Registerdata, R og bruk av dato

Are Edvardsen, SKDE

2018-07-11

Problembeskrivelse

Vanlig arbeidsflyt i framstilling av registerinformasjon på ved bruk av R på Rapporteket består forenklet av 1) å spørre om data fra en database, 2) gjøre talloperasjoner og 3) presentere resultatet i form av tabeller, figurer og tekst. I registerdata vil det bestandig finnes ulike former for tidsinformasjon slik som dato og klokkeslett. Følgelig er det da også normalt å måtte forholde seg til tid i de talloperasjoner som gjøres underveis fram til et ferdig resultat. I R er det mange pakker og enda flere funksjoner som er aktuelle for bruk på tidsvariabler. Felles for slike "kalender-funksjoner"" er at det ofte kan være vanskelig å få full oversikt og forståelse av hvordan de virker. I praktisk bruk forsøker man gjerne flere varianter til man har noe som synes å gi et tilforlatelig svar. Slik tilnærming vil gi en større risiko for feil. Ved bruk av et konkret eksempel er det her gjort et forsøk på redegjøre for noen vanlig brukte kalenderfunksjoner slik at at risiko for feil ved framtidig bruk reduseres.

Eksempel på feil som oppstår

Følgende eksempel gir en reell reproduksjon av feil oppdaget i *NorGAst* der endepunktet i datoutvalget ikke kommer med i analysen. Kommandoene under er kjørt på *Rapporteket* TEST. Av hensyn til forenkling er det bare benyttet en øvre (*datoTil*) og ingen nedre (*datoFra*) begrensning av dato.

Angi register, databasemotor og sluttdato:

```
registryName <- "norgast"
dbType <- "mysql"
datoTil <- "2017-07-31"
```

Lag databasespørring:

```
query <- paste0("SELECT
                  a.OpDato
                FROM
                  AlleVariablerNum a
                INNER JOIN
                  ForlopsOversikt f
                ON
                  a.ForlopsID = f.ForlopsID
                WHERE
                  f.HovedDato <= \'", datoTil, "'")
```

Hent data:

```
RegData <- rapbase::LoadRegData(registryName, query, dbType)
```

Lag ny variabel *OperasjonsDato* slik det er gjort i *NorGAst*, sjekk maxverdi og klasse på opprinnelig og ny variabel:

```
RegData$OperasjonsDato <- as.POSIXlt(RegData$OpDato, format="%Y-%m-%d")

max(RegData$OpDato)
[1] "2017-07-31"
```

```
class(RegData$OpDato)
[1] "Date"

max(RegData$OperasjonsDato)
[1] "2017-07-31 UTC"
class(RegData$OperasjonsDato)
[1] "POSIXlt" "POSIXt"
```

Maksimumsverdi stemmer med den som er oppgitt i spørringen. Legg merke til at opprinnelig variabel er av klassen *Date* og at tidssone ikke er angitt. Ved konvertering til klassen *POSIXlt* benyttes default tidssone UTC.

Det neste som skjer i *NorGAst* er en (ekstra) filtrering på angitt datobegrensning:

```
indDato <- which(RegData$OperasjonsDato <= as.POSIXlt(datoTil))

length(RegData$OperasjonsDato)
[1] 14574
length(indDato)
[1] 14554

max(RegData$OperasjonsDato[indDato])
[1] "2017-07-30 UTC"
```

Selv om datobegrensningen er den samme som i opprinnelig spørring mot databasen (og at man da forventer at ingen verdier filtreres ut) så tas det i dette tilfellet likevel vekk 20 observasjoner (som alle har Operasjonsdato 2017-07-31). Årsaken ligger i ulik tidssone for de verdier som sammenholdes i filteret over:

```
as.POSIXlt(datoTil)
[1] "2017-07-31 CEST"
```

CEST (Central European Summer Time) ligger to timer foran UTC og siden tidsinformasjonen som benyttes bare inneholder dato (og ikke klokkeslett) så filtreres disse ut fra datagrunnlaget. I et tenkt tilfelle kunne alle 20 observasjoner fra 31. juli vært registrert kl 22:01 og i så fall vil det gi et korrekt utfall at disse filtreres vekk når sommertid gjelder mens for vintertid vil det gitt feil utfall. I realiteten er klokkeslett ikke kjent og da vil den logiske operatoren som benyttes i filteret fungere som beskrevet over for POSIX-klassene, altså at alle opersjonsdatoer som er lik *datoTil* men som ligger "øst" for UTC filtreres ut.

Og hvorfor ender man opp med datoer fra ulike tidssoner?

For verdiene i *OperasjonsDato* er svaret allerede gitt over. Ved konvertering fra klassen *Date* til klassen *POSIXlt* så er ikke tidssone definert og det benyttes da default tidssone UTC. Ved konvertering av *datoTil* fra klassen *character* til klassen *POSIXlt* spør funksjonen *as.POSIXlt* egen server hvilken tidssone den befinner seg i. I eksempelet over fikk funksjonen svaret "CEST" (UTC+2). Den samme funksjonen ville gitt "CET" (UTC+1) om den ble kjørt i desember. Bruk av klassen *POSIXct* ville gitt samme resultat. En digresjon angående disse klassene er at "ct" (calendar time) forholder seg til antall sekunder fra et gitt tidspunkt (the epoch, *i.e.* 1970-01-01 UTC) mens "lt" (local time) inneholder en liste med vektorer som hver angir år, måned, dag, time, osv.

Mulige løsninger

I eksempelet over introduseres feilen ved bruk av POSIX-klassene og den påfølgende bruken av tidssoner. Selv om det ikke kan utelukkes helt så vil praktisk håndtering av data fra norske kvalitetsregistre ikke ha behov for bruk av tidssone. I så måte kunne man unngått bruk av disse klassene for datoer. Men, det kan

finnes mange gode grunner til å fortsette bruken og da foreutsetter det at man håndterer tidssoner riktig. I eksempelet over kunne man tatt vekk datofilteret i R-koden da det allerede er definert i sql (spørringen til databasen). Om man likevel ønkser å ivareta et slik filter i R-koden ved bruk av *POSIX* kan dette gjøre gjennom å definere samme tidssone eksplisitt for alle variabler:

```
RegData$OperasjonsDato <- as.POSIXlt(RegData$OpDato, tz="UTC", format="%Y-%m-%d")
...
indDato <- which(RegData$OperasjonsDato <= as.POSIXlt(datoTil, tz="UTC"))
```

Definisjon av tidssone i den første linja er strengt tatt ikke nødvendig da UTC vil være default i dette eksempelet. Men (OBS, OBS), om datoformatet i databasen hadde vært et annet så kunne det likevel være nødvendig å håndtere tidssone eksplisitt. Se under.

Dato og tidsformater i MySQL (og ikke MSSQL)

MySQL (den mest vanlige databasemotoren på *Rapporteket*) benytter tre ulike tidsformater: DATE, DATETIME og TIMESTAMP. De to første definerer ikke tidssone mens den siste gjør det. Felter med format TIMESTAMP vil gjennom *DBI* være definert som klassen *POSIXct* og dermed ivareta tidssone når verdiene representeres i R. Den tekniske manualen for MySQL gir flere detaljer som er vel verdt å lese. Rapportutviklere som benytter data fra en database (gjennom sql) bør derfor være oppmerksom på de formater som benyttes i databasen når spørringen settes opp for å sikre at videre tallbehandling i R gjøres riktig.