Manipulating and analyzing data with dplyr

Learning Objectives

- Select certain columns in a data frame with the **dplyr** function `select`.
- Select certain rows in a data frame according to filtering conditions with the **dplyr** function `filter` .
- Link the output of one **dplyr** function to the input of another function with the 'pipe' operator `%>%`.
- Add new columns to a data frame that are functions of existing columns with `mutate`.
- Use `summarize` and `group_by` to split a data frame into groups of observations, apply summary statistics for each group, and then combine the results.

Data Manipulation using `dplyr`

- Bracket subsetting `[,]` (with logical operators) is handy, but it can be cumbersome and difficult to read, especially for complicated operations.
- **dplyr** is a package for making tabular data manipulation easier.
- It pairs nicely with **tidyr** which enables you to swiftly convert between different data formats for plotting and analysis.
- To learn more about **dplyr** and **tidyr** , you may want to check out this handy data transformation with **dplyr** cheatsheet and this one about **tidyr**.

```
## load dplyr
library("dplyr")
```

Sample data

```
## load sample data
NUTS2.DF <- read.csv("datasets/NUTS2data.csv")
# summary(NUTS2.DF)
str(NUTS2.DF)
```

```
## 'data.frame':    791 obs. of  6 variables:
##  $ Year       : int  2010 2010 2010 2010 2010 2010 2010 2010 2010 2011 ...
##  $ NUTS2      : Factor w/ 113 levels "AT11","AT12",..: 1 3 7 9 4 8 6 2 5 9 ...
##  $ NUTS0      : Factor w/ 11 levels "AT","BE","CZ",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ GDP_MIO_EUR: num  7431 86676 24259 14915 18396 ...
##  $ TotPopNr   : int  283697 1689995 526730 368366 557998 704662 1409253 1605897 1205045 369300 ...
##  $ Area       : num  4449 265 7480 2634 9393 ...
```

Variable	Description
Year	time identification of the observation 2010 - 2016

Variable	Description
NUTS2	NUTS2 geographic identification of the observation
NUTS0	State-level identification (AT BE CZ DE DK HU LU NL PL SI SK)
GDP_MIO_EUR	GDP in Mio EUR per NUTS2 per Year
TotPopNr	Number of inhabitants
Area	geographic area in km sq.

Basic `dplyr` functionality

`select()` - subset columns

- The first argument to this function is the data frame: `NUTS2.DF`,
- Subsequent arguments are the columns to keep.

```
DF2 <- select(NUTS2.DF, Year, NUTS2, TotPopNr)
head(DF2,10)
```

```
##    Year NUTS2 TotPopNr
## 1  2010  AT11   283697
## 2  2010  AT13  1689995
## 3  2010  AT32   526730
## 4  2010  AT34   368366
## 5  2010  AT21   557998
## 6  2010  AT33   704662
## 7  2010  AT31  1409253
## 8  2010  AT12  1605897
## 9  2010  AT22  1205045
## 10 2011  AT34   369300
```

- To select all columns *except* certain ones, put a "-" in front of the variable to exclude it.

```
DF3 <- select(NUTS2.DF, -TotPopNr, -Area)
head(DF3,10)
```

```
##    Year NUTS2 NUTS0 GDP_MIO_EUR
## 1  2010  AT11    AT    7430.663
## 2  2010  AT13    AT   86676.281
## 3  2010  AT32    AT   24258.931
## 4  2010  AT34    AT   14914.577
## 5  2010  AT21    AT   18395.829
## 6  2010  AT33    AT   28390.282
## 7  2010  AT31    AT   55016.641
## 8  2010  AT12    AT   51349.013
## 9  2010  AT22    AT   41686.266
## 10 2011  AT34    AT   15256.561
```

`filter()` - subset rows on conditions

- To choose rows based on a specific criteria, use `filter()`:

```
# Choose all records for Slovenia
filter(NUTS2.DF, NUTS0 == "SI")
```

```
##    Year NUTS2 NUTS0 GDP_MIO_EUR TotPopNr     Area
## 1  2010  SI03    SI    16927.34  1099674 12017.44
## 2  2010  SI04    SI    21696.14   947302  7714.61
## 3  2011  SI03    SI    16999.27  1098506 12017.44
## 4  2011  SI04    SI    21510.28   951683  7714.61
## 5  2012  SI03    SI    16142.13  1099196 12017.44
## 6  2012  SI04    SI    20483.25   956300  7714.61
## 7  2013  SI04    SI    20151.39   961623  7714.61
## 8  2013  SI03    SI    15943.23  1097198 12017.44
## 9  2014  SI03    SI    16468.19  1094709 12017.44
## 10 2014  SI04    SI    20859.38   966376  7714.61
## 11 2015  SI03    SI    17077.00  1093545 12017.44
## 12 2015  SI04    SI    21760.00   969329  7714.61
## 13 2016  SI03    SI    17679.52  1092193 12017.44
## 14 2016  SI04    SI    22799.20   971995  7714.61
```

```
# Choose all records for Slovenia AND year 2011
filter(NUTS2.DF, Year == 2011, NUTS0 == "SI")
```

```
##   Year NUTS2 NUTS0 GDP_MIO_EUR TotPopNr     Area
## 1 2011  SI03    SI    16999.27  1098506 12017.44
## 2 2011  SI04    SI    21510.28   951683  7714.61
```

Pipes %>%

What if you want to select and filter at the same time? There are three ways to do this:

- use intermediate steps,
- nested functions,
- pipes.

Sample exercise: Get TotPopNr data (*plus id info* Year and NUTS2) for Slovenia, year 2011 and older

With **intermediate steps**, you create a temporary data frame and use that as input to the next function, like this:

```
DF4 <- filter(NUTS2.DF, Year <= 2011, NUTS0 == "SI")
DF5 <- select(DF4, Year, NUTS2, TotPopNr)
DF5
```

```
##   Year NUTS2 TotPopNr
## 1 2010  SI03  1099674
## 2 2010  SI04   947302
## 3 2011  SI03  1098506
## 4 2011  SI04   951683
```

- This is readable, but can clutter up your workspace with lots of objects that you have to name individually. With multiple steps, that can be hard to keep track of.

You can also **nest functions** (i.e. one function inside of another), like this:

```
select(filter(NUTS2.DF, Year <= 2011, NUTS0 == "SI"), Year, NUTS2, TotPopNr)
```

```
##   Year NUTS2 TotPopNr
## 1 2010  SI03  1099674
## 2 2010  SI04   947302
## 3 2011  SI03  1098506
## 4 2011  SI04   951683
```

- This is handy, but can be difficult to read if too many functions are nested, as R evaluates the expression from the inside out (in this case, filtering, then selecting).

The last option, **pipes**, are a recent addition to R. Pipes let you take the output of one function and send it directly to the next, which is useful when you need to do many things to the same dataset.

Pipes in R look like `%>%` and are available within `{dplyr}` (as well as other packages).

- If you use RStudio, you can type the pipe with Ctrl + Shift + M if you have a PC
- or Cmd + Shift + M if you have a Mac.

```
NUTS2.DF %>%
  filter(Year <= 2011, NUTS0 == "SI") %>%
  select(Year, NUTS2, TotPopNr)
```

```
##   Year NUTS2 TotPopNr
## 1 2010  SI03  1099674
## 2 2010  SI04   947302
## 3 2011  SI03  1098506
## 4 2011  SI04   951683
```

- The pipe operator `%>%` takes the object on its left and passes it as the first argument to the function on its right, we don't need to explicitly include the data frame as an argument to the `filter()` and `select()` functions any more.
- In the above code, we use the pipe to send the `NUTS2.DF` dataset first through `filter()` to keep rows where `Year` is $\leq$ 2011 AND `NUTS0 == "SI"`, then through `select()` to keep only the `Year`, `NUTS2` and `TotPopNr` columns.
- Some may find it helpful to read the pipe like the word **then**.
- The `dplyr` functions by themselves are somewhat simple, but by combining them into linear workflows with the pipe, we can accomplish more complex manipulations of data frames.
- If we want to create a new object with this smaller version of the data, we can assign it a new name:

```
NUTS.SI <- NUTS2.DF %>%
  filter(Year <= 2011, NUTS0 == "SI") %>%
  select(Year, NUTS2, TotPopNr)
NUTS.SI
```

```
##   Year NUTS2 TotPopNr
## 1 2010  SI03  1099674
## 2 2010  SI04   947302
## 3 2011  SI03  1098506
## 4 2011  SI04   951683
```

- Note that the final data frame is the leftmost part of the piping expression.

Quick exercise 1:

Complete `R` script below as follows, using the `NUTS2.DF` data frame:

- Filter GDP data: restrict only to Austria (`"AT"`) AND Years 2015 or later,
- Select columns: Year, NUTS2, GDP_MIO_EUR,
- Use the pipe syntax

```
# Uncomment and complete the task
#NUTS2.DF %>%
```

`mutate()` - create new columns using information in other columns

```
# For Slovenia, calculate GDP per capita (in EUR)
NUTS2.DF %>%
  filter(NUTS0 == "SI") %>%
  mutate(GDPpc = (GDP_MIO_EUR/TotPopNr)*1000000) %>%
  head(10) # pipes work with non-dplyr commands as well (if dplyr is loaded)
```

```
##    Year NUTS2 NUTS0 GDP_MIO_EUR TotPopNr     Area    GDPpc
## 1  2010  SI03    SI    16927.34  1099674 12017.44 15393.05
## 2  2010  SI04    SI    21696.14   947302  7714.61 22903.09
## 3  2011  SI03    SI    16999.27  1098506 12017.44 15474.90
## 4  2011  SI04    SI    21510.28   951683  7714.61 22602.36
## 5  2012  SI03    SI    16142.13  1099196 12017.44 14685.40
## 6  2012  SI04    SI    20483.25   956300  7714.61 21419.27
## 7  2013  SI04    SI    20151.39   961623  7714.61 20955.61
## 8  2013  SI03    SI    15943.23  1097198 12017.44 14530.86
## 9  2014  SI03    SI    16468.19  1094709 12017.44 15043.45
## 10 2014  SI04    SI    20859.38   966376  7714.61 21585.16
```

Quick exercise 2:

Complete `R` script below as follows:

Show population dentisty by NUTS2 regions

- Use (filter for) year 2016,
- Calculate population density `PopDens` (TotPopNr / Area) ,
- Show columns: NUTS2, `PopDens`,
- Use the pipe syntax
- Show the first 15 rows in your Rmd output,

```
# Uncomment and complete the task
#NUTS2.DF %>%
```

`group_by()` and `summarize()` - summary on grouped data

```r
# Calculate average value of GDP per capita at the State level, year 2016
# .. serves for illustration only - NUTS2 to NUTS0 averages are not weighted by population
NUTS2.DF %>%
  filter(Year == 2016) %>%
  mutate(GDPpc = (GDP_MIO_EUR/TotPopNr)*1000000) %>%
  group_by(NUTS0) %>%
  summarize(mean_GDPpc = mean(GDPpc, na.rm = TRUE))
```

```
## # A tibble: 11 x 2
##    NUTS0 mean_GDPpc
##    <fct>      <dbl>
##  1 AT        39569.
##  2 BE        35128.
##  3 CZ        16636.
##  4 DE        36685.
##  5 DK        44998.
##  6 HU        10254.
##  7 LU        91946.
##  8 NL        37753.
##  9 PL        10180.
## 10 SI        19822.
## 11 SK        18155.
```

- Note the output is not a `data.frame` table, but a `tibble` - `{dplyr}` / `{tidyverse}` specific format.
- Many data analysis tasks can be approached using the *split-apply-combine* paradigm: split the data into groups, apply some analysis to each group, and then combine the results.
- `group_by()` is often used together with `summarize()`, which collapses each group into a single-row summary of that group. `group_by()` takes as arguments the column names that contain the **categorical** variables for which you want to calculate the summary statistics.

---

group_by() and mutate()

- May be used for calculations on grouped data,
- Easy to calculate lags and individual means for panel data

```r
# For Slovenia, calculate first lag of GDP and individual means (over time) for TotPopNr
NUTS2.DF %>%
  select(-Area) %>%
  filter(NUTS0 == "SI") %>%
  group_by(NUTS2) %>%
  mutate(GDP_lag1 = lag(GDP_MIO_EUR, k = 1), PopAvg = mean(TotPopNr))
```

```
## # A tibble: 14 x 7
## # Groups:   NUTS2 [2]
##     Year NUTS2 NUTS0 GDP_MIO_EUR TotPopNr GDP_lag1   PopAvg
##    <int> <fct> <fct>       <dbl>    <int>    <dbl>    <dbl>
##  1  2010 SI03  SI         16927.  1099674       NA  1096432.
##  2  2010 SI04  SI         21696.   947302       NA   960658.
##  3  2011 SI03  SI         16999.  1098506   16927. 1096432.
##  4  2011 SI04  SI         21510.   951683   21696.  960658.
##  5  2012 SI03  SI         16142.  1099196   16999. 1096432.
##  6  2012 SI04  SI         20483.   956300   21510.  960658.
```

```
##  7  2013 SI04  SI         20151.   961623   20483.  960658.
##  8  2013 SI03  SI         15943.  1097198   16142. 1096432.
##  9  2014 SI03  SI         16468.  1094709   15943. 1096432.
## 10  2014 SI04  SI         20859.   966376   20151.  960658.
## 11  2015 SI03  SI         17077   1093545   16468. 1096432.
## 12  2015 SI04  SI         21760    969329   20859.  960658.
## 13  2016 SI03  SI         17680.  1092193   17077  1096432.
## 14  2016 SI04  SI         22799.   971995   21760   960658.
```

`arrange()` - sort results

```
# For Slovenia, calculate first lag of GDP and sort: fist by region, then by time
NUTS2.DF %>%
  select(-Area, -TotPopNr) %>%
  filter(NUTS0 == "SI") %>%
  group_by(NUTS2) %>%
  mutate(GDP_lag1 = lag(GDP_MIO_EUR, k = 1)) %>%
  arrange(NUTS2,Year) # sorts by NUTS2, then by Year - both ascending
```

```
## # A tibble: 14 x 5
## # Groups:   NUTS2 [2]
##     Year NUTS2 NUTS0 GDP_MIO_EUR GDP_lag1
##    <int> <fct> <fct>       <dbl>    <dbl>
##  1  2010 SI03  SI         16927.      NA
##  2  2011 SI03  SI         16999.   16927.
##  3  2012 SI03  SI         16142.   16999.
##  4  2013 SI03  SI         15943.   16142.
##  5  2014 SI03  SI         16468.   15943.
##  6  2015 SI03  SI         17077    16468.
##  7  2016 SI03  SI         17680.   17077
##  8  2010 SI04  SI         21696.      NA
##  9  2011 SI04  SI         21510.   21696.
## 10  2012 SI04  SI         20483.   21510.
## 11  2013 SI04  SI         20151.   20483.
## 12  2014 SI04  SI         20859.   20151.
## 13  2015 SI04  SI         21760    20859.
## 14  2016 SI04  SI         22799.   21760
```

- To sort in descending order, use `desc()`.
- e.g. `arrange(desc(NUTS2),Year)`
- You can use `ungroup()` in the pipe for removing the grouping (e.g. for subsequent analysis).

Joining data from multiple datasets

Read in additional dataset

Variable	Description
Year	time identification of the observation 2011 - 2016 **(no 2010)**

Variable	Description
NUTS2	NUTS2 id **(same 113 regions)**
Unem	Unemployment rate in %

```
## load sample data
Unem <- read.csv("datasets/NUTS2data2.csv")
str(Unem)
```

```
## 'data.frame':    678 obs. of  3 variables:
##  $ Year : int  2011 2011 2011 2011 2011 2011 2011 2011 2011 2012 ...
##  $ NUTS2: Factor w/ 113 levels "AT11","AT12",..: 9 2 8 7 5 1 6 3 4 9 ...
##  $ Unem : num  4.1 4.5 2.7 2.9 3.6 3.8 3.4 8.1 4.3 4 ...
```

`left_join()` - joins two datasets

Start with `NUTS2.DF` and *append* `Unem` dataset

```
# Note the missing 2010 Unem values
NewDF <- left_join(NUTS2.DF, Unem, by = c("Year", "NUTS2"))
str(NewDF)
```

```
## 'data.frame':    791 obs. of  7 variables:
##  $ Year       : int  2010 2010 2010 2010 2010 2010 2010 2010 2010 2011 ...
##  $ NUTS2      : Factor w/ 113 levels "AT11","AT12",..: 1 3 7 9 4 8 6 2 5 9 ...
##  $ NUTS0      : Factor w/ 11 levels "AT","BE","CZ",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ GDP_MIO_EUR: num  7431 86676 24259 14915 18396 ...
##  $ TotPopNr   : int  283697 1689995 526730 368366 557998 704662 1409253 1605897 1205045 369300 ...
##  $ Area       : num  4449 265 7480 2634 9393 ...
##  $ Unem       : num  NA NA NA NA NA NA NA NA NA 4.1 ...
```

```
# Show output - head of the table only
NewDF %>%
 arrange(NUTS2,Year) %>%
 head(12)
```

```
##    Year NUTS2 NUTS0 GDP_MIO_EUR TotPopNr      Area Unem
## 1  2010  AT11    AT    7430.663   283697  4449.336   NA
## 2  2011  AT11    AT    7512.587   284581  4449.336  3.8
## 3  2012  AT11    AT    7691.906   285782  4449.336  4.6
## 4  2013  AT11    AT    7710.954   286691  4449.336  4.3
## 5  2014  AT11    AT    7778.226   287416  4449.336  4.8
## 6  2015  AT11    AT    8023.000   288356  4449.336  5.2
## 7  2016  AT11    AT    8082.599   291011  4449.336  5.7
## 8  2010  AT12    AT   51349.013  1605897 20165.283   NA
## 9  2011  AT12    AT   51966.792  1609474 20165.283  4.5
## 10 2012  AT12    AT   52013.577  1614455 20165.283  4.6
## 11 2013  AT12    AT   51621.152  1618592 20165.283  5.0
## 12 2014  AT12    AT   52414.315  1625485 20165.283  5.1
```

- All observations in the `left` dataset (`NUTS2.DF`) are preserved.
- `NA` generated if `Unem` observation for a given `"Year", "NUTS2"` combination is not available

---

Alternative ordering od data.frames to join: Start with `Unem` and *append* `NUTS2.DF` dataset

```
# Note the missing 2010 Unem values
# Show output - head of the table only
Unem %>%
  left_join(NUTS2.DF, by = c("Year", "NUTS2")) %>%
  arrange(NUTS2,Year) %>%
  head(12)
```

```
##    Year NUTS2 Unem NUTS0 GDP_MIO_EUR TotPopNr      Area
## 1  2011  AT11  3.8    AT    7512.587   284581  4449.336
## 2  2012  AT11  4.6    AT    7691.906   285782  4449.336
## 3  2013  AT11  4.3    AT    7710.954   286691  4449.336
## 4  2014  AT11  4.8    AT    7778.226   287416  4449.336
## 5  2015  AT11  5.2    AT    8023.000   288356  4449.336
## 6  2016  AT11  5.7    AT    8082.599   291011  4449.336
## 7  2011  AT12  4.5    AT   51966.792  1609474 20165.283
## 8  2012  AT12  4.6    AT   52013.577  1614455 20165.283
## 9  2013  AT12  5.0    AT   51621.152  1618592 20165.283
## 10 2014  AT12  5.1    AT   52414.315  1625485 20165.283
## 11 2015  AT12  5.2    AT   53739.000  1636778 20165.283
## 12 2016  AT12  5.2    AT   54433.990  1653691 20165.283
```

- Note the changed ordering of columns.
- All observations in the `left` dataset (`Unem`) are preserved.
- Observations for `2010` in `NUTS2.DF` are NOT *imported*, as there is no such combination of `"Year", "NUTS2"` in the `Unem` dataset.
- `inner_join()` , `right_join()` are available in `dplyr` package.
- sometimes, `merge()` from the `base` packge may be a reasonable alternative.

This worksheet draws from

- Manipulating, analyzing and exporting data with tidyverse
- Data wrangling webinar