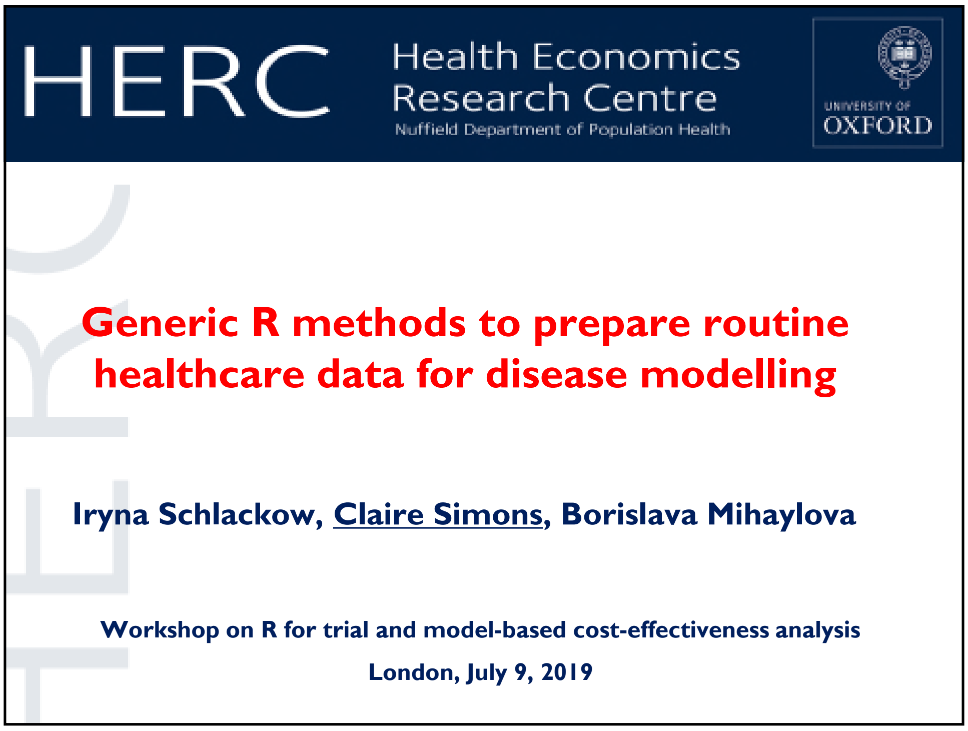


HERC Health Economics Research Centre
Nuffield Department of Population Health

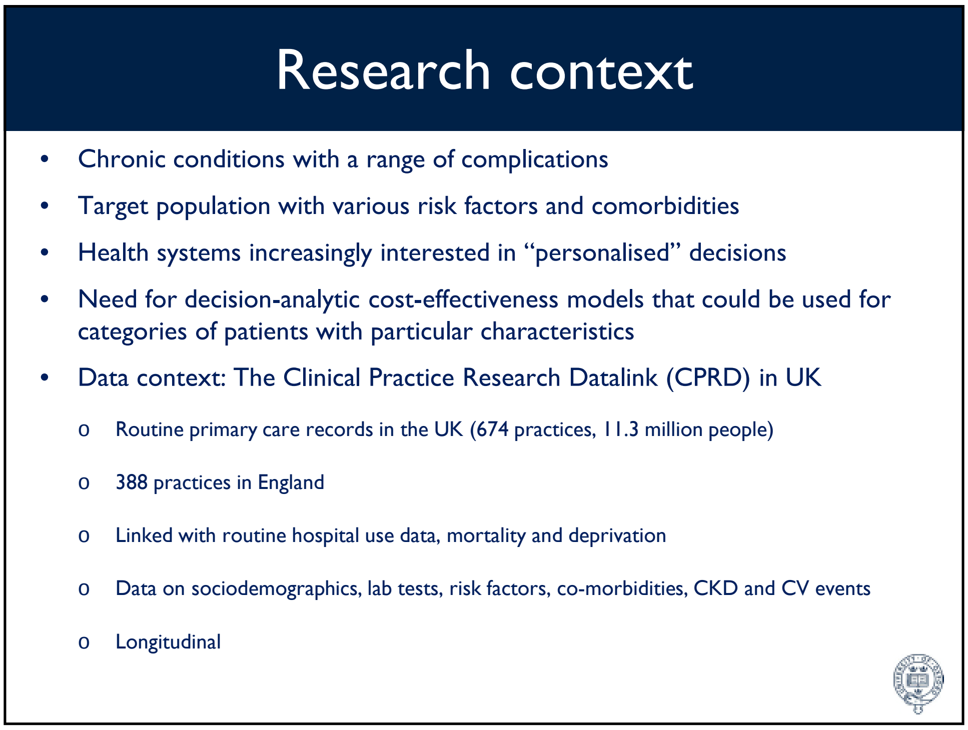
UNIVERSITY OF OXFORD

Generic R methods to prepare routine healthcare data for disease modelling

Iryna Schlackow, Claire Simons, Borislava Mihaylova

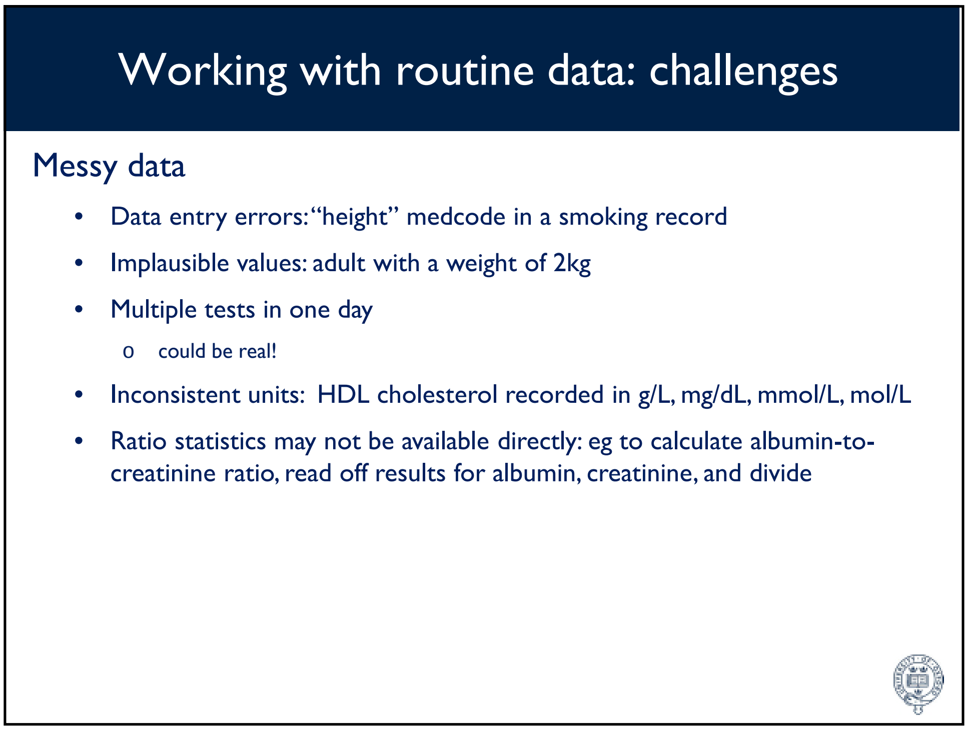
Workshop on R for trial and model-based cost-effectiveness analysis

London, July 9, 2019



Research context

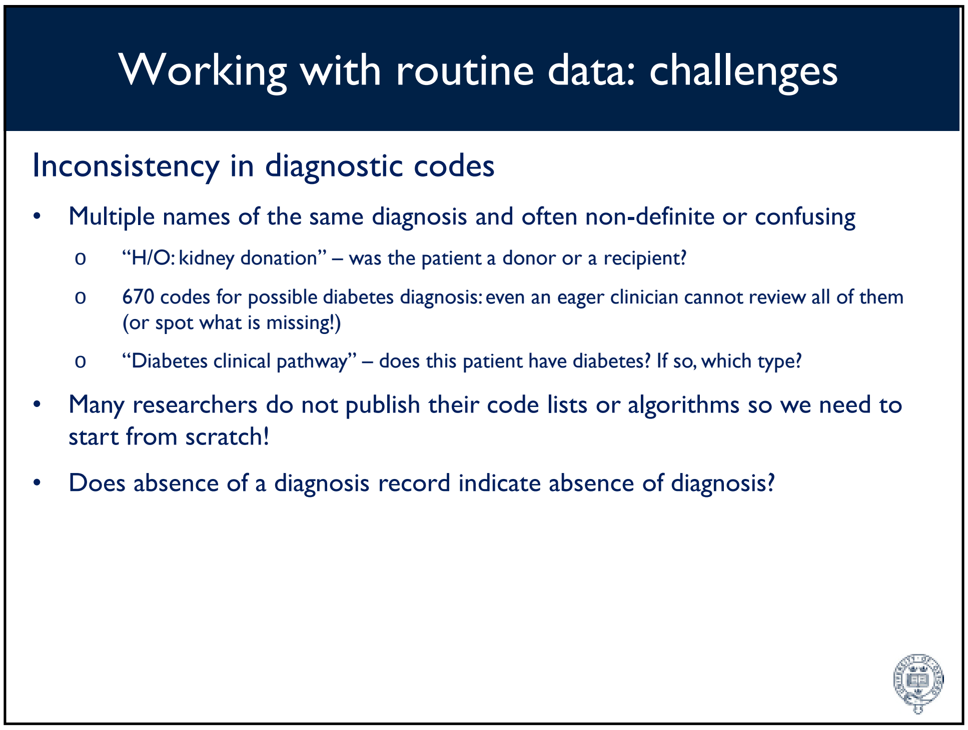
- Chronic conditions with a range of complications
- Target population with various risk factors and comorbidities
- Health systems increasingly interested in "personalised" decisions
- Need for decision-analytic cost-effectiveness models that could be used for categories of patients with particular characteristics
- Data context: The Clinical Practice Research Datalink (CPRD) in UK
    - Routine primary care records in the UK (674 practices, 11.3 million people)
    - 388 practices in England
    - Linked with routine hospital use data, mortality and deprivation
    - Data on sociodemographics, lab tests, risk factors, co-morbidities, CKD and CV events
    - Longitudinal



Working with routine data: challenges

Messy data

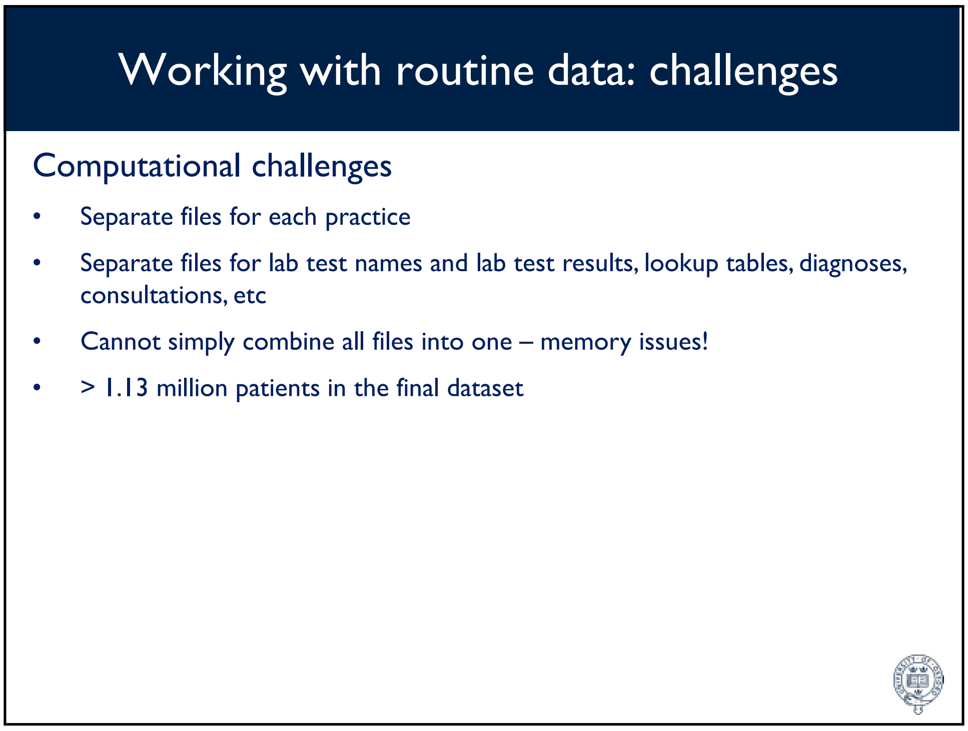
- Data entry errors: "height" medcode in a smoking record
- Implausible values: adult with a weight of 2kg
- Multiple tests in one day
  - could be real!
- Inconsistent units: HDL cholesterol recorded in g/L, mg/dL, mmol/L, mol/L
- Ratio statistics may not be available directly: eg to calculate albumin-to-creatinine ratio, read off results for albumin, creatinine, and divide



Working with routine data: challenges

Inconsistency in diagnostic codes

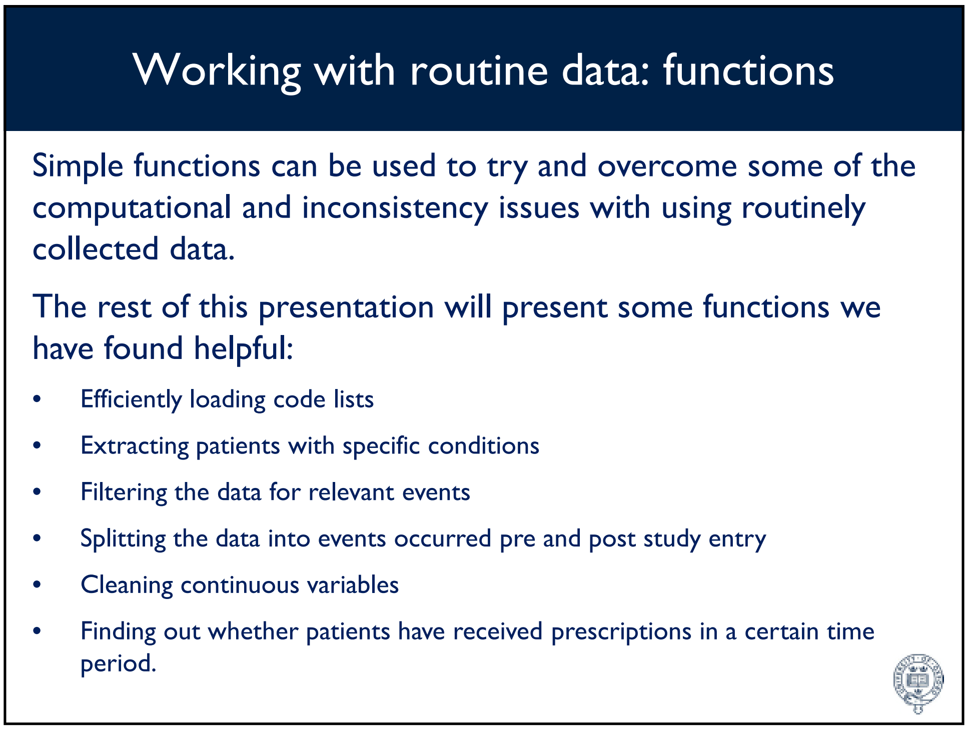
- Multiple names of the same diagnosis and often non-definite or confusing
  - "H/O: kidney donation" – was the patient a donor or a recipient?
  - 670 codes for possible diabetes diagnosis: even an eager clinician cannot review all of them (or spot what is missing!)
  - "Diabetes clinical pathway" – does this patient have diabetes? If so, which type?
- Many researchers do not publish their code lists or algorithms so we need to start from scratch!
- Does absence of a diagnosis record indicate absence of diagnosis?



Working with routine data: challenges

Computational challenges

- Separate files for each practice
- Separate files for lab test names and lab test results, lookup tables, diagnoses, consultations, etc
- Cannot simply combine all files into one – memory issues!
- > 1.13 million patients in the final dataset

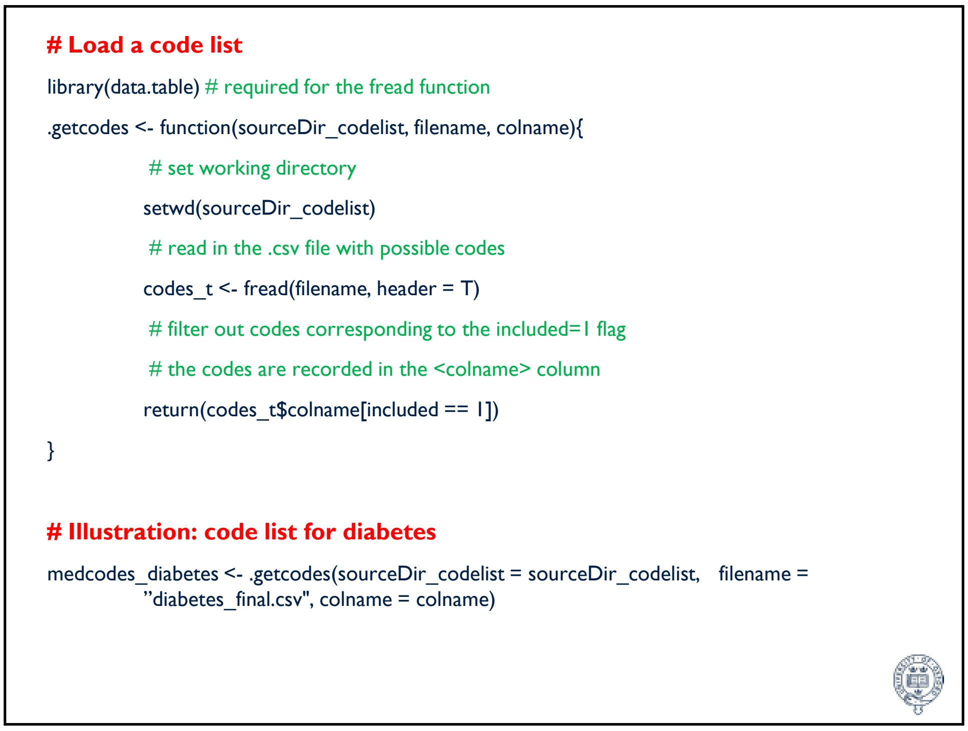


Working with routine data: functions

Simple functions can be used to try and overcome some of the computational and inconsistency issues with using routinely collected data.

The rest of this presentation will present some functions we have found helpful:

- Efficiently loading code lists
- Extracting patients with specific conditions
- Filtering the data for relevant events
- Splitting the data into events occurred pre and post study entry
- Cleaning continuous variables
- Finding out whether patients have received prescriptions in a certain time period.

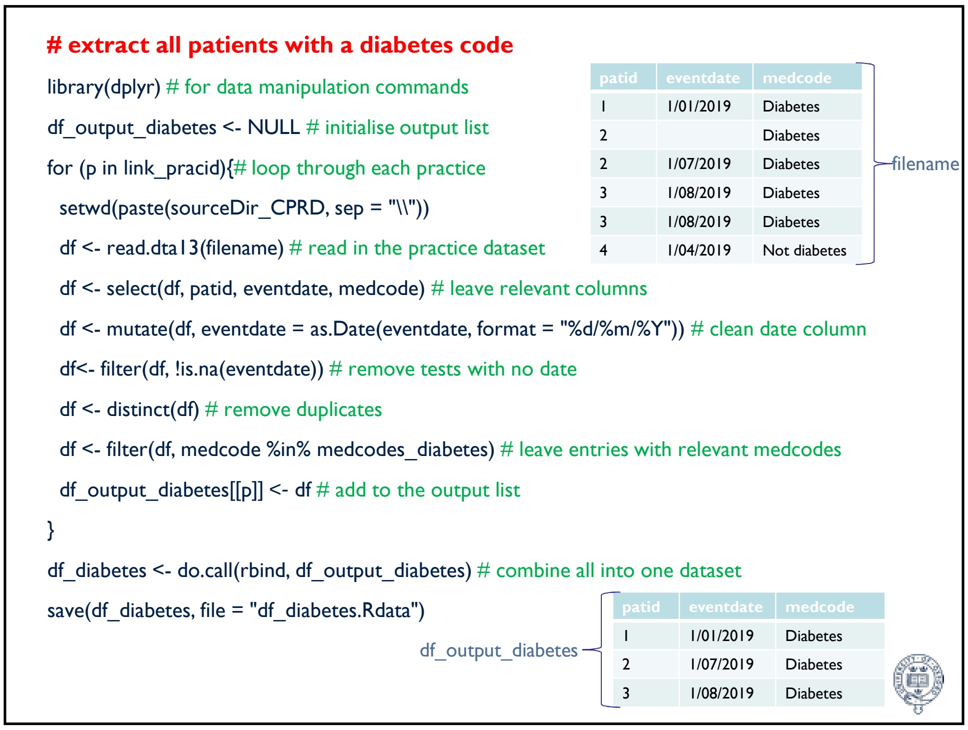


# Load a code list

```
library(data.table) # required for the fread function
.getcodes <- function(sourceDir_codelist, filename, colname){
        # set working directory
        setwd(sourceDir_codelist)
        # read in the .csv file with possible codes
        codes_t <- fread(filename, header = T)
        # filter out codes corresponding to the included=1 flag
        # the codes are recorded in the <colname> column
        return(codes_t$colname[included == 1])
}
```

# Illustration: code list for diabetes

```
medcodes_diabetes <- .getcodes(sourceDir_codelist = sourceDir_codelist,   filename =
        ”diabetes_final.csv", colname = colname)
```



# extract all patients with a diabetes code

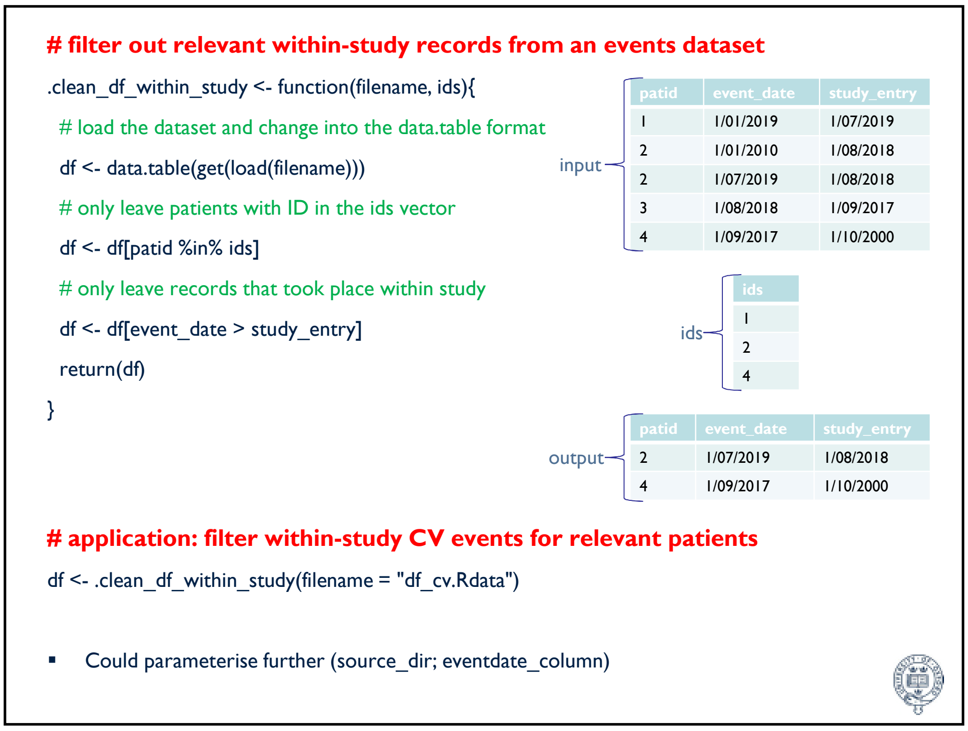
```
library(dplyr) # for data manipulation commands
df_output_diabetes <- NULL # initialise output list
for (p in link_pracid){# loop through each practice
  setwd(paste(sourceDir_CPRD, sep = "\\"))
  df <- read.dta13(filename) # read in the practice dataset
  df <- select(df, patid, eventdate, medcode) # leave relevant columns
  df <- mutate(df, eventdate = as.Date(eventdate, format = "%d/%m/%Y")) # clean date column
  df<- filter(df, !is.na(eventdate)) # remove tests with no date
  df <- distinct(df) # remove duplicates
  df <- filter(df, medcode %in% medcodes_diabetes) # leave entries with relevant medcodes
  df_output_diabetes[[p]] <- df # add to the output list
}
df_diabetes <- do.call(rbind, df_output_diabetes) # combine all into one dataset
save(df_diabetes, file = "df_diabetes.Rdata")
```

filename

patid	eventdate	medcode
1	1/01/2019	Diabetes
2		Diabetes
2	1/07/2019	Diabetes
3	1/08/2019	Diabetes
3	1/08/2019	Diabetes
4	1/04/2019	Not diabetes

df_output_diabetes

patid	eventdate	medcode
1	1/01/2019	Diabetes
2	1/07/2019	Diabetes
3	1/08/2019	Diabetes



# filter out relevant within-study records from an events dataset

```
.clean_df_within_study <- function(filename, ids){
  # load the dataset and change into the data.table format
  df <- data.table(get(load(filename)))
  # only leave patients with ID in the ids vector
  df <- df[patid %in% ids]
  # only leave records that took place within study
  df <- df[event_date > study_entry]
  return(df)
}
```

input

patid	event_date	study_entry
1	1/01/2019	1/07/2019
2	1/01/2010	1/08/2018
2	1/07/2019	1/08/2018
3	1/08/2018	1/09/2017
4	1/09/2017	1/10/2000

ids

ids
1
2
4

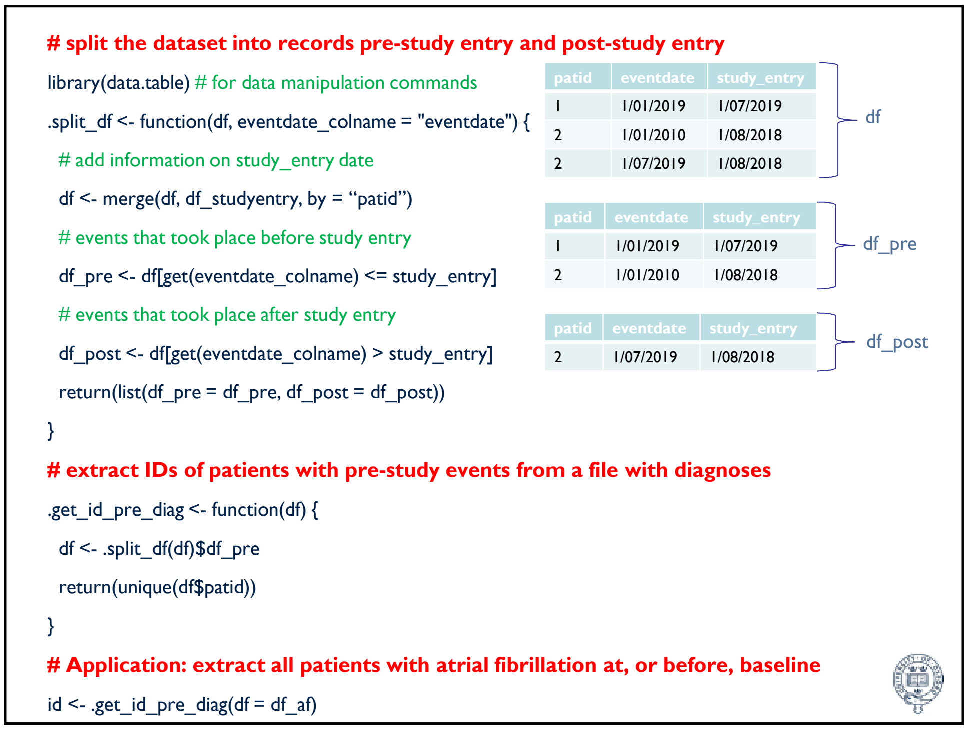
output

patid	event_date	study_entry
2	1/07/2019	1/08/2018
4	1/09/2017	1/10/2000

# application: filter within-study CV events for relevant patients

```
df <- .clean_df_within_study(filename = "df_cv.Rdata")
```

- Could parameterise further (source_dir; eventdate_column)



# split the dataset into records pre-study entry and post-study entry

```
library(data.table) # for data manipulation commands
.split_df <- function(df, eventdate_colname = "eventdate") {
  # add information on study_entry date
  df <- merge(df, df_studyentry, by = “patid”)
  # events that took place before study entry
  df_pre <- df[get(eventdate_colname) <= study_entry]
  # events that took place after study entry
  df_post <- df[get(eventdate_colname) > study_entry]
  return(list(df_pre = df_pre, df_post = df_post))
}
```

df

patid	eventdate	study_entry
1	1/01/2019	1/07/2019
2	1/01/2010	1/08/2018
2	1/07/2019	1/08/2018

df_pre

patid	eventdate	study_entry
1	1/01/2019	1/07/2019
2	1/01/2010	1/08/2018

df_post

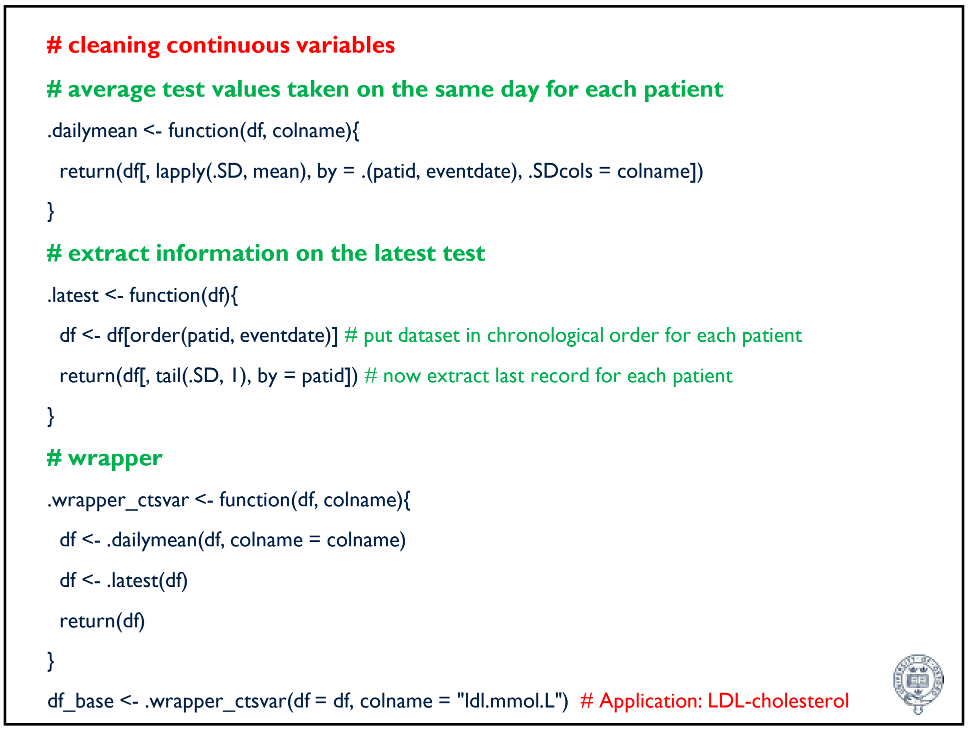
patid	eventdate	study_entry
2	1/07/2019	1/08/2018

# extract IDs of patients with pre-study events from a file with diagnoses

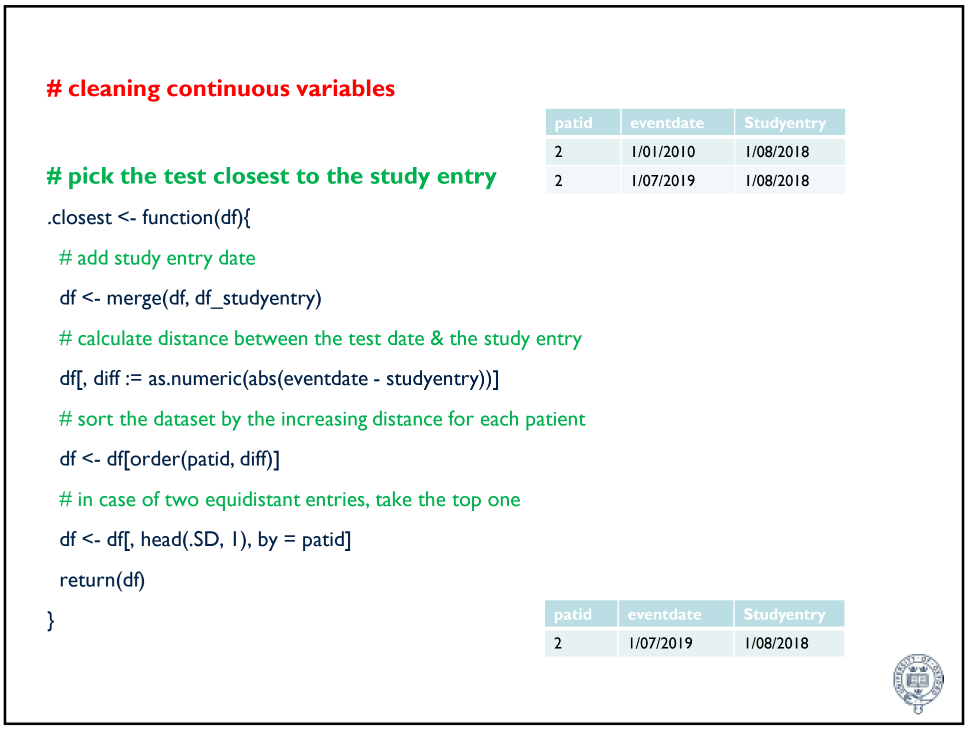
```
.get_id_pre_diag <- function(df) {
  df <- .split_df(df)$df_pre
  return(unique(df$patid))
}
```

# Application: extract all patients with atrial fibrillation at, or before, baseline

```
id <- .get_id_pre_diag(df = df_af)
```



```
# cleaning continuous variables
# average test values taken on the same day for each patient
.dailymean <- function(df, colname){
  return(df[, lapply(.SD, mean), by = .(patid, eventdate), .SDcols = colname])
}
# extract information on the latest test
.latest <- function(df){
  df <- df[order(patid, eventdate)] # put dataset in chronological order for each patient
  return(df[, tail(.SD, 1), by = patid]) # now extract last record for each patient
}
# wrapper
.wrapper_ctsvar <- function(df, colname){
  df <- .dailymean(df, colname = colname)
  df <- .latest(df)
  return(df)
}
df_base <- .wrapper_ctsvar(df = df, colname = "ldl.mmol.L")  # Application: LDL-cholesterol
```



```
# cleaning continuous variables
```

patid	eventdate	Studyentry
2	1/01/2010	1/08/2018
2	1/07/2019	1/08/2018

```
# pick the test closest to the study entry
.closest <- function(df){
  # add study entry date
  df <- merge(df, df_studyentry)
  # calculate distance between the test date & the study entry
  df[, diff := as.numeric(abs(eventdate - studyentry))]
  # sort the dataset by the increasing distance for each patient
  df <- df[order(patid, diff)]
  # in case of two equidistant entries, take the top one
  df <- df[, head(.SD, 1), by = patid]
  return(df)
}
```

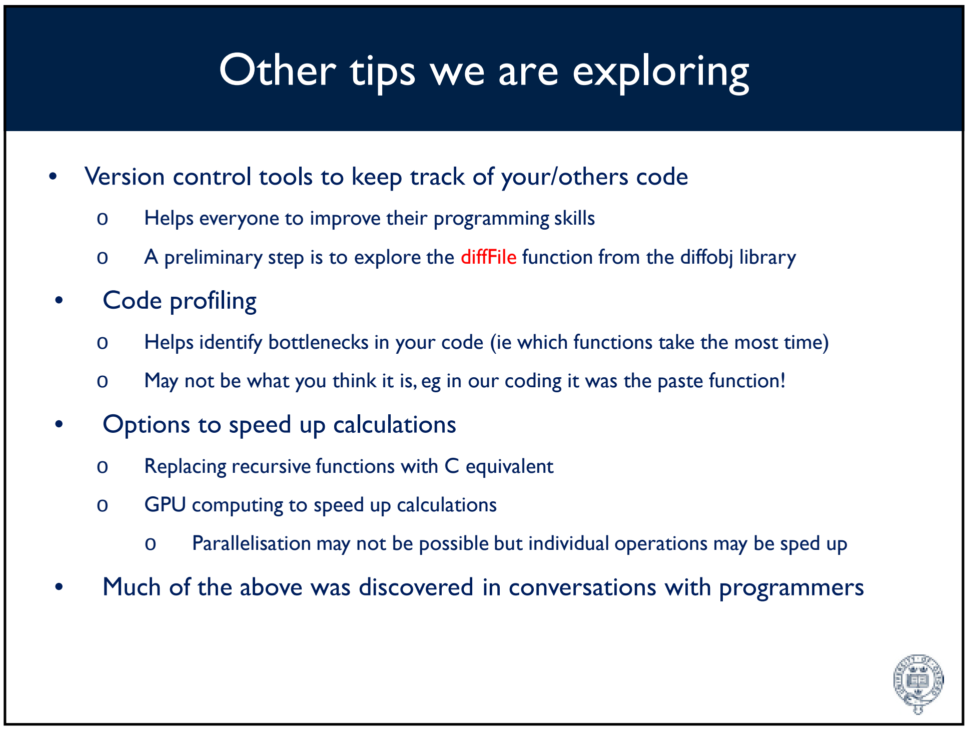
patid	eventdate	Studyentry
2	1/07/2019	1/08/2018

# extract IDs of patients who used a specific medication at/before study entry

```
get_id_pre_Tx <- function(df, days = 28) {
  df <- .split_df(df)$df_pre  # extract all prescriptions by baseline
  # require at least two prescriptions
  df_N <- df[, .N , by = patid] # count number of prescriptions by patient
  df_N <- df_N[N >= 2] # leave only patients with at least two prescriptions
  # of these, require the latest prescription at most <days> away from studyentry
  id <- df_N$patid # read off IDs of these patients
  df <- df[patid %in% id] # only leave patients with two prescriptions
  df <- df[, diff := as.numeric(studyentry - eventdate)] # distance between prescription and entry
  df <- df[diff <= days] # only leave patients with prescription within <days> of study entry
  return(unique(df$patid))
}
```

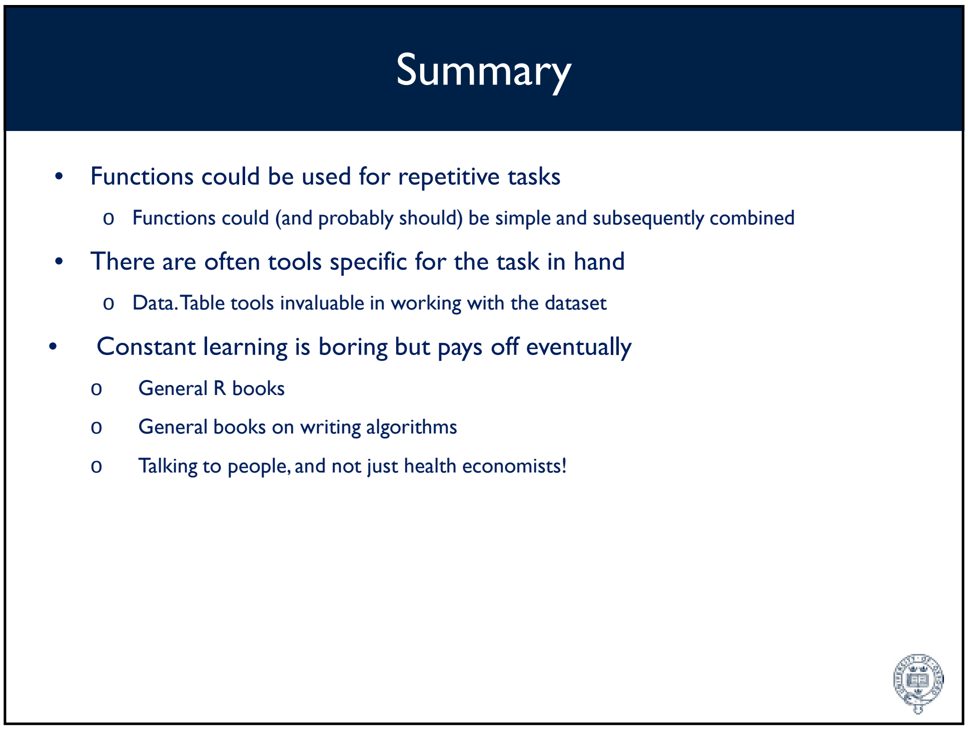
More tips
(written on blood, sweat and sleepless nights)

- Use packages designed for working with big data
  - merge from data.table reduced running time from 24 hours to 2 minutes
- Use meaningful file names
  - A must on collaborative projects
  - Helpful for replicating results
  - Create a master file with file descriptions
- Do debugging, many-many-many times
  - Debugging of the code
  - Face validity of results
  - Sanity-check analyses (set treatment effects to 0)
- Investing time in doing things properly pays off



Other tips we are exploring

- Version control tools to keep track of your/others code
  - Helps everyone to improve their programming skills
  - A preliminary step is to explore the diffFile function from the diffobj library
- Code profiling
  - Helps identify bottlenecks in your code (ie which functions take the most time)
  - May not be what you think it is, eg in our coding it was the paste function!
- Options to speed up calculations
  - Replacing recursive functions with C equivalent
  - GPU computing to speed up calculations
    - Parallelisation may not be possible but individual operations may be sped up
- Much of the above was discovered in conversations with programmers



Summary

- Functions could be used for repetitive tasks
  - Functions could (and probably should) be simple and subsequently combined
- There are often tools specific for the task in hand
  - Data.Table tools invaluable in working with the dataset
- Constant learning is boring but pays off eventually
  - General R books
  - General books on writing algorithms
  - Talking to people, and not just health economists!