I’m always trying not to be tied into that *verse* too much by replicating certain tasks with other tools (and languages) as an exercise.

Code Chunks – Tidyverse and Data.Table – Sitting Side By Side

## Getting the polls

**library**(readr)

polls\_2016 <- **read\_tsv**(**url**("http://elections.huffingtonpost.com/pollster/api/v2/questions/16-US-Pres-GE%20TrumpvClinton/poll-responses-clean.tsv"))

## Wrangling the polls

**library**(dplyr)

polls\_2016 <- polls\_2016 **%>%**

**filter**(sample\_subpopulation **%in%** **c**("Adults","Likely Voters","Registered Voters"))

**library**(lubridate)

polls\_2016 <- polls\_2016 **%>%**

**mutate**(end\_date = **ymd**(end\_date))

polls\_2016 <- polls\_2016 **%>%**

**right\_join**(**data.frame**(end\_date = **seq.Date**(**min**(polls\_2016**$**end\_date),

**max**(polls\_2016**$**end\_date), by="days")))

## Average the polls

polls\_2016 <- polls\_2016 **%>%**

**group\_by**(end\_date) **%>%**

**summarise**(Clinton = **mean**(Clinton),

Trump = **mean**(Trump))

**library**(zoo)

rolling\_average <- polls\_2016 **%>%**

**mutate**(Clinton.Margin = Clinton**-**Trump,

Clinton.Avg = **rollapply**(Clinton.Margin,width=14,

FUN=**function**(x){**mean**(x, na.rm=TRUE)},

by=1, partial=TRUE, fill=NA, align="right"))

**library**(ggplot2)

**ggplot**(rolling\_average)**+**

**geom\_line**(**aes**(x=end\_date,y=Clinton.Avg),col="blue") **+**

**geom\_point**(**aes**(x=end\_date,y=Clinton.Margin))

It uses five packages to i) read some data off them interwebs, ii) then filters / subsets / modifies it leading to a right (outer) join with itself before iv) averaging per-day polls first and then creates rolling averages over 14 days before v) plotting. Several standard *verbs* are used: filter(), mutate(), right\_join(), group\_by(), and summarise(). One non-verse function is rollapply() which comes from [zoo](https://cran.r-project.org/web/packages/zoo/index.html), a popular package for time-series data.

**Complete Code using Approach "DT"**

As I will show below, we can do the same with fewer packages as data.table covers the reading, slicing/dicing and time conversion. We still need zoo for its rollapply() and of course the same plotting code:

## Getting the polls

**library**(data.table)

pollsDT <- **fread**("http://elections.huffingtonpost.com/pollster/api/v2/questions/16-US-Pres-GE%20TrumpvClinton/poll-responses-clean.tsv")

## Wrangling the polls

pollsDT <- pollsDT[sample\_subpopulation **%in%** **c**("Adults","Likely Voters","Registered Voters"), ]

pollsDT[, end\_date **:=** **as.IDate**(end\_date)]

pollsDT <- pollsDT[ **data.table**(end\_date = **seq**(**min**(pollsDT[,end\_date]),

**max**(pollsDT[,end\_date]), by="days")), on="end\_date"]

## Average the polls

**library**(zoo)

pollsDT <- pollsDT[, .(Clinton=**mean**(Clinton), Trump=**mean**(Trump)), by=end\_date]

pollsDT[, Clinton.Margin **:=** Clinton**-**Trump]

pollsDT[, Clinton.Avg **:=** **rollapply**(Clinton.Margin, width=14,

FUN=**function**(x){**mean**(x, na.rm=TRUE)},

by=1, partial=TRUE, fill=NA, align="right")]

**library**(ggplot2)

**ggplot**(pollsDT) **+**

**geom\_line**(**aes**(x=end\_date,y=Clinton.Avg),col="blue") **+**

**geom\_point**(**aes**(x=end\_date,y=Clinton.Margin))

This uses several of the components of data.table which are often called [i, j, by=...]. Row are selected (i), columns are either modified (via := assignment) or summarised (via =), and grouping is undertaken by by=.... The outer join is done by having a data.table object indexed by another, and is pretty standard too. That allows us to do all transformations in three lines. We then create per-day average by grouping by day, compute the margin and construct its rolling average as before. The resulting chart is, unsurprisingly, the same.

**Benchmark Reading**

We can looking how the two approaches do on getting data read into our session. For simplicity, we will read a local file to keep the (fixed) download aspect out of it:

R**>** url <- "http://elections.huffingtonpost.com/pollster/api/v2/questions/16-US-Pres-GE%20TrumpvClinton/poll-responses-clean.tsv"

R**>** **download.file**(url, destfile=file, quiet=TRUE)

R**>** file <- "/tmp/poll-responses-clean.tsv"

R**>** res <- **microbenchmark**(tidy=**suppressMessages**(readr**::read\_tsv**(file)),

**+** dt=data.table**::fread**(file, showProgress=FALSE))

R**>** res

Unit**:** milliseconds

expr min lq mean median uq max neval

tidy 6.67777 6.83458 7.13434 6.98484 7.25831 9.27452 100

dt 1.98890 2.04457 2.37916 2.08261 2.14040 28.86885 100

R**>**

That is a clear relative difference, though the absolute amount of time is not that relevant for such a small (demo) dataset.

**Benchmark Processing**

We can also look at the processing part:

R**>** rdin <- **suppressMessages**(readr**::read\_tsv**(file))

R**>** dtin <- data.table**::fread**(file, showProgress=FALSE)

R**>**

R**>** **library**(dplyr)

R**>** **library**(lubridate)

R**>** **library**(zoo)

R**>**

R**>** transformTV <- **function**(polls\_2016=rdin) {

**+** polls\_2016 <- polls\_2016 **%>%**

**+** **filter**(sample\_subpopulation **%in%** **c**("Adults","Likely Voters","Registered Voters"))

**+** polls\_2016 <- polls\_2016 **%>%**

**+** **mutate**(end\_date = **ymd**(end\_date))

**+** polls\_2016 <- polls\_2016 **%>%**

**+** **right\_join**(**data.frame**(end\_date = **seq.Date**(**min**(polls\_2016**$**end\_date),

**+** **max**(polls\_2016**$**end\_date), by="days")))

**+** polls\_2016 <- polls\_2016 **%>%**

**+** **group\_by**(end\_date) **%>%**

**+** **summarise**(Clinton = **mean**(Clinton),

**+** Trump = **mean**(Trump))

**+**

**+** rolling\_average <- polls\_2016 **%>%**

**+** **mutate**(Clinton.Margin = Clinton**-**Trump,

**+** Clinton.Avg = **rollapply**(Clinton.Margin,width=14,

**+** FUN=**function**(x){**mean**(x, na.rm=TRUE)},

**+** by=1, partial=TRUE, fill=NA, align="right"))

**+** }

R**>**

R**>** transformDT <- **function**(dtin) {

**+** pollsDT <- **copy**(dtin) ## extra work to protect from reference semantics for benchmark

**+** pollsDT <- pollsDT[sample\_subpopulation **%in%** **c**("Adults","Likely Voters","Registered Voters"), ]

**+** pollsDT[, end\_date **:=** **as.IDate**(end\_date)]

**+** pollsDT <- pollsDT[ **data.table**(end\_date = **seq**(**min**(pollsDT[,end\_date]),

**+** **max**(pollsDT[,end\_date]), by="days")), on="end\_date"]

**+** pollsDT <- pollsDT[, .(Clinton=**mean**(Clinton), Trump=**mean**(Trump)),

**+** by=end\_date][, Clinton.Margin **:=** Clinton**-**Trump]

**+** pollsDT[, Clinton.Avg **:=** **rollapply**(Clinton.Margin, width=14,

**+** FUN=**function**(x){**mean**(x, na.rm=TRUE)},

**+** by=1, partial=TRUE, fill=NA, align="right")]

**+** }

R**>**

R**>** res <- **microbenchmark**(tidy=**suppressMessages**(**transformTV**(rdin)),

**+** dt=**transformDT**(dtin))

R**>** res

Unit**:** milliseconds

expr min lq mean median uq max neval

tidy 12.54723 13.18643 15.29676 13.73418 14.71008 104.5754 100

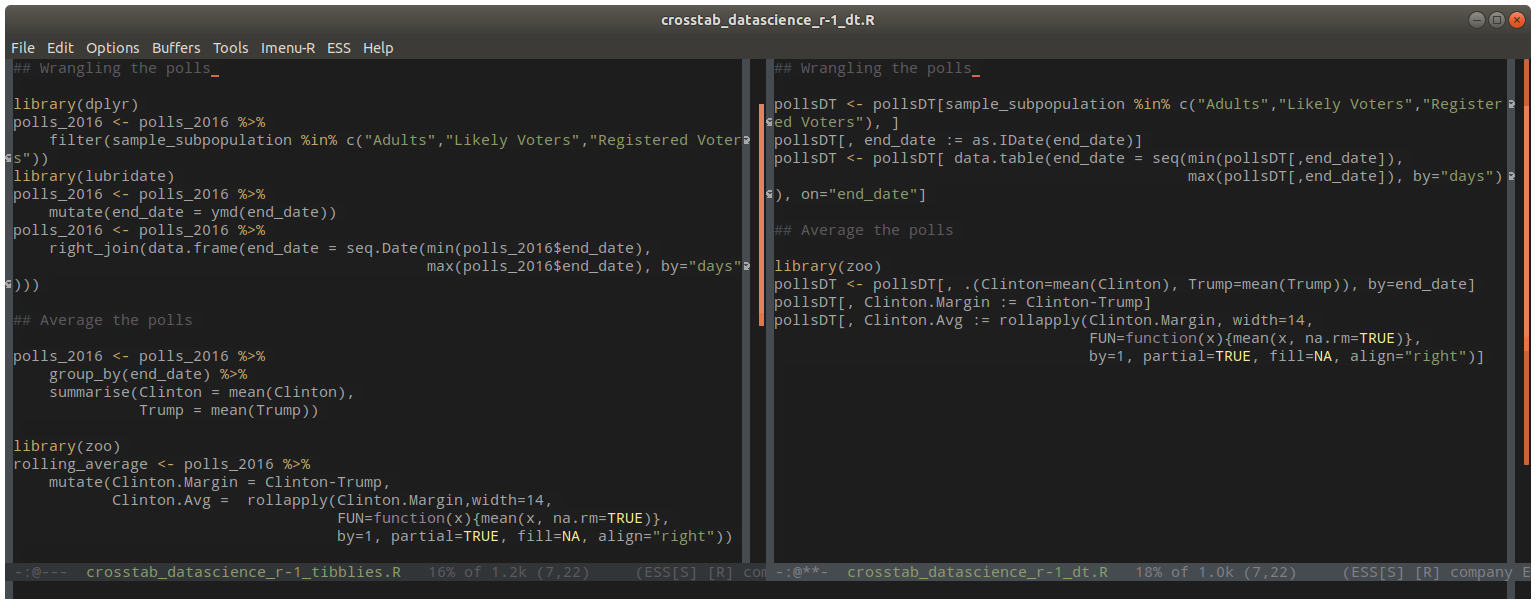
dt 7.66842 8.02404 8.60915 8.29984 8.72071 17.7818 100

R**>**

Not quite a factor of two on the small data set, but again a clear advantage. data.table has a reputation for doing really well for large datasets; here we see that it is also faster for small datasets.

**Side-by-side**

Stripping the reading, as well as the plotting both of which are about the same, we can compare the essential data operations.



First of all, I would like to clean up the tidyverse version a little, because the original was distributed in chunks and was a little bit too verbose. We can also avoid using lubridate, because readr already parses the end\_date column as a date (and that’s why it is significantly slower, among other reasons). This is how I would do it:

## Getting the polls

library(tidyverse)

library(zoo)

polls\_2016 <- read\_tsv(url("http://elections.huffingtonpost.com/pollster/api/v2/questions/16-US-Pres-GE%20TrumpvClinton/poll-responses-clean.tsv"))

## Wrangling the polls

polls\_2016 <- polls\_2016 %>%

filter(sample\_subpopulation %in% c("Adults","Likely Voters","Registered Voters")) %>%

right\_join(data.frame(end\_date = seq.Date(min(.$end\_date), max(.$end\_date), by="days")), by="end\_date")

## Average the polls

rolling\_average <- polls\_2016 %>%

group\_by(end\_date) %>%

summarise(Clinton = mean(Clinton), Trump = mean(Trump)) %>%

mutate(Clinton.Margin = Clinton-Trump,

Clinton.Avg = rollapply(Clinton.Margin,width=14,

FUN=function(x){mean(x, na.rm=TRUE)},

by=1, partial=TRUE, fill=NA, align="right"))

ggplot(rolling\_average) +

geom\_line(aes(x=end\_date, y=Clinton.Avg), col="blue") +

geom\_point(aes(x=end\_date, y=Clinton.Margin))

which, by the way, has exactly the very same number of lines of code than the data.table version:

## Getting the polls

library(data.table)

library(zoo)

library(ggplot2)

pollsDT <- fread("http://elections.huffingtonpost.com/pollster/api/v2/questions/16-US-Pres-GE%20TrumpvClinton/poll-responses-clean.tsv")

## Wrangling the polls

pollsDT <- pollsDT[sample\_subpopulation %in% c("Adults","Likely Voters","Registered Voters"), ]

pollsDT[, end\_date := as.IDate(end\_date)]

pollsDT <- pollsDT[ data.table(end\_date = seq(min(pollsDT[,end\_date]),

max(pollsDT[,end\_date]), by="days")), on="end\_date"]

## Average the polls

pollsDT <- pollsDT[, .(Clinton=mean(Clinton), Trump=mean(Trump)), by=end\_date]

pollsDT[, Clinton.Margin := Clinton-Trump]

pollsDT[, Clinton.Avg := rollapply(Clinton.Margin, width=14,

FUN=function(x){mean(x, na.rm=TRUE)},

by=1, partial=TRUE, fill=NA, align="right")]

ggplot(pollsDT) +

geom\_line(aes(x=end\_date, y=Clinton.Avg), col="blue") +

geom\_point(aes(x=end\_date, y=Clinton.Margin))

Let’s translate this into base R. It is easier to start from the data.table version, mainly because filtering and assigning have a similar look and feel. Unsurprisingly, we have base::merge for the merge operation and stats::aggregate for the aggregation phase. base::as.Date works just fine for these dates and utils::read.csv has the only drawback that you have to specify the separator. Without further ado, this is my version in base R:

## Getting the polls

library(zoo)

pollsB <- read.csv(url("http://elections.huffingtonpost.com/pollster/api/v2/questions/16-US-Pres-GE%20TrumpvClinton/poll-responses-clean.tsv"), sep="\t")

## Wrangling the polls

pollsB <- pollsB[pollsB$sample\_subpopulation %in% c("Adults","Likely Voters","Registered Voters"), ]

pollsB$end\_date <- base::as.Date(pollsB$end\_date)

endDate <- data.frame(end\_date = seq.Date(min(pollsB$end\_date), max(pollsB$end\_date), by="days"))

pollsB <- merge(pollsB, endDate, by="end\_date", all=TRUE)

## Average the polls

pollsB <- aggregate(cbind(Clinton, Trump) ~ end\_date, data=pollsB, mean, na.action=na.pass)

pollsB$Clinton.Margin <- pollsB$Clinton - pollsB$Trump

pollsB$Clinton.Avg <- rollapply(pollsB$Clinton.Margin, width=14,

FUN=function(x){mean(x, na.rm=TRUE)},

by=1, partial=TRUE, fill=NA, align="right")

plot(pollsB$end\_date, pollsB$Clinton.Margin, pch=16)

lines(pollsB$end\_date, pollsB$Clinton.Avg, col="blue", lwd=2)

which is the shortest one! Finally, let’s repeat the benchmark too:

library(microbenchmark)

url <- "http://elections.huffingtonpost.com/pollster/api/v2/questions/16-US-Pres-GE%20TrumpvClinton/poll-responses-clean.tsv"

file <- "/tmp/poll-responses-clean.tsv"

download.file(url, destfile=file, quiet=TRUE)

res <- microbenchmark(tidy=suppressMessages(readr::read\_tsv(file)),

dt=data.table::fread(file, showProgress=FALSE),

base=read.csv(file, sep="\t"))

res

## Unit: milliseconds

## expr min lq mean median uq max neval

## tidy 13.877036 15.127885 18.549393 15.861311 17.813541 202.389391 100

## dt 4.084022 4.505943 5.152799 4.845193 5.652579 7.736563 100

## base 29.029366 30.437742 32.518009 31.449916 33.600937 45.104599 100

Base R is clearly the slowest option for the reading phase. Or, one might say, both readr and data.table have done a great job in improving things! Let’s take a look at the processing part now:

tvin <- suppressMessages(readr::read\_tsv(file))

dtin <- data.table::fread(file, showProgress=FALSE)

bsin <- read.csv(file, sep="\t")

library(tidyverse)

library(data.table)

library(zoo)

transformTV <- function(polls\_2016) {

polls\_2016 <- polls\_2016 %>%

filter(sample\_subpopulation %in% c("Adults","Likely Voters","Registered Voters")) %>%

right\_join(data.frame(end\_date = seq.Date(min(.$end\_date), max(.$end\_date), by="days")), by="end\_date")

rolling\_average <- polls\_2016 %>%

group\_by(end\_date) %>%

summarise(Clinton = mean(Clinton), Trump = mean(Trump)) %>%

mutate(Clinton.Margin = Clinton-Trump,

Clinton.Avg = rollapply(Clinton.Margin,width=14,

FUN=function(x){mean(x, na.rm=TRUE)},

by=1, partial=TRUE, fill=NA, align="right"))

}

transformDT <- function(dtin) {

pollsDT <- copy(dtin) ## extra work to protect from reference semantics for benchmark

pollsDT <- pollsDT[sample\_subpopulation %in% c("Adults","Likely Voters","Registered Voters"), ]

pollsDT[, end\_date := as.IDate(end\_date)]

pollsDT <- pollsDT[ data.table(end\_date = seq(min(pollsDT[,end\_date]),

max(pollsDT[,end\_date]), by="days")), on="end\_date"]

pollsDT <- pollsDT[, .(Clinton=mean(Clinton), Trump=mean(Trump)), by=end\_date]

pollsDT[, Clinton.Margin := Clinton-Trump]

pollsDT[, Clinton.Avg := rollapply(Clinton.Margin, width=14,

FUN=function(x){mean(x, na.rm=TRUE)},

by=1, partial=TRUE, fill=NA, align="right")]

}

transformBS <- function(pollsB) {

pollsB <- pollsB[pollsB$sample\_subpopulation %in% c("Adults","Likely Voters","Registered Voters"), ]

pollsB$end\_date <- base::as.Date(pollsB$end\_date)

endDate <- data.frame(end\_date = seq.Date(min(pollsB$end\_date), max(pollsB$end\_date), by="days"))

pollsB <- merge(pollsB, endDate, by="end\_date", all=TRUE)

pollsB <- aggregate(cbind(Clinton, Trump) ~ end\_date, data=pollsB, mean, na.action=na.pass)

pollsB$Clinton.Margin <- pollsB$Clinton - pollsB$Trump

pollsB$Clinton.Avg <- rollapply(pollsB$Clinton.Margin, width=14,

FUN=function(x){mean(x, na.rm=TRUE)},

by=1, partial=TRUE, fill=NA, align="right")

}

res <- microbenchmark(tidy=transformTV(tvin),

dt=transformDT(dtin),

base=transformBS(bsin))

res

## Unit: milliseconds

## expr min lq mean median uq max neval

## tidy 20.68435 22.58603 26.67459 24.56170 27.85844 84.55077 100

## dt 17.25547 18.88340 21.43256 20.24450 22.26448 41.65252 100

## base 28.39796 30.93722 34.94262 32.97987 34.98222 109.14005 100