Introducing the Tidyverse

Matteo Fasiolo

University of Bristol

Overview

What is the "Tidyverse"?

The Tidyverse is a set of inter-compatible R packages sharing the same programming philosophy.

What does it offer?

A coherent and consistent R programming framework for:

1. `readr`: loading data into `R`
2. `tidyr`: putting your data in a tidy format
3. `dplyr`: manipulating data
4. `ggplot2`: building data visualizations
5. `purrr`: doing functional programming in `R`

As well as for more specific purposes:

1. `stringr` (strings), `forcats` (factor variables), `lubridate` (dates and times), ...

Overview

You will probably never use the whole Tidyverse.

Here we cover the basics:

1. `magrittr`: how pipes %>% enhance code readability
2. `ggplot2`: how layered graphics work
3. `dplyr`: how to modify dataframes in a clear and consistent manner
4. `tidyr`: how to reshape your data for plotting and modelling

Why do you need to know this:

1. by manipulating **real data** in base `R` (e.g., `tapply`, `sapply`, ...) routinely, you might go crazy
2. industry jobs
3. Tidyverse packages illustrates many programming concepts that are widely useful

Online material 1: pipes

The pipe operator %>% is provided by `magrittr`.

Consider plotting $\sqrt{|\cos(x)|}$ on a grid:

```
x <- seq(0, 2*pi, by = 0.01)

plot( sqrt ( abs( cos(x) ) ) )
```

The piped equivalent:

```
x %>% cos %>% abs %>% sqrt %>% plot
```

How does this work? Consider `f(a1, a2, a3)`:

```
x %>% f()        equivalent to  f(a1 = x)

x %>% f(3, 5)    equivalent to  f(a1 = x, a2 = 3, a3 = 5)
```

Online material 1: pipes

How is this useful? Electricity demand example:

```
data(UKload)

plot(NetDemand ~ Posan,
     transform(head(subset(UKload, Dow == "lundi",
     select = c("NetDemand", "Posan")), 100),
     Posan = Posan * 365))
```

The piped equivalent:

```
UKload %>%
  subset(Dow == "lundi",
         select = c("NetDemand", "Posan")) %>%
  head(100) %>%
  transform(Posan = Posan * 365) %>%
  plot(NetDemand ~ Posan, data = .)
```

Online material 1: pipes

The advantages are:

- aesthetic (subjective)
- improved clarity $\rightarrow$ fewer errors
- compatibility with rest of Tidyverse

The material covers also other pipes: %<>%, %$%, ...

Note that there is a danger of going to far, e.g.:

```
sqrt( abs( cos(x) ) )
```

is closer to $\sqrt{|\cos(x)|}$ than

```
x %>% cos %>% abs %>% sqrt
```

Online material 2: ggplot2

We briefly introduce ggplot2:

1. how to build basic plots
2. how to add layers and facets

A scatterplot in base R:

```
data(mcycle)

plot(x = mcycle$times, y = mcycle$accel)
```

Function plot is called for its side effects:

```
tmp <- plot(x = mcycle$times, y = mcycle$accel)

tmp

## NULL
```

Online material 2: ggplot2

A scatterplot in ggplot2:

```
# Building object
pl <- ggplot(data = mcycle)

# Adding layers
pl <- pl + geom_point(mapping = aes(x=times, y=accel))

# Rendering on screen
pl
```

Basic template:

```
ggplot(data = <data.frame>) +
  <geom_layer>(mapping = aes(<variables_map>))
```

Online material 3: ggplot2 case study

Ok, `ggplot2` plots are pretty, but what else?

`mgcViz` case study provides some motivation.

Context is GAM modelling with `mgcv`:

```
fitG <- gam(Demand ~ Dow + s(Posan) + s(wM) + s(Trend),
            data = UKload)
```

We plot the effect of `Posan` by doing:

```
plot(fitG, select = 1)
```

This will call `plot.gam`.

Online material 3: ggplot2 case study

`plot.gam` does its job but:

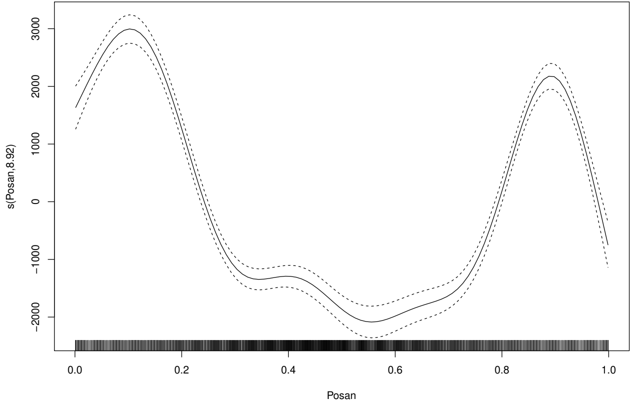
```
args(plot.gam)

function (x, residuals = FALSE, rug = NULL, se = TRUE,
          pages = 0, select = NULL, scale = -1, n = 100,
          n2 = 40, n3 = 3, pers = FALSE, theta = 30,
          phi = 30, jit = FALSE, xlab = NULL, ylab = NULL,
          main = NULL, ylim = NULL, xlim = NULL,
          too.far = 0.1, all.terms = FALSE, shade = FALSE,
          shade.col = "gray80", shift = 0, trans = I,
          seWithMean = FALSE, unconditional = FALSE,
          by.resids = FALSE, scheme = 0, ...)
```

Quite a lot of arguments and...

Online material 3: ggplot2 case study

Quite a lot of arguments and:

- difficult to add new features
- cannot control properties of elements
- order in which elements are rendered is fixed



Online material 3: ggplot2 case study

Quite a lot of arguments and:

- difficult to add new features
- cannot control properties of elements
- order in which elements are rendered is fixed

`mgcViz` wraps GAM object allows us to do:

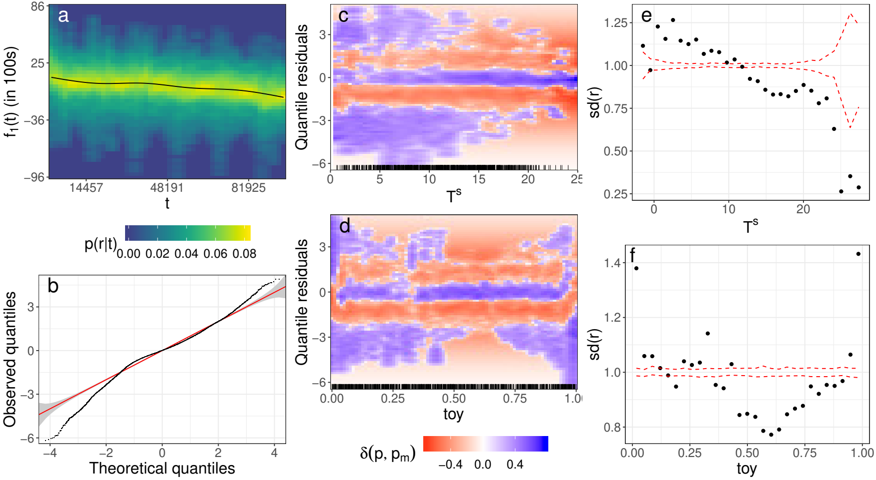
```
plot(fitG, 1) + l_fitLine(col = "red") +
                l_ciLine(linetype = 2) +
                xlim(0.25, 0.75)
```

where, e.g., `l_fitLine` is a wrapper for `geom_line`.

Amazingly, this solves all the problems above.

Online material 3: ggplot2 case study

Personal conclusion is that, if you want to build an flexible and extensible graphical library in R, ggplot2 might be the way to go.



Online material 4: dplyr

`ggplot2` wants you to provide a `data.frame`:

```
ggplot(data = <data.frame>) +
  <geom_layer>(mapping = aes(<variables_map>))
```

This is annoying when you just want `plot(x, y)`, `hist(x)`...

For more complex plots, effort is justified.

Many modelling functions (e.g. `lm`, `glm`, `gam`) have same requirement.

`dplyr` and `tidyr` help us building the "right" `data.frame` for visualization and modelling.

Online material 4: dplyr

The online notes focus on the basics. Why? Well...

```
length( getNamespaceExports("dplyr") ) # Oct 2022
287
```

UK demand example:

```
head(UKload)
    NetDemand   wM wM_s95 Posan       Dow      Trend
25      38353 6.05   5.56  0.00    samedi 1293879600
73      41192 2.80   3.23  0.00 dimanche 1293966000
121     43442 2.10   1.86  0.01     lundi 1294052400
```

Online material 4: dplyr

Example of `dplyr` code:

```
UKload %>% select(NetDemand, wM, Dow, Posan) %>%
           filter(wM < 5 & Dow == "lundi") %>%
           arrange(desc(wM)) %>%
           ggplot() +
           geom_point(aes(x=wM, NetDemand))
```

Base `R` equivalent:

```
d0 <- UKload[ , c("NetDemand", "wM", "Dow", "Posan")]
d0 <- d0[d0$wM < 5 & d0$Dow == "lundi", ]
d0 <- d0[rev(order(d0$wM)), ]
plot(NetDemand ~ wM, d0)
```

Online material 4: dplyr

A more interesting example:

```
d0 <- UKload %>% mutate(wk = week(Date)) %>%
                 group_by(Year, wk) %>%
                 summarise(TotDemand = sum(NetDemand),
                           tempMax = max(wM),
                           tempMin = min(wM),
                           nHoly = sum(Holy=="1"))
```

Base R equivalent:

```
d0 <- UKload
d0$wk <- week(UKload$Date)
???
```

There surely is a good base R solution but dplyr code generally clearer and more concise.

Online material 5: data reshaping with tidyr and dplyr

We consider Irish electricity smart meter data:

```
data(Irish)
indCons <- Irish$indCons
head(indCons)
##      I1002 I1003 I1004 I1005 I1013 I1015 I1018 ...
## 8114 0.022 0.593 2.002 0.755 0.035 0.398 0.547 ...
## 8115 0.133 0.707 1.602 0.898 0.112 0.689 0.603 ...
## 8116 0.094 0.684 1.525 0.736 0.046 0.407 0.511 ...
```

But we need data in “long” format for ggplotting and modelling:

```
##      ID   dem
## 1 I1002 0.022
## 2 I1002 0.133
## 3 I1002 0.094
## 4 I1002 0.023
```

Online material 5: data reshaping with tidyr and dplyr

Easily done with `tidyr::pivot_longer`:

```
longDat <- indCons %>% pivot_longer(cols = everything(),
 names_to = "ID", values_to = "dem") %>% arrange(ID)
head(longDat)
##       ID   dem
## 1 I1002 0.022
## 2 I1002 0.133
## 3 I1002 0.094
```

But there is a memory price to pay:

```
indCons %>% object.size %>% format(units = "MB")
# "12.9 Mb"

longDat %>% object.size %>% format(units = "MB")
# "25.6 Mb"
```

Online material 5: data reshaping with tidyr and dplyr

Opposite transformation achieved by `tidyr::pivot_wider`:

```
wideDat <- longDat %>% pivot_wider(names_from = "ID",
                                   values_from = "dem")
head(wideDat)

 # A tibble: 16,799 x 101
   I1002 I1003 I1004 I1005 I1013 I1015 I1018
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1 0.022 0.593  2.00 0.755 0.035 0.398 0.547
 2 0.133 0.707  1.60 0.898 0.112 0.689 0.603
 3 0.094 0.684  1.52 0.736 0.046 0.407 0.511
 4 0.023 0.563  1.39 0.738 0.036 0.223 0.593
 5 0.133 0.489  1.22 0.849 0.065 0.132 0.570
 #  with 1.679e+04 more rows, and 94 more variables
```

Another common task is merging data frames, e.g.:

```
extra <- as_tibble( Irish$extra )
head(extra)
## # A tibble: 6 x 6
##    time   toy dow   holy    tod  temp
##   <int> <dbl> <fct> <lgl> <dbl> <dbl>
## 1     1 0.986 Wed   FALSE     0     4
## 2     2 0.986 Wed   FALSE     1     4
## 3     3 0.986 Wed   FALSE     2     4
## 4     4 0.986 Wed   FALSE     3     4
## 5     5 0.986 Wed   FALSE     4     4
## 6     6 0.986 Wed   FALSE     5     4
```

Online material 5: data reshaping with tidyr and dplyr

Here merging is easy:

```
allDat <- cbind(longDat, extra)
head(allDat)
##      ID   dem  time       toy dow  holy tod temp
## 1 I1002 0.022     1 0.9863014 Wed FALSE   0    4
## 2 I1002 0.133     2 0.9863014 Wed FALSE   1    4
## 3 I1002 0.094     3 0.9863014 Wed FALSE   2    4
## 4 I1002 0.023     4 0.9863014 Wed FALSE   3    4
## 5 I1002 0.133     5 0.9863014 Wed FALSE   4    4
## 6 I1002 0.090     6 0.9863014 Wed FALSE   5    4
```

But remember about memory costs ...

Online material 5: data reshaping with tidyr and dplyr

But how to add also the customer survey data:

```
survey <- as_tibble( Irish$survey )
head(survey)

# A tibble: 6 x 12
  ID     meanDem SCLASS OWNERSHIP YEAR HEAT.HOME HEAT.WAT
  <chr>    <dbl> <fct>  <fct>    <dbl> <fct>     <fct>
1 I1002    0.208 DE     O         1975 Other     Elec
2 I1003    0.622 C1     O         2004 Other     Other
3 I1004    0.962 C1     O         1987 Other     Elec
4 I1005    0.640 C1     O         1930 Other     Other
5 I1013    0.241 C2     O         2003 Other     Elec
6 I1015    0.463 DE     R         1989 Elec      Other
#  with 5 more variables: WINDOWS.doubleglazed <fct>,
#   HOME.APPLIANCE..White.goods. <dbl>, Code <int>,
```

Online material 5: data reshaping with tidyr and dplyr

Solution is offered by `dplyr::left_join`:

```
allDat <- allDat %>% left_join(survey, by = "ID") %>%
                     as_tibble()

## # A tibble: 6 x 20
##   ID      dem    time   toy dow   holy    tod   SCLASS
##   <chr> <dbl>   <int> <dbl> <fct> <lgl> <dbl>  <FCT>
## 1 I1002 0.022       1 0.986 Wed   FALSE     0   DE
## 2 I1002 0.133       2 0.986 Wed   FALSE     1   DE
## 3 I1002 0.094       3 0.986 Wed   FALSE     2   DE
## 4 I1002 0.023       4 0.986 Wed   FALSE     3   DE
## 5 I1002 0.133       5 0.986 Wed   FALSE     4   DE
## 6 I1002 0.09        6 0.986 Wed   FALSE     5   DE
## #  with 10 more variables: OWNERSHIP <fct>,
## #   BUILT.YEAR <dbl>, HEAT.HOME <fct>, HEAT.WAT <fct>,
## #   WINDOWS.doubleglazed <fct>
```

Online material 5: data reshaping with tidyr and dplyr

Now we have all the info in one `data.frame` but:

```
indCons %>% object.size() %>% format("MB")
## [1] "12.9 Mb"

survey %>% object.size() %>% format("MB")
## [1] "0.3 Mb"

extra %>% object.size() %>% format("MB")
## [1] "0.6 Mb"
```

```
allDat %>% object.size() %>% format("MB")
## [1] "217.9 Mb"
```

Further topics

Online material covers basics, for details see "R for Data Science" book.

Available online at `https://r4ds.had.co.nz/`.

For a skeptical point of view on the Tidyverse, see:

`https://github.com/matloff/TidyverseSkeptic`

- Tidyverse "bad" when teaching to ppl with no background
- Tidyverse averse to `$`, `[[ ]]`, loops and `plot()`
- Tidyverse is advertised by Rstudio (data.table example)
- too many functions: mutate, mutate_, mutate_all, mutate_at, mutate_each, mutate_each_, mutate_if, transmute, transmute_, transmute_all, transmute_at

Computer lab

Probably the most useful thing to do for you is to experiment the Tidyverse packages on real data.

The data sets used in the online notes are in `qgam` and `electBook` packages.

An ideal data set is also provided in this Kaggle challenge:

`https://www.kaggle.com/c/ashrae-energy-prediction`

Computer lab

Consumption data:

```
head(train)
  building_id meter           timestamp meter_reading
1           0     0 2016-01-01 00:00:00             0
2           1     0 2016-01-01 00:00:00             0
3           2     0 2016-01-01 00:00:00             0
4           3     0 2016-01-01 00:00:00             0
```

Building information:

```
head(buildMeta)
  site_id building_id primary_use square_feet year_built
1       0           0   Education        7432       2008
2       0           1   Education        2720       2004
3       0           2   Education        5376       1991
```

Computer lab

Weather data:

```
head(weather)
  site_id air_temperature cloud_coverage wind_speed
1       0            25.0              6        0.0
2       0            24.4             NA        1.5
3       0            22.8              2        0.0
```

The data could get big:

```
train %>% object.size %>% format("MB")
"386.4 Mb"
weather %>% object.size %>% format("MB")
"9.3 Mb"

nlevels(train$building_id)
1449
```

References I

Hastie, T. and R. Tibshirani (1990). *Generalized Additive Models*, Volume 43. CRC Press.