Automatiseren en condities

Programming

Data Science & Business Analytics

AUTOMATISEREN

- Soms willen we code herhaald uitvoeren (zoals het aanroepen van een functie)
- Het is dan gemakkelijk als we dit proces kunnen automatiseren

- Dit kan met o.a.:
  - For loop
  - While loop

- Ook willen we soms checken of aan bepaalde condities wordt voldaan
- Dit kan met de if en else statements

FOR LOOP

- Een for loop:
  - Herhaalt een proces een vooraf bepaald aantal keer
  - Doet dit door een meegegeven reeks af te lopen
  - Stopt als het einde van de reeks bereikt is

```
a <- 0

for (i in 1:5)
{
        a <- a + 1
}
```

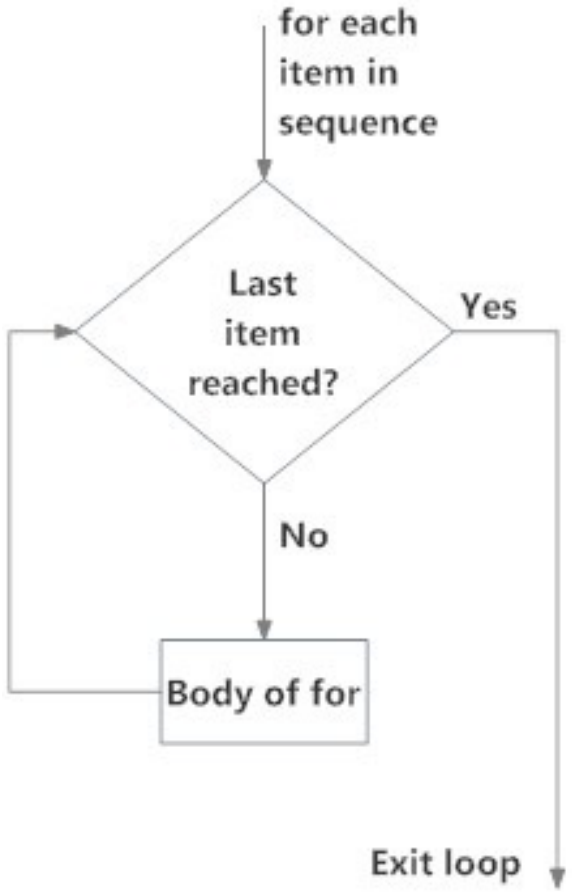


Fig: operation of for loop

FOR LOOP

- Een for loop kan met verschillende reeksen werken. De functie seq() is hiervoor handig.
- Tel terug van 5 naar 1 in stapjes van 0.5

```
for (i in seq(from = 5, to = 1, by=-0.5))
{
  a <- a + i
}
```

- Itereer over elementen in een lijst

```
klanten = list("Kwik", "Kwek", "Kwak")
for (i in 1:length(klanten))
{
   print(klanten[i])
}
```

WHILE LOOP

- Een while loop:
  - Herhaalt een proces totdat aan een vooraf opgegeven voorwaarde is voldaan

```
aantal_klanten <- 0
capaciteit <- 100

while (aantal_klanten < capaciteit)
{
      aantal_klanten <- aantal_klanten + 1
}
```

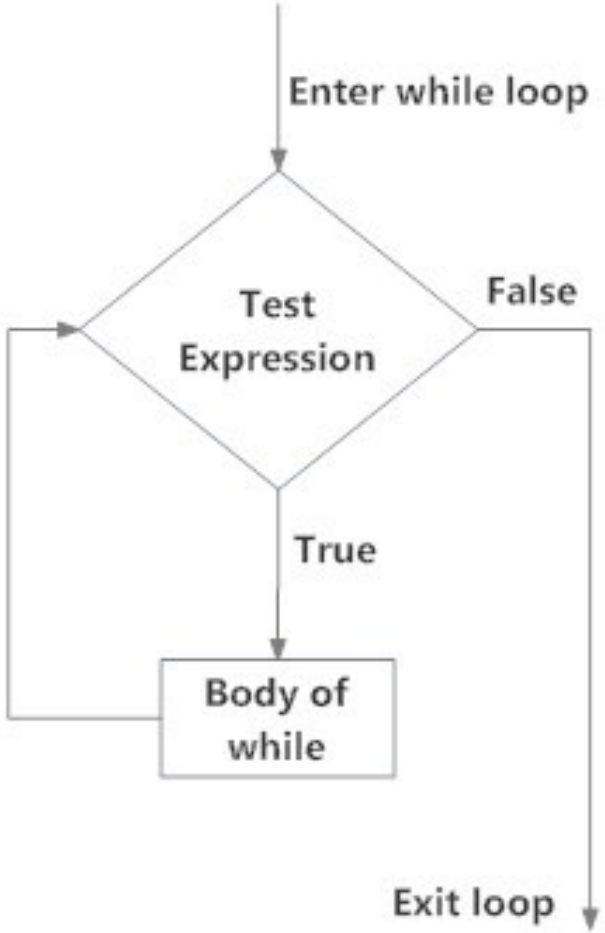


Fig: operation of while loop

IF ELSE STATEMENT

- Een if statement:
  - Voert code alleen uit als aan een voorwaarde is voldaan
- Een else statement
  - Voert code uit als niet aan de voorwaarde is voldaan

```
if (!is.null(klant_id))
{
        print(klant_id)
}else
{
        print('Klant ID is onbekend.')
}
```

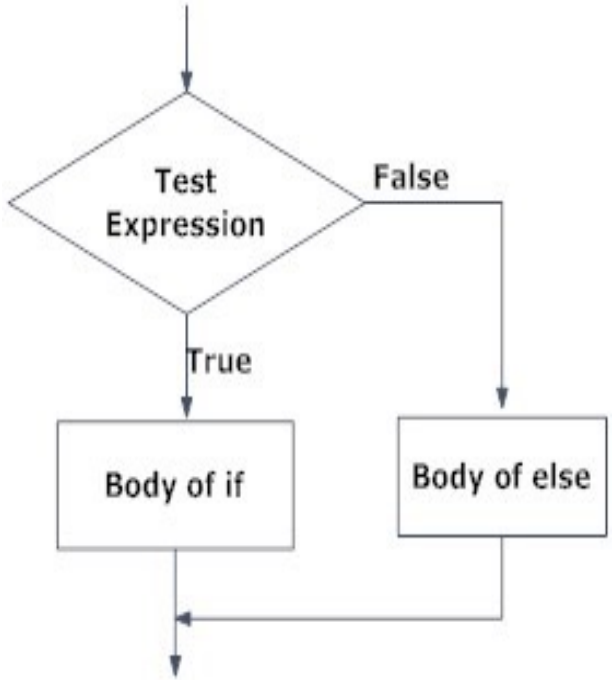


Fig: Operation of if...else statement

SPECIFICEREN VAN VOORWAARDEN

- We kunnen verschillende condities specificeren:
  - ! not vb: !is.null() Geeft aan dat iets NIET waar moet zijn
  - & and vb: a > 3 & a < 10 Geeft aan dat aan beide condities voldaan moet zijn
  - | or vb: b < 0 | b > 6 Geeft aan dat aan een van de twee condities voldaan moet zijn

```
if (!is.na(input) & (input <= 10 | input > 20))
{
        print('condition is TRUE')
}
```

IF ELSE STATEMENTS KUNNEN OOK GENEST WORDEN

```
if (!is.null(transactie_id))
{
	if (transactie_id == 1000)
	{
		print('Gefeliciteerd! U bent de 1,000e klant!')
	}else if (transactie_id < 1000)
	{
		print('Helaas! Volgende keer beter!')
	}else
	{
		print('Deze actie is afgelopen.')
	}
}else
{
	print('Transactie ID is onbekend.')
}
```

STATEMENTS COMBINEREN

- Loops en if else statements kunnen ook gecombineerd worden

- In dit voorbeeld zijn een while loop en een if else statement gecombineerd

```
worp <- 1
while (worp <= 6) {
  if (worp < 6) {
    print("Geen Yahtzee")
  } else {
    print("Yahtzee!")
  }
  worp <- worp + 1
}
```

EEN PRAKTISCH VOORBEELD

- Laten we kijken naar de Ben and Jerry dataset
- We willen de eenheidsprijs berekenen, maar we willen voorkomen dat er door 0 gedeeld wordt
- Dit kan met een for loop en if statement

```
BenAndJerry <- read_excel("BenAndJerry.xlsx")
nbObs <- length(BenAndJerry$household_id)
BenAndJerry$unit_price <- 0
for (i in 1:nbObs)
{
  if (BenAndJerry$quantity[i] > 0)
  {
    BenAndJerry$unit_price[i] <-
       BenAndJerry$price_paid[i]/BenAndJerry$quantity[i]
  }
}
```

COMBINATIE MET TIDYVERSE

- In het package tidyverse zit een speciale variant van de if else statement (de if_else() functie) die gebruikt kan worden in combinatie met mutate
- data %>% mutate(column_name = if_else(condition, value_if_true, value_if_false))

```
BenAndJerry %>%
       mutate(unit_price = if_else(quantity > 0, price_paid /
              quantity, 0))
```

OEFENING

- Laad de Ben and Jerry dataset in
- Automatiseer een proces dat voor iedere smaak ijs, een apart csv bestand maakt met de regels die betrekking hebben op die smaak
- Tips:
  - Zet de kolom flavor_description om met de functie as.factor
  - Bepaal de verschillende smaken met het commando levels(BenAndJerry$flavor_description)
  - Schrijf een bestand weg met de functie write.csv()

EFFICIËNT AUTOMATISEREN

- Tot nu toe heb je met for loops geleerd hoe je operaties kunt automatiseren
- Gaat het om grote datasets of heel veel operaties, dan is het belangrijk dat het automatiseren efficiënt (snel) gebeurt
- Bij lange berekeningen kan dit minuten of uren verschil maken

VOORBEELD VERTRAGINGEN

- Laten we weer kijken naar de flights dataset
- Eerder hebben we de vertragingen in seconden/minuten bekeken en geanalyseerd
- We willen deze vertraging nu graag indelen in categorieën:

Geen vertraging	0 minuten (of minder)
Weinig vertraging	0-30 minuten
Veel vertraging	30-180 minuten
Extreem veel vertraging	>180 minuten

- Uiteindelijk willen we weten welke vluchtmaatschappij de meeste extreme vertragingen heeft

VOORBEELD VERTRAGINGEN

Deze code doet wat we willen…

maar is niet efficiënt!

Waarom niet?

```
load("flights.RData")
prepared_flights <- prepare_flight_data(flights)

delay_levels <- c()

for (i in 1:nrow(prepared_flights)) {
   delay_mins <- prepared_flights$arr_delay[i]/dminutes(1)
   if (delay_mins <= 0) {
      delay_levels <- c(delay_levels,"no delay")
   } else if (delay_mins <= 30) {
      delay_levels <- c(delay_levels,"short delay")
   } else if (delay_mins <= 180) {
      delay_levels <- c(delay_levels,"long delay")
   } else {
      delay_levels <- c(delay_levels,"extreme delay")
   }
}
```

HAAL BEREKENINGEN BUITEN DE FOR-LOOP

```
load("flights.RData")
prepared_flights <- prepare_flight_data(flights)

delay_levels <- c()

for (i in 1:nrow(prepared_flights)) {
   delay_mins <- prepared_flights$arr_delay[i]/dminutes(1)
   if (delay_mins <= 0) {
      delay_levels <- c(delay_levels,"no delay")
   } else if (delay_mins <= 30) {
      delay_levels <- c(delay_levels,"short delay")
   } else if (delay_mins <= 180) {
      delay_levels <- c(delay_levels,"long delay")
   } else {
      delay_levels <- c(delay_levels,"extreme delay")
   }
}
```

Inefficiëntie #1

We zetten nu iedere vertraging los om naar minuten

Het is sneller om voor de for-loop al de hele kolom naar minuten om te zetten

HAAL BEREKENINGEN BUITEN DE FOR-LOOP

```
load("flights.RData")
prepared_flights <- prepare_flight_data(flights)

delay_levels <- c()
delay_mins <- prepared_flights$arr_delay/dminutes(1)

for (i in 1:nrow(prepared_flights)) {
   if (delay_mins[i] <= 0) {
      delay_levels <- c(delay_levels,"no delay")
   } else if (delay_mins[i] <= 30) {
      delay_levels <- c(delay_levels,"short delay")
   } else if (delay_mins[i] <= 180) {
      delay_levels <- c(delay_levels,"long delay")
   } else {
      delay_levels <- c(delay_levels,"extreme delay")
   }
}
```

Inefficiëntie #1

We zetten nu iedere vertraging los om naar minuten

Het is sneller om voor de for-loop al de hele kolom naar minuten om te zetten

GEEF OBJECTEN EEN VASTE GROOTTE

```
load("flights.RData")
prepared_flights <- prepare_flight_data(flights)

delay_levels <- c()
delay_mins <- prepared_flights$arr_delay/dminutes(1)

for (i in 1:nrow(prepared_flights)) {
   if (delay_mins[i] <= 0) {
      delay_levels <- c(delay_levels,"no delay")
   } else if (delay_mins[i] <= 30) {
      delay_levels <- c(delay_levels,"short delay")
   } else if (delay_mins[i] <= 180) {
      delay_levels <- c(delay_levels,"long delay")
   } else {
      delay_levels <- c(delay_levels,"extreme delay")
   }
}
```

Inefficiëntie #2

We beginnen nu met een vector van lengte 0, en maken die in iedere iteratie 1 element groter

Het is sneller om vooraf een lege vector te maken met de juiste grootte, en deze te vullen in de loop

GEEF OBJECTEN EEN VASTE GROOTTE

```
load("flights.RData")
prepared_flights <- prepare_flight_data(flights)

delay_levels <- vector(mode="character", length=nrow(prepared_flights))
delay_mins <- prepared_flights$arr_delay/dminutes(1)

for (i in 1:nrow(prepared_flights)) {
   if (delay_mins[i] <= 0) {
      delay_levels[i] <- "no delay"
   } else if (delay_mins[i] <= 30) {
      delay_levels[i] <- "short delay"
   } else if (delay_mins[i] <= 180) {
      delay_levels[i] <- "long delay"
   } else {
      delay_levels[i] <- "extreme delay"
   }
}
```

Inefficiëntie #2

We beginnen nu met een vector van lengte 0, en maken die in iedere iteratie 1 element groter

Het is sneller om vooraf een lege vector te maken met de juiste grootte, en deze te vullen in de loop

GEBRUIK LOGICAL INDEXING WAAR MOGELIJK

```
load("flights.RData")
prepared_flights <- prepare_flight_data(flights)

delay_levels <- vector(mode="character", length=nrow(prepared_flights))
delay_mins <- prepared_flights$arr_delay/dminutes(1)

for (i in 1:nrow(prepared_flights)) {
   if (delay_mins[i] <= 0) {
      delay_levels[i] <- "no delay"
   } else if (delay_mins[i] <= 30) {
      delay_levels[i] <- "short delay"
   } else if (delay_mins[i] <= 180) {
      delay_levels[i] <- "long delay"
   } else {
      delay_levels[i] <- "extreme delay"
   }
}
```

Inefficiëntie #3

We doen nu 1 voor 1 een check voor iedere vlucht afzonderlijk

Met logical indexing kunnen we alle checks in 1 keer doen!

GEBRUIK LOGICAL INDEXING WAAR MOGELIJK

```
load("flights.RData")
prepared_flights <- prepare_flight_data(flights)

delay_levels <- vector(mode="character", length=nrow(prepared_flights))
delay_mins <- prepared_flights$arr_delay/dminutes(1)

delay_levels[delay_mins <= 0] <- "no delay"
delay_levels[delay_mins > 0 & delay_mins <= 30] <- "short delay"
delay_levels[delay_mins > 30 & delay_mins <= 180] <- "long delay"
delay_levels[delay_mins > 180] <- "extreme delay“
```

Inefficiëntie #3

We doen nu 1 voor 1 een check voor iedere vlucht afzonderlijk

Met logical indexing kunnen we alle checks in 1 keer doen!

OOK SNEL: MUTATE MET IF_ELSE

```
load("flights.RData")
prepared_flights <- prepare_flight_data(flights)

prepared_flights %>%
  mutate(delay_levels = if_else(delay_mins <= 0, "no delay",
                        if_else(delay_mins <= 30, "short delay",
                        if_else(delay_mins <= 180, "long delay",
                        "extreme delay"))))
```

lets langzamer dan logical indexing, maar iets sneller dan de for loop

VERTRAGINGEN PER VLUCHTMAATSCHAPPIJ

- We kunnen nu een tabel maken van het aantal vluchten in iedere vertragingscategorie, per vluchtmaatschappij
- Om ervoor te zorgen dat de categorieën logisch geordend zijn, maken we een ordered factor van de delay levels

```
prepared_flights$delay_levels <- factor(delay_levels, levels = c("no
                                 delay", "short delay", "long delay",
                                 "extreme delay"), ordered=TRUE)

delay_table <- table(prepared_flights$carrier, prepared_flights$delay_levels)
```

VERTRAGINGEN PER VLUCHTMAATSCHAPPIJ

Omdat we een ordered factor hebben gemaakt, zijn de levels niet alfabetisch gesorteerd, maar van geen naar veel vertraing

Welke vluchtmaatschappij presteert hier het best?

Moeilijk te vergelijken…

	no delay	short delay	long delay	extreme delay
9E	10657	3544	2828	316
AA	21241	6879	3625	314
AS	520	128	61	3
B6	30440	14188	8877	632
DL	31245	10881	5088	532
EV	26624	12818	10872	937
F9	289	221	153	18
FL	1280	1230	595	76
HA	245	78	18	1
MQ	13344	7256	4229	247
OO	19	4	6	0
UA	35560	14344	7416	596
US	12482	5301	1938	121
VX	3370	1131	527	99
WN	6740	3378	1743	214
YV	286	132	114	12

OEFENING

- Zorg ervoor dat de tabel omgezet wordt naar proporties per vluchtmaatschappij; de rijen moeten dus optellen tot 1

```
      no delay short delay  long delay extreme delay
9E 0.614413376 0.204324013 0.163044105   0.018218507
AA 0.662559656 0.214573131 0.113072772   0.009794441
AS 0.730337079 0.179775281 0.085674157   0.004213483
...
```

- Probeer je code zo efficiënt mogelijk te maken.

OMGAAN MET ERRORS

- Of je een vaardige programmeur bent of niet: errors krijgen overkomt iedereen regelmatig
- Errors herkennen en oplossen is een belangrijke programmeervaardigheid:
  - Deels een kwestie van ervaring
  - Met een aantal tips kom je een heel eind

```
> price_1 <- 2
> price_2 <- 4
> av_price <- mean(price_1, price_2)
> quantity <- 10
> revenue <- quantity * av_price
> print(revenue)
```

```
[1] 20
```

Geen error betekent niet per se een foutloze code; soms zie je liever wel dan niet…

Wat gaat hier fout?

BEGIN BIJ DE EERSTE ERROR

- Een fout aan het begin van je code kan veel errors erna veroorzaken
- Begin bij de eerste error die je ziet, vaak los je daar latere errors ook mee op

```
mileages <- c(10,20,15,18,12)
av_mileage <- average(mileages)
costs_per_mile <- 3
av_costs <- av_mileage * costs_per_mile
print(av_costs)
```

```
> mileages <- c(10,20,15,18,12)
> av_mileage <- average(mileages)
Error in average(mileages): could not find function "average"
> costs_per_mile <- 3
> av_costs <- av_mileage * costs_per_mile
Error: object 'av_mileage' not found
> print(av_costs)
Error in print(av_costs): object 'av_costs' not found
```

ERRORS LEZEN

- De error message in R geeft je vaak een aanwijzing
  - Waar de fout plaatsvindt
  - Wat er fout gaat

```
> average(5,7)
Error in average(5, 7) : could
not find function "average"
```

```
> delay_table[17,]
Error in `[.default`(delay_table, 17, ) :
subscript out of bounds
```

```
> 1,5 * 3,7
Error: unexpected ',' in "1,"
```

```
> for i in 1:10 { print(i) }
Error: unexpected symbol in "for i"
```

WAT ALS JE EEN ERROR NIET BEGRIJPT?

- Soms zijn error messages moeilijker te begrijpen
- Vaak kun je nog steeds aanwijzingen naar het probleem terugvinden

```
> load("BenAndJerry.csv")
Error in load("BenAndJerry.csv") :
  bad restore file magic number (file may be corrupted) -- no data loaded
In addition: Warning message:
file 'BenAndJerry.csv' has magic number '"hous'
  Use of save versions prior to 2 is deprecated
```

Er is blijkbaar een probleem met het bestand…

- Heb je nog steeds geen idee wat het probleem is? Google is je beste vriend :)
- Er zijn ongetwijfeld anderen geweest met hetzelfde probleem

Google

Error in load("") : bad restore file magic number (file may be corruptec

Alle | Video's | Nieuws | Shopping | Afbeeldingen | Meer | Tools

Ongeveer 1.100.000 resultaten (0,69 seconden)

https://community.rstudio.com › err... · Vertaal deze pagina

error message: bad restore file magic number (file ma

6 jan. 2020 — Unfortunately, that means the file that **the workspace was sa**
there. This could have by passing the .RData path to a ...
1 antwoord · Topantwoord: Write() vs Save() I expect someone did something

https://stackoverflow.com › questions · Vertaal deze pagina

The cause of "bad magic number" error when loading

30 jan. 2018 — This error indicates you **are trying to load a non-valid file t**
some reason, R no longer recognizes this file as an R workspace file.
10 antwoorden · Topantwoord: I got that error when I accidentally used load()

R: Use of save versions prior to 2 is deprecated - Stack Overflow 27 okt.
How to remove **error** : **bad restore file magic number** (**file may** ... 17 jun.
"bad restore file magic number" error in R - Stack Overflow 8 mei 2
workspace cannot be **loaded** in server, **file** has **magic number** ... 28 jul.
Meer resultaten van stackoverflow.com

Vaak beland je op forums als StackOverflow

error message: bad restore file magic number (file may ...

General | rstudio

1 Reply 1

created	last reply	replies	views	users	likes
2020-01-06	2020-01-14	5	49.9k	2	5

grosscol 2020-01-06

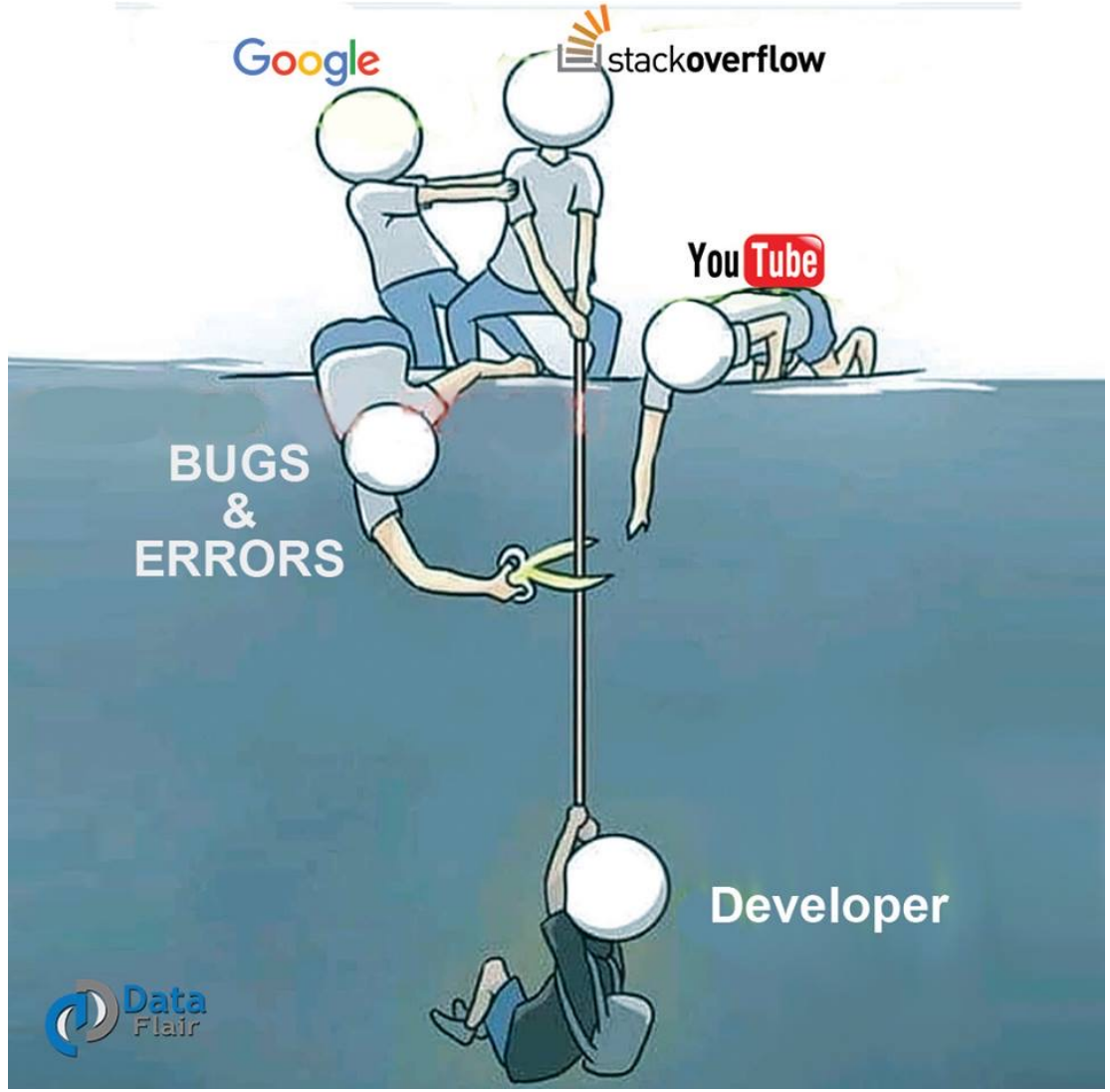
It's probably not an RData file.

Likely it's a file that got misnamed with the wrong extension. The magic number should start with "RDA2" or something like that. File format magic numbers are traditionally the first bytes of a file. Seeing something that looks like a fragment of a column name leads one to believe that it's not an RData file, some plain text that begins with "Featu"

> tushard1987:
>
> file 'ecallcx subsets.RData' has magic number 'Featu'

Could be a CSV or TSV file

Is the first column of the data named "Feature"? It might be an issue of a someone using `write()` instead of `save()` to serialize a data frame to disk. If that is the case, `read.csv()` or


Google
stackoverflow
YouTube
BUGS
&
ERRORS
Developer
Data
Flair

TRACEBACKS

- Vindt de error in een functie plaats, dan kun je vaak de traceback bekijken; die wijst aan waar de error precies plaatsvond

```
count_flights <- function(flight_data, org, dst) {
  prep <- flight_data %>%
    prepare_flight_data()

  out <- prep %>%
    filter(origin == org, dest == dst)

  out1 <- nrow(out)
  out2 <- mean(out$arr_delay/dminutes(1)>30)

  return(list(n_flights = out1, perc_delayed = out2))
}
```

```
> count_flights("JFK","ANC")
```

```
Error in UseMethod("mutate") :
  no applicable method for 'mutate' applied to an object of class "character"
```

Show Traceback

Rerun with Debug

Blijkbaar vindt de error plaats in de “mutate”-stap van de “prepare_flight_data” functie

```
> count_flights("JFK","ANC")
```

```
Error in UseMethod("mutate") :
  no applicable method for 'mutate' applied to an object of class "character"
```

Show Traceback

Rerun with Debug

```
 7. mutate(., arr_time = ymd_hm(arr_time), sched_arr_time = ymd_hm(sched_arr_time),
        dep_time = ymd_hm(dep_time), sched_dep_time = ymd_hm(sched_dep_time),
        arr_delay = arr_time - sched_arr_time, dep_delay = dep_time -
            sched_dep_time, air_time = arr_time - dep_time, sched_air_time = sched_arr_time -  ...
 6. filter(., !is.na(arr_time))
 5. flight_data %>% mutate(arr_time = ymd_hm(arr_time), sched_arr_time = ymd_hm(sched_arr_time),
        dep_time = ymd_hm(dep_time), sched_dep_time = ymd_hm(sched_dep_time),
        arr_delay = arr_time - sched_arr_time, dep_delay = dep_time -
            sched_dep_time, air_time = arr_time - dep_time, sched_air_time = sched_arr_time -  ...
 4. prepare_flight_data(.)
 3. filter(., origin == org, dest == dst)
 2. flight_data %>% prepare_flight_data() %>% filter(origin == org,
        dest == dst)
 1. count_flights("JFK", "ANC")
```

Onderaan staat ons commando, daarboven de stappen in de functie die zijn gevolgd tot de error ontstond

EEN SCRIPT DEBUGGEN

- Word je niet veel wijzer uit de traceback?
- Dan biedt RStudio een aantal debug-functionaliteiten

```
> count_flights(flights, JFK, MIA)
Error: Problem with `filter()` input `..1`.
i Input `..1` is `origin == org`.
x object 'JFK' not found
Run `rlang::last_error()` to see where the error occurred.
```

Show Traceback
Rerun with Debug

```
12. stop(fallback)
11. signal_abort(cnd)
10. abort(c(cnd_bullet_header(), i = cnd_bullet_input_info(), x = conditionMessage(e),
        i = cnd_bullet_cur_group_label()), class = "dplyr_error")
 9. h(simpleError(msg, call))
 8. .handleSimpleError(function (e)
    {
        local_call_step(dots = dots, .index = env_filter$current_expression,
            .fn = "filter") ...
 7. mask$eval_all_filter(dots, env_filter)
 6. withCallingHandlers({
        dots <- new_quosures(flatten(imap(dots, function(dot, index) {
            env_filter$current_expression <- index
            expand_if_across(dot) ...
 5. filter_rows(.data, ..., caller_env = caller_env())
 4. filter.data.frame(., origin == org, dest == dst)
 3. filter(., origin == org, dest == dst)
 2. prep %>% filter(origin == org, dest == dst)
 1. count_flights(flights, JFK, MIA)
```

DE BROWSER() FUNCTIE

- Met de 'browser()' functie kun je een functie tussentijds pauzeren
- Plaats browser() in de body van je functie, voor het punt waar de error plaatsvindt
- Weet je nog niet waar de error plaatsvindt? Plaats browser() dan aan het begin

```
count_flights <- function(flight_data, org, dst) {
  browser()
  prep <- flight_data %>%
    prepare_flight_data()

  out <- prep %>%
    filter(origin == org, dest == dst)

  out1 <- nrow(out)
  out2 <- mean(out$arr_delay/dminutes(1)>30)

  return(list(n_flights = out1, perc_delayed = out2))
}
```

DE BROWSER() FUNCTIE

- Zodra je de code opnieuw runt, verschijnt er geen error, maar een Browse[1] > command prompt
- Terwijl de functie is gepauzeerd, kun je tussentijdse variabelen bekijken en commando's uitvoeren
- Om de gemarkeerde regel uit te voeren typ je n + Enter

```
> count_flights(flights, JFK, MIA)
Called from: count_flights(flights, JFK, MIA)
Browse[1]>
```

```
> count_flights(flights, JFK, MIA)
Called from: count_flights(flights, JFK, MIA)
Browse[1]> nrow(flights)
[1] 336776
Browse[1]> n
debug at #3: prep <- flight_data %>% prepare_flight_data()
Browse[2]>
```

Function: count_flights (.GlobalEnv)

```
1  function(flight_data, org, dst) {
2    browser()
3    prep <- flight_data %>%
4      prepare_flight_data()
5
6    out <- prep %>%
7      filter(origin == org, dest == dst)
8
9    out1 <- nrow(out)
10   out2 <- mean(out$arr_delay/dminutes(1)>30)
11
12   return(list(n_flights = out1, perc_delayed = out2))
13 }
```

Function: count_flights (.GlobalEnv)

```
1  function(flight_data, org, dst) {
2    browser()
3    prep <- flight_data %>%
4      prepare_flight_data()
5
6    out <- prep %>%
7      filter(origin == org, dest == dst)
8
9    out1 <- nrow(out)
10   out2 <- mean(out$arr_delay/dminutes(1)>30)
11
12   return(list(n_flights = out1, perc_delayed = out2))
13 }
```

DE BROWSER() FUNCTIE

- Zodra de error plaatsvindt, stopt het “browsen”: je weet nu in welke regel de error plaatsvindt
- Je kunt ook eerder stoppen, door Q + Enter te typen
- Met c + Enter voer je alle resterende code in 1 keer uit

```
> count_flights(flights, JFK, MIA)
Called from: count_flights(flights, JFK, MIA)
Browse[1]> nrow(flights)
[1] 336776
Browse[1]> n
debug at #3: prep <- flight_data %>% prepare_flight_data()
Browse[2]> n
debug at #6: out <- prep %>% filter(origin == org, dest == dst)
Browse[2]> n
Error: Problem with `filter()` input `..1`.
i Input `..1` is `origin == org`.
x object 'JFK' not found
Run `rlang::last_error()` to see where the error occurred.
In addition: Warning message:
In .Call(dplyr_mask_eval_all_filter, quos, private, nrow(private$data),  :
  restarting interrupted promise evaluation
`
```

OEFENING

- Bekijk de volgende functie, die in de onderste regel wordt uitgevoerd
- Er staan meerdere fouten in de functie, die ieder leiden tot een error
- Vind alle fouten en verbeter ze

```
count_flights <- function(flight_data, org, dst) {
  prep <- flight_data %>%
    prepare_flight_data()

  out <- prep %>%
    filter(origin = org, dest = dst)

  out1 <- nrow(out)
  out2 <- mean(out$arr_delay/dminutes>30)

  return(list(n_flights = out1), perc_delayed = out2)
}

load("flights.RData")
count_flights(flights, "JFK", "MIA")
```