaning/preparation/reshaping** more efficient and easy
- Advantages
  - Very intuitive, once you understand the basics
  - Very fast
  - Easy for those migrating from the SQL world
  - “Code the way you think”

`dplyr` is part of the `tidyverse` package collection → revolutionized working with data in R

Prepare your R environment

- Install required R-package

```
install.packages("dplyr")
```

- Remember that you'll also want to load the package...

```
library(dplyr)
```

Most important `dplyr` functions

- Accessing information
  1. Extract existing observations: `filter()`
  2. Reorder observations: `arrange()`
  3. Extract existing variables: `select()`
  4. Calculate/derive new variables: `mutate()`
  5. `summarise()` multiple values into one value
  6. `group_by()` groups factors together
- Joins
  - `left_join()`, `right_join()`, `anti_join()`, `inner_join()`, and more

Creating a dataframe

```
df <- data.frame(color = c("blue", "black", "blue", "blue", "black"),
                 value = 1:5)
df
```

```
##   color value
## 1  blue     1
## 2 black     2
## 3  blue     3
## 4  blue     4
## 5 black     5
```

Pick rows with `filter()`

- Filter the rows that are blue using `filter(data, ...)`, where `data` is a dataframe and `...` are filtering conditions

```
filter(df, color == "blue")
```

```
##   color value
## 1  blue     1
## 2  blue     3
## 3  blue     4
```

- Note: This is equivalent to the following base R code

```
df[df$color == "blue", ]
```

Filter based on certain values

- Filter the rows with value=1 or value=4

```
filter(df, value %in% c(1,4))
```

```
##   color value
## 1  blue     1
## 2  blue     4
```

- Note: This is equivalent to the following base R code

```
df[df$value %in% c(1,4), ]
```

Exercise

- Load dataset `mpg` from the `ggplot2` package

```
library(ggplot2)
data(mpg)
head(mpg, 3)
```

```
## # A tibble: 3 x 11
##   manufacturer model displ  year   cyl trans      drv     cty   hwy fl    class
##   <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
## 1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p     compa~
## 2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p     compa~
## 3 audi         a4      2    2008     4 manual(m6) f        20    31 p     compa~
```

- **You do**: Find all the cars from Audi or Volkswagen

```
# Insert your code
```

Exercise: Solution

- Find all the cars from Audi or Volkswagen:

```
filter(mpg, manufacturer %in% c("audi", "volkswagen"))
```

```
## # A tibble: 45 x 11
##    manufacturer model    displ  year   cyl trans   drv     cty   hwy fl    class
##    <chr>        <chr>    <dbl> <int> <int> <chr>   <chr> <int> <int> <chr> <chr>
##  1 audi         a4         1.8  1999     4 auto(l~ f        18    29 p     comp~
##  2 audi         a4         1.8  1999     4 manual~ f        21    29 p     comp~
##  3 audi         a4         2    2008     4 manual~ f        20    31 p     comp~
##  4 audi         a4         2    2008     4 auto(a~ f        21    30 p     comp~
##  5 audi         a4         2.8  1999     6 auto(l~ f        16    26 p     comp~
##  6 audi         a4         2.8  1999     6 manual~ f        18    26 p     comp~
##  7 audi         a4         3.1  2008     6 auto(a~ f        18    27 p     comp~
##  8 audi         a4 quat~   1.8  1999     4 manual~ 4        18    26 p     comp~
##  9 audi         a4 quat~   1.8  1999     4 auto(l~ 4        16    25 p     comp~
## 10 audi         a4 quat~   2    2008     4 manual~ 4        20    28 p     comp~
## # ... with 35 more rows
```

Sort values with `arrange()`

- **Arrange the data** to be sorted by a particular column using `arrange()`
- Example: sort data by number of cylinders (`cyl`) in ascending order

```
arrange(mpg, cyl)
```

```
## # A tibble: 234 x 11
##    manufacturer model displ  year   cyl trans      drv     cty   hwy fl    class
##    <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
## 1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p     compa~
## 2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p     compa~
## 3 audi         a4      2    2008     4 manual(m6) f        20    31 p     compa~
## 4 audi         a4      2    2008     4 auto(av)   f        21    30 p     compa~
## # ... with 230 more rows
```

Observations can be sorted in descending order via `arrange(mpg, desc(cyl))`

Three ways to pick columns with `select()`

- List the column names you **do** want

```
select(mpg, manufacturer, model)    # Pick only manufacturer and model columns
select(mpg, manufacturer:model)      # Pick manufacturer to model columns
```

- List the column names you **don't** want

```
select(mpg, -manufacturer)             # Remove one column, manufacturer
select(mpg, -c(manufacturer, model))    # Remove manufacturer and model
```

- Select column indices

```
select(mpg, 1:2)       # Pick only columns 1 & 2
```

Helper functions with `select()`

- Select columns that:
  - `contains()`: Are based on a character string
  - `starts_with()` or `ends_with()`: Contain part of a character string
  - `one_of()`: Names are in a group of names
  - `matches()`: Name matches a regular expression
- Example:

```
select(mpg, contains("m"))
```

```
## # A tibble: 234 x 2
##   manufacturer model
##   <chr>        <chr>
## 1 audi         a4
## 2 audi         a4
## # ... with 232 more rows
```

Exercise

- **You do**:
  - Select columns `manufacturer`, `model`, `year` and `cyl` from the `mpg` dataset
  - Which car model has the lowest number of cylinders?

```
# Insert your code
```

Exercise: Solution

- Select columns `manufacturer`, `model`, `year` and `cyl` from the `mpg` dataset

```
mpg_subset <- select(mpg, manufacturer, model, year, cyl)
```

- Which car model has the lowest number of cylinders?

```
arrange(mpg_subset, cyl)
```

```
## # A tibble: 234 x 4
##    manufacturer model  year   cyl
##    <chr>        <chr> <int> <int>
## 1 audi         a4     1999     4
## 2 audi         a4     1999     4
## # ... with 232 more rows
```

Create new variables with `mutate()`

- The function `mutate()` allows one to **add new variables** as a function of existing variables
- Example:

```
mutate(df, double = 2 * value)
```

```
##   color value double
## 1  blue     1      2
## 2 black     2      4
## 3  blue     3      6
## 4  blue     4      8
## 5 black     5     10
```

Exercise

- **You do:**
  - Add a column to the `mpg` dataset with a flag for “non-energy efficient” cars
  - Cars are non-energy efficient if highway miles per gallon (column `hwy`) is below 30 mpg

```
# Insert your code
```

Exercise: Solution

- Add a column to the mpg data set with a flag for “non-energy efficient” cars

```
mutate(mpg, not.energ.eff = ifelse(hwy < 30, 1, 0))
```

```
## # A tibble: 234 x 12
##   manufacturer model displ  year   cyl trans drv     cty   hwy fl    class
##   <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
## 1 audi         a4      1.8  1999     4 auto~ f        18    29 p     comp~
## 2 audi         a4      1.8  1999     4 manu~ f        21    29 p     comp~
## 3 audi         a4      2    2008     4 manu~ f        20    31 p     comp~
## 4 audi         a4      2    2008     4 auto~ f        21    30 p     comp~
## 5 audi         a4      2.8  1999     6 auto~ f        16    26 p     comp~
## # ... with 229 more rows, and 1 more variable: not.energ.eff <dbl>
```

`summarise()` multiple values to one value

- `summarise()` summarises **multiple values to a single value**
- Example:

```
summarise(mpg, mean_cyl = mean(cyl), # Mean number of cylinders
          n = n())                   # Number of values measured
```

```
## # A tibble: 1 x 2
##   mean_cyl     n
##      <dbl> <int>
## 1     5.89   234
```

Further summary statistics

Summary Statistic	R expression
Average / mean	`mean()`
Maximum	`max()`
Minimum	`min()`
Median	`median()`
Standard deviation	`sd()`
Count	`n()`
Count Distinct	`n_distinct()`

Exercise

- **You do:**
  - Calculate the median number and standard deviation of the column `hwy` in the `mpg` dataset

```
# Insert your code
```

Exercise: Solution

- Calculate the median number and standard deviation of the column `hwy` in the `mpg` dataset

```
summarise(mpg, median_hwy = median(hwy),
          sd_hwy = sd(hwy))
```

```
## # A tibble: 1 x 2
##   median_hwy sd_hwy
##        <dbl>  <dbl>
## 1         24   5.95
```

Grouping observations using `group_by()`, then summarize

- `summarise()` is particularly powerful when used **in combination** with `group_by()`

df

color	value
blue	1
black	2
blue	3
blue	4
black	5

color	total
blue	8
black	7

- Example: grouping by color

```
df_grouped <- group_by(df, color)
summarize(df_grouped, total = sum(value))
```

```
## # A tibble: 2 x 2
##   color total
##   <fct> <int>
## 1 black     7
## 2 blue      8
```

Exercise

- **You do:**
  - Calculate the mean number of cylinders (column `cyl`) for each manufacturer in the `mpg` dataset

```
# Insert your code
```

Exercise: Solution

- Calculate the mean number of cylinders (column `cyl`) for each manufacturer in the `mpg` dataset

```
mpg_grouped <- group_by(mpg, manufacturer)

summarize(
  mpg_grouped,
  mean_cyl = mean(cyl)
  )
```

```
## # A tibble: 15 x 2
##   manufacturer mean_cyl
##   <chr>           <dbl>
## 1 audi             5.22
## 2 chevrolet        7.26
## 3 dodge            7.08
## 4 ford             7.2
## 5 honda            4
## # ... with 10 more rows
```

The pipe operator: `%>%`

- The **pipe operator** `%>%` enables to **"chain" functions together**
- R's copy of the unix pipe: `|`
- Will directly move the output of one function to another function
- **Reduces the number of variables needed**
- **Increases code readability**

You don't have to use the `%>%` in `dplyr`, however it arguably makes your syntax much more readable

Example: Pipes

- Example:

```
mpg %>%
  filter(manufacturer == "audi") %>%
  arrange(cyl)
```

```
## # A tibble: 18 x 11
##   manufacturer model    displ  year   cyl trans   drv     cty   hwy fl    class
##   <chr>        <chr>    <dbl> <int> <int> <chr>   <chr> <int> <int> <chr> <chr>
## 1 audi         a4         1.8  1999     4 auto(l~ f        18    29 p     compa~
## 2 audi         a4         1.8  1999     4 manual~ f        21    29 p     compa~
## 3 audi         a4         2    2008     4 manual~ f        20    31 p     compa~
## 4 audi         a4         2    2008     4 auto(a~ f        21    30 p     compa~
## 5 audi         a4 quat~   1.8  1999     4 manual~ 4        18    26 p     compa~
## # ... with 13 more rows
```

Exercise

- **You do:**
  - Select the columns `manufacturer` and `cyl` from the `mpg` dataset.
  - Subsequently, calculate the number of observations for each manufacturer and sort the dataframe in descending order.
  - Use the pipe operator to chain the functions together.

```
# Insert your code
```

Exercise: Solution

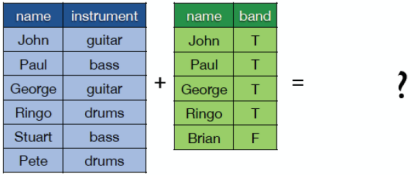
- Select the columns `manufacturer` and `cyl` from the `mpg` dataset. Subsequently, calculate the number of observations for each manufacturer and sort the dataframe in descending order. Use the pipe operator to chain the functions together.

```
mpg %>% select(manufacturer, cyl) %>%
  group_by(manufacturer) %>%
  summarize(N = n()) %>%
  arrange(desc(N))
```

```
## # A tibble: 15 x 2
##   manufacturer     N
##   <chr>        <int>
## 1 dodge           37
## 2 toyota          34
## 3 volkswagen      27
## 4 ford            25
## 5 chevrolet       19
## # ... with 10 more rows
```

Joining datasets together

- Joining two separate datasets



name	instrument
John	guitar
Paul	bass
George	guitar
Ringo	drums
Stuart	bass
Pete	drums

+

name	band
John	T
Paul	T
George	T
Ringo	T
Brian	F

=

?

- Example: Creating two dataframes

```
x <- data.frame(name = c("John", "Paul", "George", "Ringo", "Stuart", "Pete"),
                instrument = c("guitar", "bass", "guitar", "drums", "bass", "drums"))

y <- data.frame(name = c("John", "Paul", "George", "Ringo", "Brian"),
                band = c("TRUE", "TRUE", "TRUE", "TRUE", "FALSE"))
```

inner_join

- `inner_join(a, b, by = "V1")`: Join data but only keep rows in both sets
- Example

```
inner_join(x, y, by = "name")
```

```
##      name instrument band
## 1    John     guitar TRUE
## 2    Paul       bass TRUE
## 3 George     guitar TRUE
## 4  Ringo      drums TRUE
```

left_join()

- `left_join(a, b, by = "V1")`: Join matching rows from **b** to **a**
- Example

```
left_join(x, y, by = "name")
```

```
##      name instrument band
## 1    John     guitar TRUE
## 2    Paul       bass TRUE
## 3  George     guitar TRUE
## 4   Ringo      drums TRUE
## 5  Stuart       bass <NA>
## 6    Pete      drums <NA>
```

`right_join(a, b, by = "V1")`: Join matching rows from **a** to **b**

anti_join()

- `anti_join(a, b, by = "V1")`: Keep all rows that *do not* have a match in **b**
- Example

```
anti_join(x, y, by = "name")
```

```
##     name instrument
## 1 Stuart       bass
## 2   Pete      drums
```

`semi_join(a, b, by = "V1")`: Keep all rows that *have a match* in **b**

Wrap-up

- Key takeaways
  - The R-package `dplyr` makes data **cleaning/preparation/reshaping** more efficient and easy
  - The **pipe operator** `%>%` enables to "chain" functions together
  - **Join functions** as provided by the `dplyr` package allow one to easily join separate datasets in R
- Further reading
  - Data wrangling cheatsheet with tidyr & dplyr
  - Book: R for Data Science (O'Reilly, 2017, by H. Wickham & G. Grolemund)