1. Del Raw data & Transformation

Jan Maurycy Pedersen, Jolene Kaye Jensen, Lukas Beslic Christiansen

Table of contents

1 Datagrundlag og data management

Dette dokument beskriver de centrale datahåndterings- og transformationsprocesser, der danner grundlag for projektets efterfølgende analyser og modeller. Fokus er rettet mod dokumentation af datagrundlag, behandlingslogik og kvalitetssikring frem for fuld teknisk reproducerbarhed. Dette skyldes, at væsentlige dele af datagrundlaget er placeret i et adgangsbegrænset og betalt SQL-baseret datalager, som suppleres af eksterne offentlige datakilder.

Af hensyn til databeskyttelse, systemadgang og omkostningsstyring stilles der ikke direkte adgang til SQL-databasen eller det samlede rå datagrundlag til rådighed. Dokumentationen skal derfor læses som en faglig og metodisk redegørelse for, hvordan data indsamles, behandles, transformeres og klargøres til analyse. Redegørelsen understøttes af dokumenterede resultater, strukturelle outputs og beskrivende eksempler, hvor det er relevant.

1.1 PBA01_Raw

I dette afsnit præsenteres opbygningen af projektets datagrundlag, som er struktureret efter principper inspireret af klassisk data management og data warehousing. I stedet for at arbejde direkte på webscrapede tabeller og ad hoc-datasæt er datagrundlaget opdelt i tydelige og adskilte lag, der hver har et klart og afgrænset formål i det samlede dataflow.

Databasen er organiseret i tre overordnede schemaer: PBA01_Raw, PBA02_Clean og PBA03_JoinReady. RAW-laget fungerer som et landingslag, hvor data gemmes tæt på kilden og uden analytisk fortolkning. CLEAN-laget anvendes til nødvendig rensning, standardisering og validering af data, mens JOIN-READY-laget indeholder datasæt, der er struktureret og konsolideret med henblik på analyse og modellering.

Denne lagdelte struktur muliggør en robust og transparent datahåndtering, hvor fejl kan isoleres, enkelte trin kan genkøres, og ændringer i eksterne datakilder håndteres uden at påvirke hele datagrundlaget. Samtidig understøtter arkitekturen en klar adskillelse mellem rå data, behandlet data og analyseklar data, hvilket er centralt i et projekt med både kvantitative analyser og maskinlæringsbaserede modeller.

1.1.1 Webscrapping af Superligaens kampprogram (superstats.dk)

Denne kode udgør et fuldt RAW-scrape af Superligaens kampprogram direkte fra SuperStats' HTML-sider. Fokus er bevidst på at udtrække data så tæt på kilden som muligt, med kun minimal teknisk strukturering, så alle fortolkninger, beregninger og joins kan foretages konsekvent i senere clean, join-lag, og analyse lag.

1.1.1.1 Pakker

Dette afsnit indlæser de nødvendige R-pakker til webscraping og basal datamanipulation. Pakkerne er valgt ud fra faktisk anvendelse i dette RAW-scrape for at sikre et let, gennemsigtigt og reproducerbart setup.

```
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")

  pacman::p_load(
    rvest, dplyr, stringr, purrr, tibble
  )
});  cat("Pakker klar og loaded \n")
```

1.1.1.2 Hjælpefunktion: RAW-parse af én runde-tabel

Denne funktion udtrækker data fra én HTML-tabel for en given runde og gemmer felterne i et RAW-format tæt på kilden. Der foretages kun minimal strukturering (fx udtræk af rundenummer og trim af tekst), mens al egentlig fortolkning og konvertering flyttes til senere clean- og join-lag.

```
# ======================================================================
# 1) Hjælpefunktion til at parse ÉN runde-tabel (RAW-niveau)
# ============================================================================
# VIGTIGT: Dette er et RAW-parse:
#  - Vi trækker kun det ud, som fremgår direkte af HTML-tabellen.
#  - Vi laver KUN den manipulation, der er nødvendig for overhovedet
#    at få data ud i kolonner (ikke beregninger som ugedag, dato-objekter osv.).
#
# Output-kolonner (tæt på kildens struktur):
#  - season_value   : sæson-parameteren fra URL'en (fx "2000")
#  - runde_header   : rå tekst i tabel-header, fx "Runde 1"
#  - runde_nr       : selve runde-nummeret (int) udtrukket fra header
#  - ugedag_raw     : tekst som står i første kolonne (mandag/tirsdag/ el.lign.)
#  - dato_tid_raw   : ren tekst fra 2. kolonne (fx "22/07 15:30")
```

```r
#  - hold_raw       : tekst fra 3. kolonne (fx "BIF-AGF") - vi splitter IKKE her
#  - resultat_raw   : tekst/score fra 4. kolonne (fx "2-1") - ingen split til mål
#  - tilskuertal_raw: rå tekst fra 5. kolonne (fx "12.345") - ingen konvertering
#  - dommer_raw     : dommertekst fra 6. kolonne
#  - tv_kanal_raw   : alt-tekst på TV-logoet, hvis det findes - ellers NA

parse_runde_tabel_raw <- function(tbl_node, season_value) {

  # ------------------------------------------------------------------------
  # Læs runde-overskriften fra thead
  # ------------------------------------------------------------------------
  runde_header <- tbl_node %>%
    html_element("thead tr th") %>%
    html_text2() %>%
    str_squish()

  # Udtræk runde-nummer som heltal, fx "Runde 1" → 1
  runde_nr <- str_extract(runde_header, "\\d+") %>% as.integer()

  # ------------------------------------------------------------------------
  # Find alle rækker i tbody
  # ------------------------------------------------------------------------
  rows <- tbl_node %>% html_elements("tbody tr")
  if (length(rows) == 0) return(tibble())

  # ------------------------------------------------------------------------
  # Map over hver række og træk rå værdier ud
  # ------------------------------------------------------------------------
  map_dfr(rows, function(row) {

    # Alle celler i rækken
    tds <- row %>% html_elements("td")

    # Vi forventer mindst 6 kolonner (ugedag, dato/tid, hold, resultat,
    # tilskuertal, dommer). TV-kolonne er typisk nr. 7 og kan mangle.
    if (length(tds) < 6) return(tibble())

    # Rå tekster, direkte fra HTML-tabellen, kun trimmet for whitespace
    ugedag_raw   <- html_text2(tds[1]) %>% str_squish()
    dato_tid_raw <- html_text2(tds[2]) %>% str_squish()
    hold_raw     <- html_text2(tds[3]) %>% str_squish()
```

```
    # Resultat ligger ofte i en <a> inde i 4. kolonne
    res_node <- tds[4] %>% html_element("a")
    resultat_raw <- if (!is.null(res_node)) {
      res_node %>% html_text2() %>% str_squish()
    } else {
      html_text2(tds[4]) %>% str_squish()
    }

    tilskuertal_raw <- html_text2(tds[5]) %>% str_squish()
    dommer_raw      <- html_text2(tds[6]) %>% str_squish()

    # TV-kanal (kan mangle) - vi gemmer alt-teksten på billedet som RAW
    tv_node <- tryCatch(
      tds[7] %>% html_element("img"),
      error = function(e) NULL
    )

    tv_kanal_raw <- if (!is.null(tv_node)) {
      html_attr(tv_node, "alt") %>% str_squish()
    } else {
      NA_character_
    }

    # Returner én række som tibble - 100 % RAW + let strukturering
    tibble(
      season_value    = season_value,   # parameteren vi scraper sæsonen med
      runde_header    = runde_header,   # fuld header-tekst
      runde_nr        = runde_nr,       # ekstra hjælp senere, men stadig direkte afledt
      ugedag_raw      = ugedag_raw,
      dato_tid_raw    = dato_tid_raw,
      hold_raw        = hold_raw,
      resultat_raw    = resultat_raw,
      tilskuertal_raw = tilskuertal_raw,
      dommer_raw      = dommer_raw,
      tv_kanal_raw    = tv_kanal_raw
    )
  })
}
```

1.1.1.3 Scrape af én sæson i RAW-format

Denne funktion henter kampprogrammet for en given sæson fra SuperStats, identificerer alle

runde-tabeller på siden og parser dem systematisk til et samlet RAW-datasæt. Udvælgelsen sker udelukkende på baggrund af tabelheaders, så strukturen følger kilden tæt uden fortolkning.

```
# ======================================================================
# Scrape én sæson (RAW)
# ======================================================================
# Denne funktion:
#  - bygger URL til en bestemt sæson,
#  - finder alle tabeller,
#  - vælger dem, hvis headeren matcher "Runde X",
#  - parser hver runde-tabel via parse_runde_tabel_raw.

scrape_superstats_saeson_raw <- function(season_value) {

  url <- paste0("https://superstats.dk/program?season=", season_value)
  cat("Henter sæson", season_value, "fra", url, "\n")

  pg <- read_html(url)

  # Find alle <table>-elementer på siden
  alle_tabeller <- pg %>% html_elements("table")

  # Udvælg kun dem, hvor første <th> i thead starter med "Runde <tal>"
  er_runde <- vapply(
    alle_tabeller,
    function(tb) {
      head_node <- tb %>% html_element("thead tr th")

      # Hvis der ikke er noget header-node, er det ikke en runde-tabel
      if (is.null(head_node) || length(head_node) == 0) return(FALSE)

      txt <- tryCatch(
        head_node %>% html_text2() %>% str_squish(),
        error = function(e) ""
      )

      str_detect(txt, "^Runde\\s+\\d+")
    },
    logical(1)
  )

  runde_tabeller <- alle_tabeller[er_runde]
```

```
  if (length(runde_tabeller) == 0) {
    cat("Ingen runde-tabeller fundet for sæson", season_value, "\n")
    return(tibble())
  }

  # Parse alle runde-tabeller for denne sæson og bind dem sammen
  map_dfr(runde_tabeller, ~parse_runde_tabel_raw(.x, season_value))
}
```

1.1.1.4 Samling af RAW-data på tværs af sæsoner

I dette trin køres scrape-funktionen for alle valgte sæsoner, hvorefter resultaterne samles i ét samlet RAW-datasæt. Afslutningsvis foretages et simpelt struktur- og indholdstjek for at validere, at data er indlæst korrekt.

```
# ======================================================================
# Loop over flere sæsoner → samlet RAW-datasæt
# ======================================================================

# Her vælger du selv hvilke sæsoner, du vil have med.
# Eksempel: 2000-2026 som i din oprindelige kode.
saesoner_vec <- 2000:2026

superliga_program_raw <- map_dfr(saesoner_vec, scrape_superstats_saeson_raw)

cat("\nSamlet antal rækker i superliga_program_raw:", nrow(superliga_program_raw), "\n\n")

# Hurtig struktur- og indholdstjek i R:
# glimpse(superliga_program_raw)
str(superliga_program_raw)
```

```
> str(superliga_program_raw)
tibble [5,446 × 10] (S3: tbl_df/tbl/data.frame)
 $ season_value   : int [1:5446] 2000 2000 2000 2000 2000 2000 2000 2000 2000 2000 ...
 $ runde_header   : chr [1:5446] "Runde 1" "Runde 1" "Runde 1" "Runde 1" ...
 $ runde_nr       : int [1:5446] 1 1 1 1 1 1 2 2 2 2 ...
 $ ugedag_raw     : chr [1:5446] "LÃ¸r" "SÃ¸n" "SÃ¸n" "SÃ¸n" ...
 $ dato_tid_raw   : chr [1:5446] "24/07 19:00" "25/07 15:00" "25/07 15:00" "25/07 15:00" ...
 $ hold_raw       : chr [1:5446] "BIF-LBK" "AB-AGF" "HER-AaB" "VB-SIF" ...
 $ resultat_raw   : chr [1:5446] "1-0" "1-0" "3-1" "0-4" ...
 $ tilskuertal_raw: chr [1:5446] "9.114" "2.553" "1.908" "3.141" ...
 $ dommer_raw     : chr [1:5446] "Lars Gerner" "Steen Knudsen" "Knud Stadsgaard" "JÃ¸rn West Larsen" ...
 $ tv_kanal_raw   : chr [1:5446] NA NA NA NA ...
```

1.1.1.5 Upload af kampprogram til RAW-laget i Azure SQL

Her uploades det samlede RAW-scrape til Azure SQL som en faktatabel i schemaet PBA01_Raw, så data lander centralt og kan genbruges i de efterfølgende lag. Uploaden sker med fuld overskrivning for at sikre en konsistent version, og der kan efterfølgende laves et hurtigt sanity check direkte fra databasen.

```
# Drift-/ETL-kode (køres ikke ved render)

# ======================================================================
# Upload til Azure som RAW-fact (PBA01_Raw)
# ======================================================================
# Nu hvor vi har et rent RAW-scrape i superliga_program_raw, skal det op som
# en RAW-tabel i Azure SQL. Det passer ind i din lagdeling:
#  - PBA01_Raw     : landings-/RAW-lag (ingen business-logik, minimal parsing)
#  - PBA02_Clean   : rensning, parsing af datoer, split af hold, mål osv.
#  - PBA03_JoinReady / marts-lag: join med billetdata, vejr, dim_dato, osv.
#
# Vi bruger samme connection-setup som i de andre scripts, men skriver eksplicit
# til schema = "PBA01_Raw".

library(DBI)
library(odbc)

con_azure <- dbConnect(
  odbc::odbc(),
  driver   = "ODBC Driver 18 for SQL Server",
  server   = Sys.getenv("AZURE_SQL_SERVER"),
  database = Sys.getenv("AZURE_SQL_DB"),
  uid      = Sys.getenv("AZURE_SQL_UID"),
  pwd      = Sys.getenv("AZURE_SQL_PWD"),
  port     = 1433,
  Encrypt  = "yes",
  TrustServerCertificate = "no",
  ConnectionTimeout      = 30
)

cat("Forbundet til Azure (", Sys.getenv("AZURE_SQL_DB"), ") - skriver til PBA01_Raw\n\n", se

# Skriv/overskriv RAW-tabellen
DBI::dbWriteTable(
  conn      = con_azure,
  name      = DBI::Id(schema = "PBA01_Raw", table = "fact_superliga_program_raw"),
```

```
  value     = superliga_program_raw,
  overwrite = TRUE
)

cat("Tabel PBA01_Raw.fact_superliga_program_raw er nu opdateret.\n")

# (valgfrit) lille sanity check fra databasen
head_db <- DBI::dbReadTable(
  con_azure,
  DBI::Id(schema = "PBA01_Raw", table = "fact_superliga_program_raw")
)
print(head_db)

dbDisconnect(con_azure)
cat("  Forbindelse til Azure lukket.\n")
```

1.1.2 Webscrapping VFF rundeplacering samlet pipeline

Dette script afviger bevidst fra vores normale lagdeling, fordi vi her har prioriteret en enkel og driftssikker "én-knaps"-pipeline. Det scraper rundeplaceringer, gemmer dem i RAW (overwrite), transformer dem til CLEAN-format (overwrite) og bygger til sidst en kamp-nøglebaseret feature-tabel til PBA03_JoinReady, så den kan joines direkte på baseline via kamp_id.

1.1.2.1 Pakker

```
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(
    rvest, dplyr, tidyr, stringr, purrr, tibble,
    DBI, odbc, rstudioapi
  )
});cat("Pakker klar og loaded \n")
```

1.1.2.2 Safeguard .Renviron (login-check)

Afsnittet validerer, at alle nødvendige Azure SQL–miljøvariabler findes, før scriptet fortsætter. Hvis noget mangler, stoppes kørslen tidligt, så vi undgår fejlkørsler og uklare forbindelsesfejl senere.

```r
# Drift-/ETL-kode (køres ikke ved render)

# ======================================================================
# Safeguard .Renviron
# ======================================================================
required_vars <- c("AZURE_SQL_SERVER", "AZURE_SQL_DB", "AZURE_SQL_UID", "AZURE_SQL_PWD")
values <- Sys.getenv(required_vars)
missing_vars <- required_vars[values == ""]

if (length(missing_vars) > 0) {
  cat("  Følgende miljøvariabler mangler i .Renviron:\n")
  print(missing_vars)
  cat("\nÅbner ~/.Renviron - udfyld værdierne og gem filen.\n")
  file.edit("~/.Renviron")
  if (rstudioapi::isAvailable()) rstudioapi::restartSession()
  stop("Stopper her. Genstart R og kør scriptet igen.")
} else {
  cat("  Alle nødvendige .Renviron-variabler er sat.\n\n")
}
```

1.1.2.3 Funktion: hent VFF's rundeplaceringer pr. sæson

Denne funktion scraper SuperStats for én sæson og finder den tabelrække, der matcher "VFF". Output gemmes i et "wide" RAW-format (Runde …Runde) med sæsonmetadata, så vi senere kan omsætte det til CLEAN/FEATURE uden at gen-scrape.

```r
# ======================================================================
# Funktion: hent VFF placering pr. runde for én sæson (RAW wide)
# ======================================================================
hent_vff_alle_runder <- function(aarstal) {

  url <- paste0(
    "https://superstats.dk/stilling/pladser-runde?season=",
    aarstal,
    "&klub=vff"
  )

  cat("  Scraper season_code:", aarstal, "- URL:", url, "\n")

  tryCatch({

    webpage <- read_html(url)
```

```r
  # 2A) læs sæson-label fra <h2>
  h2_node <- webpage %>% html_element("h2")
  if (length(h2_node) == 0) {
    cat("    Ingen <h2> fundet. Springer over.\n")
    return(NULL)
  }

  h2_txt <- h2_node %>% html_text2() %>% str_squish()
  saeson_label <- str_extract(h2_txt, "\\d{4}/\\d{4}")
  saeson_url   <- if (!is.na(saeson_label)) str_replace(saeson_label, "/", "%2F") else NA_c

  if (is.na(saeson_label)) {
    cat("    Kunne ikke udlede sæson-label. Springer over.\n")
    return(NULL)
  }

  # 2B) læs alle tabeller
  tabeller <- html_table(webpage, header = FALSE, fill = TRUE)
  if (length(tabeller) == 0) {
    cat("    Ingen tabeller fundet. Springer over.\n")
    return(NULL)
  }

  # 2C) find tabel som indeholder "VFF"
  idx <- which(purrr::map_lgl(
    tabeller,
    function(tbl) {
      cell_values <- tbl %>%
        mutate(across(everything(), as.character)) %>%
        unlist(use.names = FALSE) %>%
        str_squish()
      any(cell_values == "VFF")
    }
  ))

  if (length(idx) == 0) {
    cat("    Ingen tabel med VFF (VFF ikke i ligaen?). Springer over.\n")
    return(NULL)
  }

  data <- tabeller[[idx[1]]]
  first_col_name <- names(data)[1]
```

```
    vff_row <- data %>%
      mutate(Klub = str_squish(as.character(.data[[first_col_name]]))) %>%
      filter(Klub == "VFF")

    if (nrow(vff_row) == 0) {
      cat("    VFF-række ikke fundet i valgt tabel. Springer over.\n")
      return(NULL)
    }

    # 2D) fjern klub-kolonne, cast til numeric
    vff_round_data <- vff_row[, -1, drop = FALSE] %>%
      mutate(across(everything(), ~ as.numeric(str_squish(as.character(.x)))))

    # 2E) navngiv Runde_1..Runde_n
    new_names <- paste0("Runde_", seq_len(ncol(vff_round_data)))
    colnames(vff_round_data) <- new_names

    # 2F) add sæson-info
    out <- vff_round_data %>%
      mutate(
        SeasonCode         = aarstal,
        Saeson_label       = saeson_label,
        Saeson_url_encoded = saeson_url
      ) %>%
      relocate(SeasonCode, Saeson_label, Saeson_url_encoded)

    cat("    OK:", saeson_label, "(runder:", ncol(vff_round_data), ")\n")
    out

  }, error = function(e) {
    cat("    Fejl i season_code", aarstal, ":", e$message, "\n")
    NULL
  })
}
```

1.1.2.4 Scrape alle sæsoner til ét samlet RAW-datasæt

Her looper vi alle sæsonkoder (2000–2026) og kalder scrape-funktionen for hver sæson, hvorefter resultaterne bindes sammen til én RAW wide-tabel. Til sidst logger vi antal rækker/kolonner og stopper scriptet, hvis der ikke blev hentet data (fail-fast).

```r
# ======================================================================
# Scrape alle sæsoner
# ======================================================================
season_koder <- 2000:2026

cat("\n--- Starter scraping af VFF placering pr. runde ---\n")

vff_runde_placering_raw_wide <- season_koder %>%
  map(hent_vff_alle_runder) %>%
  list_rbind()

cat("\n--- Færdig med scraping ---\n")
cat("  RAW wide rækker:", nrow(vff_runde_placering_raw_wide), "\n")
cat("  RAW wide kolonner:", ncol(vff_runde_placering_raw_wide), "\n\n")

if (nrow(vff_runde_placering_raw_wide) == 0) {
  stop("Ingen RAW-data scraped. Stopper her.")
}; str(vff_runde_placering_raw_wide)
```

1.1.2.5 Forbindelse til Azure SQL

Her oprettes én fælles databaseforbindelse, som genbruges i resten af scriptet. Det sikrer en stabil og kontrolleret adgang til Azure SQL under hele eksekveringen.

```r
# Drift-/ETL-kode (køres ikke ved render)

# ======================================================================
# Connect til Azure (én gang)
# ======================================================================
cat("  Forbinder til Azure SQL\n")

con <- DBI::dbConnect(
  odbc::odbc(),
  driver   = "ODBC Driver 18 for SQL Server",
  server   = Sys.getenv("AZURE_SQL_SERVER"),
  database = Sys.getenv("AZURE_SQL_DB"),
  uid      = Sys.getenv("AZURE_SQL_UID"),
  pwd      = Sys.getenv("AZURE_SQL_PWD"),
  port     = 1433,
  Encrypt  = "yes",
  TrustServerCertificate = "no",
  ConnectionTimeout      = 180
```

```
)

cat("  Forbundet til:", Sys.getenv("AZURE_SQL_DB"), "\n\n")
```

1.1.2.6 Overwrite af RAW-laget

Her overskrives RAW-tabellen med det fulde, ny-scrapede datasæt for VFF's rundeplaceringer. Det sikrer, at RAW-laget altid afspejler den aktuelle kilde uden historiske dubletter.

```
# Drift-/ETL-kode (køres ikke ved render)

# ======================================================================
# OVERWRITE RAW i SQL
# ======================================================================
raw_table_id <- DBI::Id(schema = "PBA01_Raw", table = "fact_vff_rundeplaceringer_raw")

cat("  Overwriter RAW:", "[PBA01_Raw].[fact_vff_rundeplaceringer_raw]\n")

DBI::dbWriteTable(
  conn      = con,
  name      = raw_table_id,
  value     = vff_runde_placering_raw_wide,
  overwrite = TRUE
)

cat("  RAW overwrite OK.\n\n")
```

```
> glimpse(vff_runde_placering_raw_wide)
Rows: 17
Columns: 40
$ SeasonCode         <int> 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2014, 20…
$ Saeson_label       <chr> "1999/2000", "2000/2001", "2001/2002", "2002/2003", "2003/2004…
$ Saeson_url_encoded <chr> "1999%2F2000", "2000%2F2001", "2001%2F2002", "2002%2F2003", "2…
$ Runde_1            <dbl> 4, 5, 9, 5, 5, 9, 1, 11, 12, 4, 9, 13, 2, 2, 12, 5, 10
$ Runde_2            <dbl> 7, 4, 7, 9, 8, 12, 1, 11, 12, 6, 9, 13, 2, 6, 10, 10, 11
$ Runde_3            <dbl> 6, 1, 8, 11, 10, 7, 1, 10, 12, 9, 6, 11, 5, 4, 7, 9, 8
$ Runde_4            <dbl> 6, 2, 7, 12, 10, 4, 1, 9, 12, 9, 6, 9, 9, 7, 6, 10, 6
$ Runde_5            <dbl> 5, 1, 10, 12, 7, 6, 1, 11, 12, 7, 8, 10, 9, 5, 6, 10, 8
```

1.1.2.7 Byg CLEAN-laget (wide → long + kamp_id)

Her transformerer vi RAW-tabellen fra bredt format til langt format, så hver række repræsenterer én sæson og én runde. Samtidig konstruerer vi kamp_id og beregner placering før kamp,

placering efter runde samt ændringen, og til sidst laver vi et safeguard der stopper scriptet, hvis kamp_id ikke er unik.

```
# Drift-/ETL-kode (køres ikke ved render)

# ======================================================================
# Byg CLEAN (long + kamp_id + før/efter + ændring)
# ======================================================================
VFF_KLUB_ID <- "KLUB046"

cat("--- Starter CLEAN (wide → long) ---\n")

vff_runde_placering_clean <- vff_runde_placering_raw_wide %>%
  select(-any_of("Saeson_url_encoded")) %>%
  pivot_longer(
    cols = starts_with("Runde_"),
    names_to  = "runde",
    values_to = "placering",
    values_drop_na = TRUE
  ) %>%
  mutate(
    runde     = as.integer(str_remove(as.character(runde), "Runde_")),
    placering = as.integer(placering),
    sæsonkode = as.integer(SeasonCode),
    sæson     = str_squish(as.character(Saeson_label)),
    kamp_id   = paste0(sæson, "R", runde, VFF_KLUB_ID)
  ) %>%
  arrange(sæsonkode, runde) %>%
  group_by(sæsonkode) %>%
  mutate(
    placering_efter_runde = placering,
    placering_før_kamp    = lag(placering, 1),
    ændring_placering     = placering_efter_runde - placering_før_kamp
  ) %>%
  ungroup() %>%
  select(
    kamp_id, sæsonkode, sæson, runde,
    placering_før_kamp, placering_efter_runde, ændring_placering
  )

# safeguard: kamp_id unik
dup <- vff_runde_placering_clean %>%
  count(kamp_id) %>%
```

```
  filter(n > 1)

if (nrow(dup) > 0) {
  print(dup)
  DBI::dbDisconnect(con)
  stop("  kamp_id er ikke unik i CLEAN. Stop.")
}

cat("  CLEAN bygget. Rækker:", nrow(vff_runde_placering_clean), "\n\n"); vff_runde_placering_
```

```
> vff_runde_placering_clean
# A tibble: 545 × 7
   kamp_id sæsonkode sæson runde placering_før_kamp placering_efter_runde ændring_placering
   <chr>       <int> <chr> <int>              <int>                 <int>             <int>
 1 1999/2…      2000 1999…     1                 NA                     4                NA
 2 1999/2…      2000 1999…     2                  4                     7                 3
 3 1999/2…      2000 1999…     3                  7                     6                -1
 4 1999/2…      2000 1999…     4                  6                     6                 0
 5 1999/2…      2000 1999…     5                  6                     5                -1
 6 1999/2…      2000 1999…     6                  5                     5                 0
 7 1999/2…      2000 1999…     7                  5                     3                -2
 8 1999/2…      2000 1999…     8                  3                     2                -1
 9 1999/2…      2000 1999…     9                  2                     4                 2
10 1999/2…      2000 1999…    10                  4                     5                 1
# i 535 more rows
# i Use `print(n = ...)` to see more rows
```

1.1.2.8 Overwrite CLEAN-laget i databasen

Det færdige CLEAN-datasæt skrives nu til PBA02_Clean og erstatter eventuelle tidligere versioner. På den måde sikres, at efterfølgende joins og analyser altid arbejder på én konsistent og opdateret sandhedskilde.

```
# Drift-/ETL-kode (køres ikke ved render)

# ======================================================================
# 7) OVERWRITE CLEAN i SQL
# ======================================================================
clean_table_id <- DBI::Id(schema = "PBA02_Clean", table = "fact_vff_rundeplaceringer_clean")

cat("  Overwriter CLEAN:", "[PBA02_Clean].[fact_vff_rundeplaceringer_clean]\n")

DBI::dbWriteTable(
  conn      = con,
```

```
  name      = clean_table_id,
  value     = vff_runde_placering_clean,
  overwrite = TRUE
)

cat("  CLEAN overwrite OK.\n\n")
```

1.1.2.9 Feature-tabel til modellering

Her kondenseres CLEAN-data til én række pr. kamp via kamp_id og gemmes som en feature-tabel. Denne tabel anvendes direkte i baseline- og analyse-datasæt og gør joinet mod modeller enkelt og stabilt.

```
# Drift-/ETL-kode (køres ikke ved render)


# ======================================================================
# 8) FEATURE-tabel til baseline (én række pr kamp_id)
#    (Den her er det du joiner ind i analysis_df/baseline på kamp_id)
# ======================================================================
feat_tbl <- vff_runde_placering_clean %>%
  transmute(
    kamp_id,
    vff_placering_før_kamp = placering_før_kamp,
    vff_placering_efter_runde = placering_efter_runde,
    vff_ændring_placering = ændring_placering
  )

feat_table_id <- DBI::Id(schema = "PBA03_JoinReady", table = "feat_vff_rundeplacering_kamp")

cat("  Overwriter FEATURE:", "[PBA03_JoinReady].[feat_vff_rundeplacering_kamp]\n")

DBI::dbWriteTable(
  conn      = con,
  name      = feat_table_id,
  value     = feat_tbl,
  overwrite = TRUE
); tibble(feat_tbl)

cat("  FEATURE overwrite OK.\n\n")
```

```
> tibble(feat_tbl)
# A tibble: 545 × 4
   kamp_id             vff_placering_før_kamp vff_placering_efter_r…¹ vff_ændring_placering
   <chr>                                <int>                   <int>                  <int>
 1 1999/2000R1KLUB046                      NA                       4                     NA
 2 1999/2000R2KLUB046                       4                       7                      3
 3 1999/2000R3KLUB046                       7                       6                     -1
 4 1999/2000R4KLUB046                       6                       6                      0
 5 1999/2000R5KLUB046                       6                       5                     -1
 6 1999/2000R6KLUB046                       5                       5                      0
 7 1999/2000R7KLUB046                       5                       3                     -2
 8 1999/2000R8KLUB046                       3                       2                     -1
 9 1999/2000R9KLUB046                       2                       4                      2
10 1999/2000R10KLUB046                      4                       5                      1
# i 535 more rows
# i abbreviated name: ¹vff_placering_efter_runde
# i Use `print(n = ...)` to see more rows
```

1.1.2.10 Luk forbindelse

Forbindelsen til Azure SQL lukkes kontrolleret for at sikre, at ressourcer frigives korrekt, og at scriptet afsluttes pænt.

```
# Drift-/ETL-kode (køres ikke ved render)

DBI::dbDisconnect(con)
cat("  Azure connection lukket. Done.\n")
```

1.1.3 VFF interne data (3 datasæt)

I dette script indlæser vi VFF's interne data fra både en lokal SQLite-database og to RDS-filer, laver et kort kvalitetstjek (klubnavne + kolonnematch), og uploader derefter tre separate RAW-tabeller til Azure SQL i PBA01_Raw.

Det giver os en stabil "landing zone", hvor vi både har klub-dimensionen samt billetsalget fra to kilder (SQLite og RDS), så rensning og senere modellering kan bygges ovenpå et ensartet RAW-lag.

1.1.3.1 Pakker

Her indlæses de nødvendige R-pakker til databaseadgang, datahåndtering og upload til Azure SQL. Pacman bruges for at sikre, at alle pakker både er installeret og loadet konsistent.

```
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(DBI, RSQLite, dplyr, tidyverse, odbc, rstudioapi)
}); cat("Pakker klar og loaded \n")
```

1.1.3.2 Filstier

Her defineres de lokale filstier til de datakilder, som indlæses i scriptet.
Stierne peger på både SQLite-databasen og de tilhørende RDS-filer, som tilsammen udgør grundlaget for RAW-uploadet.

```
{
sqlite_path <- "C:/Users/janpe/OneDrive/Skrivebord/PBA Dataanlyse/01_Første semester/1 Semest

rds_klubber_path <- "C:/Users/janpe/OneDrive/Skrivebord/PBA Dataanlyse/01_Første semester/1 S

rds_billetsalg_path <- "C:/Users/janpe/OneDrive/Skrivebord/PBA Dataanlyse/01_Første semester/
cat("Alt VFF data fundet  \n")
```

1.1.3.3 Hent data fra SQLite

I dette afsnit oprettes forbindelse til den lokale SQLite-database, hvorefter klub- og billetsalgsdata indlæses som rå datasæt. Formålet er at få et første overblik over struktur og indhold, inden data senere uploades til RAW-laget i Azure SQL.

```
# Hent data fra SQLite
# ----------------------------------------------------------------------
con_sqlite <- DBI::dbConnect(RSQLite::SQLite(), sqlite_path)

tabeller <- DBI::dbListTables(con_sqlite)
cat("Tabeller i sqlite:\n")
print(tabeller)

# tabeller[1] = klubber, tabeller[2] = billetsalg
dim_vff_klubber     <- DBI::dbReadTable(con_sqlite, tabeller[1])
fact_vff_billetsalg <- DBI::dbReadTable(con_sqlite, tabeller[2])

DBI::dbDisconnect(con_sqlite)

# Overblik + valgfri view
{
```

```
  glimpse(fact_vff_billetsalg); str(fact_vff_billetsalg)
  glimpse(dim_vff_klubber);     str(dim_vff_klubber)

  if (interactive()) {
    View(fact_vff_billetsalg)
    View(dim_vff_klubber)
  }
}
```

```
> glimpse(fact_vff_billetsalg)
Rows: 259
Columns: 8
$ sæson         <chr> "2000/2001", "2000/2001", "2000/2001", "2000/2001", "2…
$ runde         <dbl> 2, 4, 5, 8, 9, 11, 13, 15, 17, 19, 20, 22, 24, 26, 28,…
$ tilskuere     <dbl> 2089, 3198, 2378, 3878, 2017, 2308, 4132, 2308, 2308, …
$ år            <dbl> 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, …
$ hold          <chr> "VFF- AB", "VFF-FCK", "VFF-SJF", "VFF-FCM", "VFF-SIF",…
$ d10_tilskuere <dbl> 1087, 1346, 1114, 1871, 949, 1026, 2181, 1027, 1278, 9…
$ d7_tilskuere  <dbl> 1282, 2147, 1607, 2517, 1103, 1451, 2810, 1543, 1437, …
$ d3_tilskuere  <dbl> 1898, 2735, 2166, 3278, 1769, 1994, 3992, 2049, 1882, …
>
```

```
> glimpse(dim_vff_klubber)
Rows: 47
Columns: 6
$ kort               <chr> "AB", "ACH", "AGF", "AHF", "AMA", "B.93", "B1909", "B1913", …
$ langt              <chr> "AB", "AC Horsens", "AGF", "Aarhus Fremad", "Fremad Amager",…
$ navn               <chr> "Akademisk Boldklub Gladsaxe", "Alliance Club Horsens", "Aar…
$ stadion            <chr> "Gladsaxe Stadion", "Horsens Idrætspark)", "Vejlby Stadium",…
$ sponsor_stadion_navn <chr> "Gladsaxe Stadion", "Nordstern Arena Horsens", "Vejlby Stadi…
$ tilskuere          <dbl> 13507, 10495, 12000, 5000, 7200, 4400, 6000, 2000, 12000, 44…
```

1.1.3.4 Hent data fra RDS-filer

Her indlæses supplerende VFF-data fra RDS-filer, som bruges til validering og sammenligning med data fra SQLite. Formålet er at sikre konsistens i klubnavne og struktur, før datasættene uploades som RAW-tabeller.

```
RDSfil1 <- readRDS(rds_klubber_path)
df_1    <- as_tibble(RDSfil1)

tjek_ens <- df_1 |>
  mutate(findes_i_dim = ifelse(navn %in% dim_vff_klubber$navn, "ja", "nej"))
```

```
afvigelser <- tjek_ens |>
  filter(findes_i_dim == "nej")

if (interactive()) {
  View(tjek_ens)
  View(afvigelser)
}

RDSfil2 <- readRDS(rds_billetsalg_path)
df_2    <- as_tibble(RDSfil2)               # billetsalg fra RDS
df_fact <- as_tibble(fact_vff_billetsalg)   # billetsalg fra SQLite

tibble(tjek_ens)
```

kort	langt	navn	stadion	sponsor_stadion_navn	tilskuere	findes_i_dim
AB	AB	Akademisk Boldklub Gladsaxe	Gladsaxe Stadion	Gladsaxe Stadion	13507	ja
ACH	AC Horsens	Alliance Club Horsens	Horsens Idrætspark)	Nordstern Arena Horsens	10495	ja
AGF	AGF	Aarhus Gymnastikforening	Vejlby Stadium	Vejlby Stadium	12000	ja
AHF	Aarhus Fremad	Aarhus Fremad	Risvangen Stadion	Risvangen Stadion	5000	ja
AMA	Fremad Amager	Boldklubben Fremad Amager	Sundby Idrætspark	Sundby Idrætspark	7200	ja
B.93	B.93	Boldklubben af 1893	Østerbro Stadion	Østerbro Stadion	4400	ja
B1909	B1909	Boldklubben 1909	Gillested Park	Gillested Park	6000	ja
B1913	B1913	Boldklubben 1913	Campus Road	Campus Road	2000	ja

1.1.3.5 Struktur- og kolonnekontrol

Her sammenlignes kolonnerne i billetsalgsdata fra RDS og SQLite for at identificere strukturelle forskelle. Formålet er at afdække afvigelser tidligt, så RAW-tabellerne dokumenterer kilden korrekt og konsistent.

```
# ------------------------------------------------------------------
# Sammenligning af kolonner (struktur-tjek)
# ------------------------------------------------------------------
kolonner_kun_i_df2  <- setdiff(names(df_2),    names(df_fact))
kolonner_kun_i_fact <- setdiff(names(df_fact), names(df_2))

message("Kolonner i df_2 (RDS) som IKKE er i fact_vff_billetsalg (SQLite):")
print(kolonner_kun_i_df2)

message("Kolonner i fact_vff_billetsalg (SQLite) som IKKE er i df_2 (RDS):")
print(kolonner_kun_i_fact)
```

1.1.3.6 Miljø-sikring (.Renviron)

Dette afsnit sikrer, at alle nødvendige login-oplysninger til Azure SQL er sat korrekt. Hvis noget mangler, stoppes scriptet tidligt for at undgå fejlslagen upload eller ufuldstændige RAW-data.

```
# Drift-/ETL-kode (køres ikke ved render)

# ------------------------------------------------------------------------
# Safeguard: tjek at .Renviron er sat korrekt op
# ------------------------------------------------------------------------
required_vars <- c("AZURE_SQL_SERVER", "AZURE_SQL_DB", "AZURE_SQL_UID", "AZURE_SQL_PWD")
values <- Sys.getenv(required_vars)
missing_vars <- required_vars[values == ""]

if (length(missing_vars) > 0) {
  cat("   Følgende miljøvariabler mangler i .Renviron:\n")
  print(missing_vars)
  cat("\nÅbner ~/.Renviron - udfyld værdierne og gem filen.\n")

  file.edit("~/.Renviron")
  cat("  R genstartes nu via RStudio - kør scriptet igen bagefter.\n")

  if (rstudioapi::isAvailable()) {
    rstudioapi::restartSession()
  } else {
    stop("rstudioapi er ikke tilgængelig - genstart R manuelt og kør scriptet igen.")
  }
} else {
  cat("  Alle nødvendige .Renviron-variabler er sat.\n\n")
}
```

1.1.3.7 Azure-forbindelse med retry

Denne del opretter forbindelse til Azure SQL med automatisk genforsøg og stigende timeout. Formålet er at sikre en stabil forbindelse, selv ved midlertidige netværks- eller serverproblemer.

```
# Drift-/ETL-kode (køres ikke ved render)

# ------------------------------------------------------------------------
# Azure forbindelse med automatisk retry
# ------------------------------------------------------------------------
forsøg_max <- 6
timeouts  <- c(60, 180, 200, 260, 360, 600)
delay_sec <- 10
forsøg    <- 1
con_azure <- NULL
```

```
while (forsøg <= forsøg_max && is.null(con_azure)) {

  timeout_brug <- timeouts[min(forsøg, length(timeouts))]

  cat("Forsøg", forsøg, "på at forbinde (ConnectionTimeout =", timeout_brug, "sekunder)\n")

  con_try <- try(
    DBI::dbConnect(
      odbc::odbc(),
      driver   = "ODBC Driver 18 for SQL Server",
      server   = Sys.getenv("AZURE_SQL_SERVER"),
      database = Sys.getenv("AZURE_SQL_DB"),
      uid      = Sys.getenv("AZURE_SQL_UID"),
      pwd      = Sys.getenv("AZURE_SQL_PWD"),
      port     = 1433,
      Encrypt  = "yes",
      TrustServerCertificate = "no",
      ConnectionTimeout      = timeout_brug
    ),
    silent = TRUE
  )

  if (!inherits(con_try, "try-error")) {
    con_azure <- con_try
    cat("  Forbundet til:", Sys.getenv("AZURE_SQL_DB"), "på forsøg", forsøg, "\n\n")
  } else {
    cat("  Forbindelsen fejlede på forsøg", forsøg, "\n")

    if (forsøg == forsøg_max) {
      stop("Kunne ikke forbinde til Azure SQL efter ", forsøg_max, " forsøg.\n")
    }

    cat("Venter", delay_sec, "sekunder før næste forsøg\n\n")
    Sys.sleep(delay_sec)
    forsøg <- forsøg + 1
  }
}
```

1.1.3.8 Upload af data til Azure SQL database

I dette afsnit uploades de tre datasæt til RAW-laget i Azure SQL under schemaet PBA01_Raw.

Tabellerne overskrives bevidst, da de fungerer som rene landings-/kildetabeller, som senere danner grundlag for clean- og join-ready lagene.

```
# Drift-/ETL-kode (køres ikke ved render)

# ------------------------------------------------------------------
# Upload til Azure SQL (skema PBA01_Raw)
# ------------------------------------------------------------------
schema <- "PBA01_Raw"

DBI::dbWriteTable(
  conn      = con_azure,
  name      = DBI::Id(schema = schema, table = "dim_vff_klubber_raw"),
  value     = dim_vff_klubber,
  overwrite = TRUE
)

DBI::dbWriteTable(
  conn      = con_azure,
  name      = DBI::Id(schema = schema, table = "fact_vff_billetsalg_raw_SQLite"),
  value     = df_fact,
  overwrite = TRUE
)

DBI::dbWriteTable(
  conn      = con_azure,
  name      = DBI::Id(schema = schema, table = "fact_vff_billetsalg_raw_RDS"),
  value     = df_2,
  overwrite = TRUE
)
```

1.1.3.9 Hurtigt sanity check og lukning

Her laver vi et hurtigt opslag i de tre uploadede RAW-tabeller for at bekræfte, at data faktisk ligger i Azure som forventet.
Til sidst lukkes forbindelsen til databasen, så scriptet afslutter pænt og uden åbne sessions.

```
# Drift-/ETL-kode (køres ikke ved render)


head(DBI::dbReadTable(con_azure, DBI::Id(schema = schema, table = "dim_vff_klubber_raw")))
head(DBI::dbReadTable(con_azure, DBI::Id(schema = schema, table = "fact_vff_billetsalg_raw_S
```

```r
head(DBI::dbReadTable(con_azure, DBI::Id(schema = schema, table = "fact_vff_billetsalg_raw_RI

DBI::dbDisconnect(con_azure)
```

1.1.4 Websrapping Håndbold kampprogram SAH(tophaandbold.dk)

I dette afsnit indsamles kampprogrammet for **SAH**, som er det håndboldhold, der vurderes mest relevant i relation til Viborg og projektets kontekst. Formålet er at kortlægge, hvornår SAH afvikler hjemmekampe, da disse kan fungere som en konkurrerende begivenhed i forhold til publikums tilstedeværelse. Kampprogrammet bruges som et kontekstuelt datasæt, der kan indgå i analysen af tilskuertal.

1.1.4.1 Pakker

```r
# ======================================================================
# Pakker
# ======================================================================
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(
    rvest, httr, dplyr, stringr, purrr, tibble, tidyr,
    DBI, odbc, rstudioapi, dbplyr, rlang
  )
}); cat("Pakker loaded og klar til anvendelse \n")
```

1.1.4.2 Hjælpefunktion – opbygning af kampprogram-URL

Denne funktion konstruerer en korrekt forespørgsels-URL til tophåndbold.dk baseret på sæson, hold og hjemme/udekamp.
Formålet er at sikre ensartede og reproducerbare kald til kampprogrammet i det efterfølgende scrape.

```r
# ======================================================================
# Hjælpefunktion - byg URL til kampprogram
# ======================================================================
byg_kampprogram_url <- function(saeson_id,
                                hold_id,
                                hjemmekamp = 1,
                                udekamp    = 0) {
  grund_url <- "https://tophaandbold.dk/kampprogram/herreligaen"
```

```
  forespoergsel <- list(
    year      = saeson_id,
    team      = hold_id,
    home_game = hjemmekamp,
    away_game = udekamp
  )

  query_streng <- paste0(
    names(forespoergsel), "=", forespoergsel,
    collapse = "&"
  )

  paste0(grund_url, "?", query_streng)
}
```

1.1.4.3 Hjælpefunktion – parsing af kamprækker fra HTML

Denne funktion udtrækker rå kampinformation direkte fra HTML-strukturen på kampprogramsiden. Formålet er at omsætte ustruktureret webindhold til et konsistent RAW-datasæt uden fortolkning eller berigelse.

```
# ======================================================================
# Hjælpefunktion - parse kamp-rækker fra HTML
# ======================================================================
parse_kamprækker <- function(side_html) {

  kamprækker <- side_html |>
    html_elements(".match-program .row")

  if (length(kamprækker) == 0) {
    message("Ingen kamp-rækker fundet på siden.")
    return(tibble())
  }

  resultat <- purrr::map_dfr(kamprækker, function(række) {

    dato_raw <- række |>
      html_element(".match-program__row__date") |>
      html_text2()

    if (length(dato_raw) == 0 ||
```

```
        is.na(dato_raw) ||
        str_squish(dato_raw) == "") {
      return(tibble())
    }
    dato_raw <- dato_raw |> str_squish()

    hold_tekster <- række |>
      html_elements(".match-program__row__team") |>
      html_text2() |>
      str_squish()

    if (length(hold_tekster) < 2) {
      return(tibble())
    }

    hjemmehold_tekst <- hold_tekster[1]
    udehold_tekst    <- hold_tekster[2]

    tibble(
      kamp_dato_raw = dato_raw,
      hjemmehold    = hjemmehold_tekst,
      udehold       = udehold_tekst
    )
  })

  resultat
}
```

1.1.4.4 Hjælpefunktion – hent kampprogram for én sæson

Funktionen henter kampprogrammet for en given sæson og et specifikt hold via et HTTP-kald. HTML-svaret parses efterfølgende til et RAW-datasæt ved hjælp af den dedikerede parse-funktion.

```
# ======================================================================
# Hjælpefunktion - hent én sæsons kampprogram
# ======================================================================
hent_kampprogram_saeson <- function(saeson_id,
                                    hold_id,
                                    saeson_label,
                                    hjemmekamp = 1,
                                    udekamp    = 0) {
```

```
  url <- byg_kampprogram_url(
    saeson_id  = saeson_id,
    hold_id    = hold_id,
    hjemmekamp = hjemmekamp,
    udekamp    = udekamp
  )

  message("Henter: ", url, "  (", saeson_label, ")")

  svar <- httr::GET(url)
  if (status_code(svar) != 200) {
    warning("Fejl ved hentning af URL (status ",
            status_code(svar), "): ", url)
    return(tibble())
  }

  side_html <- svar |>
    content(as = "text") |>
    read_html()

  parse_kamprækker(side_html)
}
```

1.1.4.5 Opsamling af SAH-sæsoner til samlet RAW-datasæt

Her defineres de relevante SAH-sæsoner og tilhørende hold, hvorefter kampprogrammet hentes sæson for sæson.
Resultatet samles i ét konsistent RAW-datasæt, som danner grundlag for videre rensning og analyse.

```
# ======================================================================
# Oversigt over SAH-sæsoner + loop → RAW-datasæt
# ======================================================================
sah_saesoner <- tribble(
  ~saeson_id, ~saeson_label, ~hold_id,
  9,          "2025/2026",    68,
  8,          "2024/2025",    68,
  7,          "2023/2024",    68,
  6,          "2022/2023",    68,
  5,          "2021/2022",    68,
  4,          "2020/2021",    11,
  3,          "2019/2020",    11,
```

```
  2,          "2018/2019",    11,
  1,          "2017/2018",    11
)

sah_kampprogram_raw <- sah_saesoner |>
  mutate(
    kampdata = purrr::pmap(
      list(saeson_id, hold_id, saeson_label),
      ~ hent_kampprogram_saeson(
        saeson_id    = ..1,
        hold_id      = ..2,
        saeson_label = ..3,
        hjemmekamp   = 1,
        udekamp      = 0
      )
    )
  ) |>
  unnest(cols = kampdata)

str(sah_kampprogram_raw)
view(sah_kampprogram_raw)
```

```
> glimpse(sah_kampprogram_raw)
Rows: 135
Columns: 6
$ saeson_id     <dbl> 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 8, 8, 8, 8, 8, 8, 8, 8, 8…
$ saeson_label  <chr> "2025/2026", "2025/2026", "2025/2026", "2025/2026", "2025/2026", "2…
$ hold_id       <dbl> 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68, 68,…
$ kamp_dato_raw <chr> "03-09-25 19:00", "18-09-25 19:00", "04-10-25 16:00", "18-10-25 16:…
$ hjemmehold    <chr> "SAH", "SAH", "SAH", "SAH", "SAH", "SAH", "SAH", "SAH", "SAH", "SAH…
$ udehold       <chr> "Grindsted GIF Håndbold", "HØJ Elite Herrer", "Mors-Thy Håndbold", …
```

1.1.4.6 Miljøvariabler og sikker opsætning

Dette afsnit sikrer, at alle nødvendige login-oplysninger til Azure SQL er korrekt sat via .Renviron. Hvis der mangler variabler, stoppes scriptet tidligt for at undgå fejl og ufuldstændige dataindlæsninger.

```
# Drift-/ETL-kode (køres ikke ved render)

# ======================================================================
# Safeguard .Renviron (.Renviron skal være sat korrekt op)
# ======================================================================
```

```
required_vars <- c(
  "AZURE_SQL_SERVER",
  "AZURE_SQL_DB",
  "AZURE_SQL_UID",
  "AZURE_SQL_PWD"
)

values <- Sys.getenv(required_vars)
missing_vars <- required_vars[values == ""]

if (length(missing_vars) > 0) {
  cat("  Følgende miljøvariabler mangler i .Renviron:\n")
  print(missing_vars)
  cat("\nÅbner ~/.Renviron - udfyld værdierne og gem filen.\n")

  file.edit("~/.Renviron")
  cat("  R genstartes nu via RStudio - kør scriptet igen bagefter.\n")

  if (rstudioapi::isAvailable()) {
    rstudioapi::restartSession()
  } else {
    stop("rstudioapi er ikke tilgængelig - genstart R manuelt og kør scriptet igen.")
  }
} else {
  cat("  Alle nødvendige .Renviron-variabler er sat.\n\n")
}
```

1.1.4.7 Azure SQL-forbindelse med retry-logik

Dette afsnit opretter forbindelse til Azure SQL med automatisk retry og gradvist stigende timeout. Formålet er at gøre dataindlæsningen robust over for midlertidige netværks- eller serverproblemer.

```
# Drift-/ETL-kode (køres ikke ved render)

# ======================================================================
# Azure forbindelse med automatisk retry
# ======================================================================
forsøg_max  <- 6
timeouts    <- c(60, 180, 200, 260, 360, 600)
delay_sec   <- 10
forsøg      <- 1
```

```r
con         <- NULL

while (forsøg <= forsøg_max && is.null(con)) {

  timeout_brug <- timeouts[min(forsøg, length(timeouts))]

  cat(
    "Forsøg", forsøg,
    "på at forbinde (ConnectionTimeout =", timeout_brug, "sekunder)...\n"
  )

  con_try <- try(
    dbConnect(
      odbc::odbc(),
      driver   = "ODBC Driver 18 for SQL Server",
      server   = Sys.getenv("AZURE_SQL_SERVER"),
      database = Sys.getenv("AZURE_SQL_DB"),
      uid      = Sys.getenv("AZURE_SQL_UID"),
      pwd      = Sys.getenv("AZURE_SQL_PWD"),
      port     = 1433,
      Encrypt  = "yes",
      TrustServerCertificate = "no",
      ConnectionTimeout      = timeout_brug
    ),
    silent = TRUE
  )

  if (!inherits(con_try, "try-error")) {
    con <- con_try
    cat(
      "  Forbundet til:", Sys.getenv("AZURE_SQL_DB"),
      "på forsøg", forsøg, "\n\n"
    )
  } else {
    cat("  Forbindelsen fejlede på forsøg", forsøg, "- prøver igen.\n")

    if (forsøg == forsøg_max) {
      stop("Kunne ikke forbinde til Azure SQL efter ", forsøg_max, " forsøg.\n")
    }

    cat("Venter", delay_sec, "sekunder før næste forsøg...\n\n")
    Sys.sleep(delay_sec)
```

```
    forsøg <- forsøg + 1
  }
}
```

1.1.4.8 Første load og inkrementel opdatering af SAH-kampprogram

Dette afsnit håndterer både første indlæsning og efterfølgende inkrementelle opdateringer af kampprogrammet i RAW-laget. Kun nye kampe indsættes ved senere kørsler, så dubletter undgås, og tabellen holdes konsistent.

```
# Drift-/ETL-kode (køres ikke ved render)

# ======================================================================
# Første load + incremental update til PBA01_Raw.fact_kampprogram_SAH_raw
#           (ingen rå SQL, ingen dupletter)
# ======================================================================
table_id <- Id(
  schema = "PBA01_Raw",
  table  = "fact_kampprogram_SAH_raw"
)

cat("  Tjekker adgang til schema 'PBA01_Raw' og tabellen via dbExistsTable...\n")

exists_try <- try(
  dbExistsTable(con, table_id),
  silent = TRUE
)

if (inherits(exists_try, "try-error")) {
  dbDisconnect(con)
  stop(
    "Kan ikke tilgå 'PBA01_Raw.fact_kampprogram_SAH_raw'.\n",
    "Tjek at schemaet 'PBA01_Raw' findes, og at brugeren har rettigheder."
  )
}

tabel_findes <- isTRUE(exists_try)

if (!tabel_findes) {
  # ------------------------------------------------------------------
  # Første load: tabellen findes ikke → opret og indsæt alle rækker
  # ------------------------------------------------------------------
```

```
  cat("  Første load: Tabel findes ikke, opretter og indsætter alle rækker...\n")

  dbWriteTable(
    conn      = con,
    name      = table_id,
    value     = sah_kampprogram_raw,
    overwrite = FALSE,
    append    = FALSE
  )

  antal_rækker  <- nrow(sah_kampprogram_raw)
  antal_værdier <- nrow(sah_kampprogram_raw) * ncol(sah_kampprogram_raw)

  cat("  Første load gennemført.\n")
  cat("  Antal rækker indsat:", antal_rækker, "\n")
  cat("  Antal værdier i tabellen:", antal_værdier, "\n\n")

} else {
  # ------------------------------------------------------------------
  # Incremental update: indsæt kun nye kampe
  # Nøgle = (saeson_id, kamp_dato_raw, hjemmehold, udehold)
  # ------------------------------------------------------------------
  cat("  Tabel findes - laver incremental update.\n")

  fact_tbl <- tbl(
    con,
    in_schema("PBA01_Raw", "fact_kampprogram_SAH_raw")
  )

  # Hent eksisterende nøgler fra SQL
  eksisterende_nøgler <- fact_tbl |>
    select(saeson_id, kamp_dato_raw, hjemmehold, udehold) |>
    distinct() |>
    collect()

  # Find de rækker i det nye scrape, som IKKE findes i SQL-tabellen
  nye_rækker <- sah_kampprogram_raw |>
    anti_join(
      eksisterende_nøgler,
      by = c("saeson_id", "kamp_dato_raw", "hjemmehold", "udehold")
    )
```

```
  if (nrow(nye_rækker) == 0) {
    cat("  Ingen nye kampe at indsætte - alt er allerede i databasen.\n\n")
  } else {
    cat("  Indsætter kun nye kampe (antal):", nrow(nye_rækker), "\n")

    dbWriteTable(
      conn      = con,
      name      = table_id,
      value     = nye_rækker,
      append    = TRUE,
      overwrite = FALSE
    )

    cat("  Incremental insert gennemført.\n\n")
  }
}


# ========================================================================
# Luk forbindelsen pænt
# ========================================================================
dbDisconnect(con)
cat("  Forbindelsen til Azure SQL er nu lukket.\n")
```

1.1.5 API Danske helligdage Raw (Nager.Date)

I dette afsnit hentes officielle danske helligdage via Nager.Date API og samles til én RAW-dimension for perioden 2000–2026. Datasættet afspejler kilden direkte og indeholder kun let formatering/oversættelse, så det senere kan bruges som stabilt input til clean- og join-ready lag.

1.1.5.1 Pakker

```
# Pakker:
#  - httr2/jsonlite  → API-kald og JSON-parsing
#  - tibble/dplyr    → datarammer og transformationer
#  - purrr           → map-funktioner til årsløkke
#  - lubridate       → datohåndtering
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
```

```
  pacman::p_load(
    httr2, jsonlite, tibble, dplyr, purrr, lubridate
  )
}); cat("Pakker klar god arbejdslyst \n")
```

1.1.5.2 Funktion til hentning og standardisering af helligdage

Her defineres en genbrugelig funktion, der henter helligdage for ét år via Nager.Date, validerer API-svaret og returnerer en standardiseret RAW-dimension med fælles nøgler og navnefelter til senere joins.

```
# ------------------------------------------------------------
# Funktion: hent og formater helligdage for ét år
# ------------------------------------------------------------

get_public_holidays_dim <- function(year, country = "DK") {

  base_url <- "https://date.nager.at/api/v3/publicholidays"

  # Byg request og send til API'et
  resp <- request(base_url) |>
    req_url_path_append(paste0(year, "/", country)) |>
    req_perform()

  # Simpel fejlhåndtering: vi stopper scriptet, hvis API svarer med fejl
  if (resp_status(resp) >= 400) {
    stop("API-fejl for år: ", year, " land: ", country)
  }

  # JSON → tekst → liste → tibble
  dat_raw <- resp |>
    resp_body_string() |>
    fromJSON(flatten = TRUE) |>
    as_tibble()

  # Formatering til vores ønskede RAW-struktur
  out <- dat_raw |>
    mutate(
      # API-feltet "date" er tekst - vi laver det om til rigtig Date
      dato_tmp = as.Date(date)
    ) |>
    transmute(
```

```
      # datekey som ddmmåååå, så den matcher dim_dato og andre dimensioner
      datekey        = format(dato_tmp, "%d%m%Y"),

      # Behold R-dato (bruges ofte i analyser)
      dato           = dato_tmp,

      # Navne direkte fra API
      helligdag_navn = localName,   # dansk navn
      english_name   = name,        # engelsk navn

      # Type: oversæt alle API-typer til dansk tekst
      type = purrr::map_chr(
        types,
        \(x) {
          # recode mapper de enkelte kodeord til danske beskrivelser
          x_dk <- dplyr::recode(
            x,
            "Public"      = "Officiel helligdag",
            "Bank"        = "Bankferie",
            "School"      = "Skoleferie",
            "Authorities" = "Myndigheder lukket",
            "Optional"    = "Valgfri fridag",
            "Observance"  = "Højtid (ingen betalt fridag)",
            .default      = x
          )
          # hvis der er flere type-værdier, samles de i én tekststreng
          paste(x_dk, collapse = ", ")
        }
      )
    )

  out
}
```

1.1.5.3 Hent og saml alle helligdage (2000–2026)

Her hentes helligdage for hvert år i perioden 2000–2026 ved hjælp af den definerede funktion, og alle årsresultater bindes sammen til én samlet RAW-dimension, som efterfølgende valideres med simple QA-udskrifter.

```
# ------------------------------------------------------------
# Hent alle år 2000-2026 og bind til én samlet dim-tabel
```

```r
# ------------------------------------------------------------
# years:
#   - definerer den periode, vi vil dække med helligdage.
#   - kan nemt justeres senere (fx 1990:2030).
#
# purrr::map_dfr:
#   - kalder get_public_holidays_dim for hvert år
#   - r binder alle års-tabeller sammen (row-bind) til én samlet tabel.
years <- 2000:2026

dim_helligdage_dkk_raw <- purrr::map_dfr(
  years,
  \(y) get_public_holidays_dim(y, country = "DK")
)

cat("Antal rækker i dim_helligdage_dkk_raw:", nrow(dim_helligdage_dkk_raw), "\n")

# Hurtigt tjek i R - kun til udvikling / debugging
print(head(dim_helligdage_dkk_raw, 10))
str(dim_helligdage_dkk_raw)
```

```
> tibble(dim_helligdage_dkk_raw)
# A tibble: 402 × 5
   datekey  dato       helligdag_navn        english_name       type
   <chr>    <date>     <chr>                 <chr>              <chr>
 1 01012000 2000-01-01 Nytårsdag             New Year's Day     Officiel helligdag
 2 20042000 2000-04-20 Skærtorsdag           Maundy Thursday    Officiel helligdag
 3 21042000 2000-04-21 Langfredag            Good Friday        Officiel helligdag
 4 23042000 2000-04-23 Påskedag              Easter Sunday      Officiel helligdag
 5 24042000 2000-04-24 2. Påskedag           Easter Monday      Officiel helligdag
 6 19052000 2000-05-19 Store bededag         General Prayer Day Officiel helligdag
 7 01062000 2000-06-01 Kristi Himmelfartsdag Ascension Day      Officiel helligdag
 8 02062000 2000-06-02 Banklukkedag          Bank closing day   Bankferie, Skoleferie, Val…
 9 05062000 2000-06-05 Grundlovsdag          Constitution Day   Bankferie, Skoleferie, Val…
10 11062000 2000-06-11 Pinsedag              Pentecost          Officiel helligdag
# ℹ 392 more rows
# ℹ Use `print(n = ...)` to see more rows
```

1.1.5.4 Fuld load til PBA01_Raw – overwrite af helligdagsdimension

Vi sikrer at vi ikke mangler en pakke.

```r
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
```

```
  pacman::p_load(DBI, odbc, dplyr)
}); cat("Pakker loaded og klar \n")
```

1.1.5.5 Forbindelse til Azure SQL (RAW-lag)

I dette trin oprettes en sikker databaseforbindelse til Azure SQL ved hjælp af loginoplysninger fra miljøvariabler. Forbindelsen konfigureres specifikt til RAW-laget, så efterfølgende upload af helligdagetabel sker i det korrekte schema.

```
# Drift-/ETL-kode (køres ikke ved render)
# ------------------------------------------------------------
# Forbindelse til Azure SQL (login fra .Renviron)
# ------------------------------------------------------------
# Vi henter loginoplysninger fra miljøvariabler:
#   AZURE_SQL_SERVER
#   AZURE_SQL_DB
#   AZURE_SQL_UID
#   AZURE_SQL_PWD
#
# Schema:
#   - sættes nu til PBA01_Raw for at matche dit schema-design for RAW-laget.
server   <- Sys.getenv("AZURE_SQL_SERVER")
database <- Sys.getenv("AZURE_SQL_DB")
uid      <- Sys.getenv("AZURE_SQL_UID")
pwd      <- Sys.getenv("AZURE_SQL_PWD")
schema   <- "PBA01_Raw"
table    <- "dim_helligdage_dkk_raw"

if (any(!nzchar(c(server, database, uid, pwd)))) {
  stop("En eller flere miljøvariable til Azure SQL mangler. Tjek .Renviron og genstart R.")
}

con <- DBI::dbConnect(
  odbc::odbc(),
  driver   = "ODBC Driver 18 for SQL Server",
  server   = server,
  database = database,
  uid      = uid,
  pwd      = pwd,
  port     = 1433,
  Encrypt  = "yes",
  TrustServerCertificate = "no",
```

```
  ConnectionTimeout      = 30
)

cat("Forbundet til:", database, "\n")
```

1.1.5.6 Upload til Azure SQL Database

Her overskrives hele tabellen i PBA01_Raw med dim_helligdage_dkk_raw efter et kort sikkerhedstjek, så vi undgår at uploade et tomt/ikke-eksisterende objekt.
Efter upload valideres antallet af rækker direkte i SQL, og forbindelsen lukkes pænt for at afslutte load-trinnet.

```
# ------------------------------------------------------------
# Fuld load - overskriv hele tabellen i PBA01_Raw
# ------------------------------------------------------------
# Vi sikkerhedstjekker først, at objektet dim_helligdage_dkk_raw findes i R.
# Det forhindrer, at vi uploader en tom/ikke-eksisterende tabel ved en fejl.

if (!exists("dim_helligdage_dkk_raw")) {
  DBI::dbDisconnect(con)
  stop("Objektet 'dim_helligdage_dkk_raw' findes ikke - kør SCRIPT 1 først.")
}

cat(
  "Uploader dim_helligdage_dkk_raw til ",
  schema, ".", table, " ...\n",
  sep = ""
)

DBI::dbWriteTable(
  conn      = con,
  name      = DBI::Id(schema = schema, table = table),
  value     = dim_helligdage_dkk_raw,
  overwrite = TRUE,   # VIGTIGT: vi overskriver altid → ingen dubletter
  append    = FALSE
)

cat("Fuld load færdig. Antal rækker uploadet:", nrow(dim_helligdage_dkk_raw), "\n")

# Simpelt tjek direkte fra SQL - kun til validering
check_tbl <- DBI::dbReadTable(con, DBI::Id(schema = schema, table = table))
cat("Antal rækker i SQL efter upload:", nrow(check_tbl), "\n")
```

```r
print(head(check_tbl, 10))

# Luk forbindelsen, når vi er færdige
DBI::dbDisconnect(con)
cat("Forbindelse lukket.\n")
```

1.1.6 API Temperatur DMI Karup station - Raw temp_dry

I dette script etableres det rå datagrundlag for lufttemperaturmålinger (temp_dry, 2 m) fra DMI for Flyvestation Karup (station 06060). Data hentes via metObs-API'et og gemmes ufortolket i RAW-laget i Azure SQL som et faktalayer. Der foretages kun let teknisk formatering, mens al fortolkning og sammenkobling med øvrige datalag udskydes. Processen understøtter både et første fuldt historisk load og efterfølgende inkrementelle opdateringer for at sikre et konsistent og dubletfrit datasæt.

1.1.6.1 Pakker

```r
# ==============================================================================
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(httr2, jsonlite, dplyr, tibble, purrr, lubridate)
}); cat("Pakker klar til brug  \n")
```

1.1.6.2 API-nøgle

Her indlæses DMI API-nøglen sikkert fra miljøvariabler, og scriptet afbrydes tidligt, hvis nøglen mangler, så der ikke foretages uautoriserede eller fejlbehæftede API-kald.

```r
# Drift-/ETL-kode (køres ikke ved render)

# DMI_API_KEY læses fra .Renviron.
# Hvis den ikke findes, stopper vi, så vi ikke kalder API'et uden nøgle.

dmi_api_key <- Sys.getenv("DMI_API_KEY")
if (dmi_api_key == "") {
  stop("DMI_API_KEY er ikke sat - tjek .Renviron og genstart R.")
}
```

1.1.6.3 Opsætning af API-parametre (temp_dry)

I dette trin fastlægges de fælles API-parametre for alle kald, herunder endpoint, stationskode og den valgte temperaturparameter, så indsamlingen kan gennemføres konsistent på tværs af hele perioden.

```
# Faste parametre til alle kald:
#   - base_url: metObs observation-endpoint
#   - station_id: DMI-station 06060 (Karup)
#   - param_id: temp_dry (2 m lufttemperatur)
base_url   <- "https://dmigw.govcloud.dk/v2/metObs/collections/observation/items"
station_id <- "06060"        # Flyvestation Karup
param_id   <- "temp_dry"     # 2 m lufttemperatur
```

1.1.6.4 Funktion til rå API-kald pr. datointerval

Her defineres en hjælpefunktion, der opbygger et tidsinterval i UTC, kalder DMI metObs-API'et for det pågældende interval og omsætter svaret til en ensartet tibble. Funktionen håndterer tilfælde uden data ved at returnere et tomt datasæt, så den kan bruges sikkert i løkker over længere perioder.

```
# Rå kald-funktion for ét interval ----------------------------------------
# do_call_temp:
#   - bygger datetime-interval i ISO-format (UTC)
#   - kalder DMI API for det interval
#   - mapper JSON-features til en tibble
#   - returnerer tom tibble, hvis der ingen features er

do_call_temp <- function(start_date, end_date) {

  start_txt <- paste0(format(start_date, "%Y-%m-%d"), "T00:00:00Z")
  end_txt   <- paste0(format(end_date,   "%Y-%m-%d"), "T23:59:59Z")
  dt_range  <- paste0(start_txt, "/", end_txt)

  cat("  Kald DMI API for interval:", start_txt, "→", end_txt, "\n")

  req <- request(base_url) |>
    req_url_query(
      stationId   = station_id,
      parameterId = param_id,
      datetime    = dt_range,
      limit       = 100000   # rigeligt pr. år for én parameter
    ) |>
```

```
    req_headers(
      "X-Gravitee-Api-Key" = dmi_api_key
    )

  resp <- req_perform(req)
  x    <- resp_body_json(resp, simplifyVector = FALSE)

  if (is.null(x$features) || length(x$features) == 0) {
    cat("    Ingen features i dette interval.\n")
    return(tibble())
  }

  out <- purrr::map_dfr(
    x$features,
    function(feat) {
      tibble(
        stationId    = feat$properties$stationId,
        parameterId  = feat$properties$parameterId,
        observed_raw = feat$properties$observed,
        value        = feat$properties$value,
        lon          = feat$geometry$coordinates[[1]],
        lat          = feat$geometry$coordinates[[2]]
      )
    }
  )

  cat("    Rækker hentet i interval:", nrow(out), "\n")
  out
}
```

1.1.6.5 Robust hentefunktion med retry-logik

Dette trin indfører en wrapper omkring API-kaldet, som håndterer midlertidige fejl ved at forsøge igen et fast antal gange med pauser imellem. På den måde øges stabiliteten ved større datatræk, uden at hele processen afbrydes, hvis enkelte intervaller fejler.

```
#  Robust wrapper med retry (håndterer fx 504) ----------------------------
# fetch_temp_interval:
#   - kalder do_call_temp indenfor en while-løkke
#   - hvis kaldet fejler (try-error), forsøger vi igen op til max_retries
#   - indlægger 5 sekunders pause mellem forsøg
#   - returnerer tom tibble, hvis alle forsøg fejler
```

```r
fetch_temp_interval <- function(start_date, end_date, max_retries = 3) {
  attempt <- 1
  while (attempt <= max_retries) {
    cat("  Forsøg", attempt, "af", max_retries, "\n")

    res <- try(
      do_call_temp(start_date, end_date),
      silent = TRUE
    )

    if (!inherits(res, "try-error")) {
      return(res)
    }

    cat("    Fejl ved kald - prøver igen\n")

    if (attempt < max_retries) {
      cat("    Venter 5 sekunder\n")
      Sys.sleep(5)
    }

    attempt <- attempt + 1
  }

  cat("    GIVER OP for dette interval efter", max_retries, "forsøg - returnerer tomt tibble
  tibble()
}
```

1.1.6.6 Opbygning af årvise hentningsintervaller

Her opdeles hele perioden fra år 2000 til dags dato i årvise intervaller, så API-kald kan gennemføres kontrolleret og robust. Denne struktur gør det muligt at hente data systematisk og håndtere fejl uden at påvirke resten af perioden.

```r
# Lav års-intervaller fra 2000-01-01 til i dag ----------------------------
# For at gøre loadet robust laver vi ét kald pr. år:
#   - start_total / end_total definerer hele perioden
#   - years = 2000…nu
#   - intervals indeholder start- og slutdato for hvert år

start_total <- as.Date("2000-01-01")
end_total   <- Sys.Date()
```

```r
years <- seq(lubridate::year(start_total), lubridate::year(end_total))

intervals <- purrr::map(years, function(y) {
  start_y <- as.Date(paste0(y, "-01-01"))
  end_y   <- as.Date(paste0(y, "-12-31"))

  tibble(
    year  = y,
    start = max(start_y, start_total),
    end   = min(end_y,   end_total)
  )
}) |>
  bind_rows()

cat("Intervaller der hentes (år for år):\n")
print(intervals)
```

```
> print(intervals)
# A tibble: 27 × 2
   start      end
   <date>     <date>
 1 2000-01-01 2000-12-31
 2 2001-01-01 2001-12-31
 3 2002-01-01 2002-12-31
 4 2003-01-01 2003-12-31
 5 2004-01-01 2004-12-31
 6 2005-01-01 2005-12-31
 7 2006-01-01 2006-12-31
 8 2007-01-01 2007-12-31
 9 2008-01-01 2008-12-31
10 2009-01-01 2009-12-31
# i 17 more rows
# i Use `print(n = ...)` to see more rows
```

1.1.6.7 Hentning og samling af temp_dry-data

I dette trin hentes temperaturdata år for år ved hjælp af den robuste hentefunktion, hvorefter alle delresultater samles i ét samlet råt datasæt. Processen logger fremdrift pr. år og afsluttes med en samlet optælling af hentede observationer.

```r
# Drift-/ETL-kode (køres ikke ved render)

# Hent alle år og bind sammen ------------------------------------------
# Vi looper over alle årsintervaller med fetch_temp_interval
# og binder resultatet sammen til én stor tibble: temp_dry_karup_raw.
temp_list <- purrr::map(
  seq_len(nrow(intervals)),
  function(i) {
    this_year  <- intervals$year[i]
    start_i    <- intervals$start[i]
    end_i      <- intervals$end[i]

    cat("\n==========================================\n")
    cat("År:", this_year, "- henter temp_dry\n")
    cat("==========================================\n")

    fetch_temp_interval(start_i, end_i, max_retries = 3)
  }
)

temp_dry_karup_raw <- bind_rows(temp_list)

cat("\nSamlet antal rækker hentet (temp_dry):", nrow(temp_dry_karup_raw), "\n")

tibble(temp_dry_karup_raw)
```

```
> cat("\nSamlet antal rækker hentet (temp_dry):", nrow(temp_dry_karup_raw), "\n")

Samlet antal rækker hentet (temp_dry): 1211961
>
> tibble(temp_dry_karup_raw)
> tibble(temp_dry_karup_raw)
# A tibble: 1,211,961 × 6
   stationId parameterId observed_raw         value   lon   lat
   <chr>     <chr>       <chr>                <dbl> <dbl> <dbl>
 1 06060     temp_dry    2000-12-31T21:00:00Z  -3.8  9.11  56.3
 2 06060     temp_dry    2000-12-31T18:00:00Z  -4.2  9.11  56.3
 3 06060     temp_dry    2000-12-31T15:00:00Z  -4.6  9.11  56.3
 4 06060     temp_dry    2000-12-31T12:00:00Z  -4.8  9.11  56.3
 5 06060     temp_dry    2000-12-31T09:00:00Z  -9.1  9.11  56.3
 6 06060     temp_dry    2000-12-31T06:00:00Z  -8.6  9.11  56.3
 7 06060     temp_dry    2000-12-31T03:00:00Z  -6.6  9.11  56.3
 8 06060     temp_dry    2000-12-31T00:00:00Z  -6.4  9.11  56.3
 9 06060     temp_dry    2000-12-30T21:00:00Z  -2.5  9.11  56.3
10 06060     temp_dry    2000-12-30T18:00:00Z  -1    9.11  56.3
# i 1,211,951 more rows
# i Use `print(n = ...)` to see more rows
```

1.1.6.8 Teknisk klargøring af temp_dry RAW-data

Her udføres udelukkende teknisk rensning af de hentede temperaturdata, hvor tidsstempler standardiseres, kolonner afgrænses, og observationerne sorteres kronologisk. Datasættet forbliver et rent RAW-lag uden fortolkning eller mapping til andre datadimensioner.

```
# Drift-/ETL-kode (køres ikke ved render)

# Vi laver kun "teknisk" rensning:
#   - observed_raw → POSIXct med UTC
#   - vælger kun relevante kolonner
#   - sorterer kronologisk

# Dette er stadig RAW-laget (ingen tolkning, ingen mapping).
temp_dry_karup <- temp_dry_karup_raw |>
  mutate(
    observed = lubridate::ymd_hms(observed_raw, tz = "UTC")
  ) |>
  select(
    stationId,
    parameterId,
    observed,
    value,
    lon,
```

```
    lat
  ) |>
  arrange(observed)

cat("Dataset 'temp_dry_karup' er nu klar i R.\n")
print(head(temp_dry_karup))
View(temp_dry_karup, title = "temp_dry_karup - temp_dry for Karup")
```

```
> tibble(temp_dry_karup)
# A tibble: 1,211,961 × 6
   stationId parameterId observed            value   lon   lat
   <chr>     <chr>       <dttm>              <dbl> <dbl> <dbl>
 1 06060     temp_dry    2000-01-01 00:00:00   1.9  9.11  56.3
 2 06060     temp_dry    2000-01-01 03:00:00   2.8  9.11  56.3
 3 06060     temp_dry    2000-01-01 06:00:00   3.3  9.11  56.3
 4 06060     temp_dry    2000-01-01 09:00:00   4    9.11  56.3
 5 06060     temp_dry    2000-01-01 12:00:00   6.2  9.11  56.3
 6 06060     temp_dry    2000-01-01 15:00:00   5.5  9.11  56.3
 7 06060     temp_dry    2000-01-01 18:00:00   5    9.11  56.3
 8 06060     temp_dry    2000-01-01 21:00:00   3.1  9.11  56.3
 9 06060     temp_dry    2000-01-02 00:00:00   3.5  9.11  56.3
10 06060     temp_dry    2000-01-02 03:00:00   4.6  9.11  56.3
# ℹ 1,211,951 more rows
# ℹ Use `print(n = ...)` to see more rows
```

1.1.6.9 Første fulde load af temp_dry til Azure SQL

Denne del dokumenterer, hvordan det færdige RAW-datasæt for temp_dry flyttes fra R til Azure SQL som en etablering af faktalaget PBA01_Raw.fact_temp_dry_raw. Scriptet opretter forbindelse til databasen, klargør data med et load-timestamp og anvender en dummy-sikring, der afbryder, hvis der allerede findes data for station 06060 og parameteren temp_dry, så dubletter undgås. Selve uploaden sker som et append, og efter første historiske indlæsning er tanken, at tabellen vedligeholdes via en separat inkrementel opdatering. I rapportkontekst vises scriptet som dokumentation for drift/ETL og afvikles ikke nødvendigvis ved rendering.

```
# Drift-/ETL-kode (køres ikke ved render)

# ======================================================================
# SCRIPT A - FØRSTE FULDE LOAD AF TEMP_DRY_KARUP TIL Azure SQL
# (RAW-LAG)
#            → PBA01_Raw.fact_temp_dry_raw
#            (med dummy-sikring mod dublet-upload)
# ======================================================================
```

```r
# Script A flytter data fra R til Azure SQL (RAW-schema):
#   - opretter forbindelse til Azure
#   - sikrer at tabellen findes i PBA01_Raw
#   - tjekker om der allerede findes data for 06060/temp_dry
#   - uploader hele temp_dry_karup første gang (fuld historisk load)
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(DBI, odbc, dplyr, lubridate)
})

cat("=== TEMP_DRY - fuld load til Azure SQL ===\n")

# Læs login til Azure SQL fra .Renviron ----------------------------------
# Login læses fra miljøvariabler:
#   - AZURE_SQL_SERVER
#   - AZURE_SQL_DB
#   - AZURE_SQL_UID
#   - AZURE_SQL_PWD
server <- Sys.getenv("AZURE_SQL_SERVER")
db     <- Sys.getenv("AZURE_SQL_DB")
uid    <- Sys.getenv("AZURE_SQL_UID")
pwd    <- Sys.getenv("AZURE_SQL_PWD")
schema_raw <- "PBA01_Raw"

if (server == "" || db == "" || uid == "" || pwd == "") {
  stop("AZURE_SQL_* miljøvariable er ikke sat korrekt.")
}

# Opret forbindelse til Azure SQL ----------------------------------------
# Vi åbner ODBC-forbindelsen til Azure SQL.
# Denne bruges kun til selve uploaden i Script A.
cat("Opretter forbindelse til Azure SQL\n")

con <- dbConnect(
  drv    = odbc::odbc(),
  Driver   = "ODBC Driver 18 for SQL Server",
  Server   = server,
  Database = db,
  UID      = uid,
  PWD      = pwd,
  Encrypt  = "yes",
  TrustServerCertificate = "no",
```

```
  Authentication = "SqlPassword",
  Port     = 1433
)

cat("Forbindelse oprettet.\n")

# Tjek at temp_dry_karup findes ------------------------------------------
# Vi sikrer os, at Script 1 er kørt først, ellers giver det ingen mening
# at uploade.
if (!exists("temp_dry_karup")) {
  dbDisconnect(con)
  stop("Objektet 'temp_dry_karup' findes ikke i R - kør Script 1 (API) først.")
}

cat("Objekt 'temp_dry_karup' fundet i environment.\n")

# Forbered data til upload -----------------------------------------------
# Vi tilføjer load_timestamp, så vi kan se, hvornår rækken blev indlæst
# i databasen (klassisk datalager-praksis).
temp_dry_upload <- temp_dry_karup |>
  mutate(
    load_timestamp = Sys.time()
  )

cat("Antal rækker klar til upload:", nrow(temp_dry_upload), "\n")

# Opret tabel i SQL hvis den ikke findes ---------------------------------
# Her arbejder vi nu i RAW-schemaet PBA01_Raw (ikke længere dbo).
# Strukturen matcher 1:1 de kolonner, vi uploader fra R.
cat("Sikrer at PBA01_Raw.fact_temp_dry_raw findes\n")

dbExecute(con, sprintf("
IF NOT EXISTS (
  SELECT 1
  FROM INFORMATION_SCHEMA.TABLES
  WHERE TABLE_SCHEMA = '%s'
    AND TABLE_NAME   = 'fact_temp_dry_raw'
)
BEGIN
  CREATE TABLE %s.fact_temp_dry_raw (
    temp_id        bigint IDENTITY(1,1) PRIMARY KEY,
    stationId      varchar(10),
```

```
    parameterId    varchar(50),
    observed       datetime2,
    value          float,
    lon            float,
    lat            float,
    load_timestamp datetime2
  );
END;
", schema_raw, schema_raw))

cat("Tabel tjek/creation færdig.\n")

# DUMMY-SIKRING - tjek om der allerede er data for denne station/parameter
# Dummy-sikringen sørger for, at vi ikke kommer til at fuld-loade oven i
# eksisterende data for 06060/temp_dry.
# Tanken:
#   - fuld-load → kun på en tom tabel
#   - derefter bruges Script B (inkrementel) fremover.
antal_eksisterende <- dbGetQuery(con, sprintf("
  SELECT COUNT(1) AS n
  FROM %s.fact_temp_dry_raw
  WHERE stationId   = '06060'
    AND parameterId = 'temp_dry';
", schema_raw))$n[1]

if (antal_eksisterende > 0) {
  cat("\n*** DUMMY-SIKRING AKTIVERET ***\n")
  cat("Tabellen ", schema_raw, ".fact_temp_dry_raw indeholder allerede ",
      antal_eksisterende,
      " rækker for station 06060 / temp_dry.\n", sep = "")
  cat("Fuld-load afbrydes for at undgå dubletter.\n")
  dbDisconnect(con)
  stop("Fuld-load må kun køres på en tom tabel for denne station/parameter. Brug inkrementel
}

cat("Dummy-sikring: ingen eksisterende data for 06060 / temp_dry - fuld-load fortsætter.\n\n

# Upload data (første fulde load) ----------------------------------------
# Vi uploader alle rækker fra temp_dry_upload til PBA01_Raw.fact_temp_dry_raw.
# append = TRUE, overwrite = FALSE er sikkert, fordi vi lige har konstateret,
# at der ikke ligger data for 06060/temp_dry i forvejen.
cat("Uploader data til PBA01_Raw.fact_temp_dry_raw\n")
```

```
dbWriteTable(
  con,
  name      = DBI::Id(schema = schema_raw, table = "fact_temp_dry_raw"),
  value     = temp_dry_upload,
  append    = TRUE,
  overwrite = FALSE
)

cat("Første fulde load færdig. Antal rækker uploadet:", nrow(temp_dry_upload), "\n")

dbDisconnect(con)
cat("Forbindelse til Azure SQL lukket.\n")
```

1.1.6.10 Inkrementel opdatering af temp_dry (RAW-lag)

Denne del beskriver den løbende vedligeholdelse af temperaturdata efter første fulde indlæsning. Scriptet identificerer seneste registrerede observation i RAW-tabellen og henter herefter kun nye målinger fra DMI, som appenderes til databasen. Formålet er at sikre et opdateret og konsistent datasæt uden genindlæsning af historiske data eller risiko for dubletter.

```
# Drift-/ETL-kode (køres ikke ved render)

# ======================================================================
# SCRIPT B - TEMP_DRY - INKREMENTEL OPDATERING (KUN NYE RÆKKER)
# ======================================================================

# Script B bruges kun efter første fuld-load:
#   - finder seneste observed i PBA01_Raw.fact_temp_dry_raw
#   - henter data fra DMI API fra (seneste + 1 sekund) til nu
#   - appender kun de nye rækker til RAW-tabellen
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(DBI, odbc, dplyr, lubridate, httr2, tibble, purrr)
})

cat("=== TEMP_DRY - inkrementel opdatering ===\n")

# Læs login til Azure SQL ---------------------------------------------
# Igen læser vi loginoplysninger fra .Renviron.
server <- Sys.getenv("AZURE_SQL_SERVER")
db     <- Sys.getenv("AZURE_SQL_DB")
```

```
uid    <- Sys.getenv("AZURE_SQL_UID")
pwd    <- Sys.getenv("AZURE_SQL_PWD")
schema_raw <- "PBA01_Raw"

if (server == "" || db == "" || uid == "" || pwd == "") {
  stop("AZURE_SQL_* miljøvariable er ikke sat korrekt.")
}

# Opret forbindelse til Azure SQL ----------------------------------------
# Vi åbner en ny forbindelse, dedikeret til denne inkrementelle opdatering.
cat("Opretter forbindelse til Azure SQL\n")

con <- dbConnect(
  drv   = odbc::odbc(),
  Driver   = "ODBC Driver 18 for SQL Server",
  Server   = server,
  Database = db,
  UID      = uid,
  PWD      = pwd,
  Encrypt  = "yes",
  TrustServerCertificate = "no",
  Authentication = "SqlPassword",
  Port     = 1433
)

cat("Forbindelse oprettet.\n")

# Find seneste observed i SQL for Karup temp_dry
# Vi finder den maksimale observed for 06060/temp_dry.
# Dette tidspunkt styrer, hvorfra vi henter nye rækker i API'et.
cat("Finder seneste observed for station 06060, parameter 'temp_dry'\n")

last_obs_df <- dbGetQuery(con, sprintf("
  SELECT MAX(observed) AS last_obs
  FROM %s.fact_temp_dry_raw
  WHERE stationId   = '06060'
    AND parameterId = 'temp_dry';
", schema_raw))

last_obs <- last_obs_df$last_obs[1]

if (is.na(last_obs)) {
```

```
  dbDisconnect(con)
  stop("Der er ingen data i PBA01_Raw.fact_temp_dry_raw endnu - kør FULD LOAD først.")
}

cat("Seneste observed i SQL (UTC):", as.character(last_obs), "\n")

# Opsæt DMI API ----------------------------------------------------
# Vi genbruger samme endpoint og parametre:
#   - station_id = 06060
#   - param_id   = temp_dry
dmi_api_key <- Sys.getenv("DMI_API_KEY")
if (dmi_api_key == "") {
  dbDisconnect(con)
  stop("DMI_API_KEY er ikke sat - tjek .Renviron.")
}

base_url   <- "https://dmigw.govcloud.dk/v2/metObs/collections/observation/items"
station_id <- "06060"
param_id   <- "temp_dry"

# Definér tidsinterval for nye data --------------------------------
# Start = seneste observed + 1 sekund, slut = nu (UTC).
# På den måde undgår vi dubletter men mister heller ikke data.
start_time <- with_tz(last_obs, "UTC") + seconds(1)
end_time   <- with_tz(Sys.time(), "UTC")

start_txt <- format(start_time, "%Y-%m-%dT%H:%M:%SZ")
end_txt   <- format(end_time,   "%Y-%m-%dT%H:%M:%SZ")

dt_range  <- paste0(start_txt, "/", end_txt)

cat("Henter nye data i interval:", dt_range, "\n")

# Kald DMI API for nye observationer -------------------------------
# Vi kalder DMI med det dynamiske datetime-interval.
# Hvis API'et ikke returnerer features, stopper vi stille og roligt.
req <- request(base_url) |>
  req_url_query(
    stationId   = station_id,
    parameterId = param_id,
    datetime    = dt_range,
    limit       = 300000
```

```
  ) |>
  req_headers(
    "X-Gravitee-Api-Key" = dmi_api_key
  )

resp <- req_perform(req)
x    <- resp_body_json(resp, simplifyVector = FALSE)

if (is.null(x$features) || length(x$features) == 0) {
  cat("Ingen nye observationer fundet i DMI API.\n")
  dbDisconnect(con)
  cat("Forbindelse til Azure SQL lukket.\n")
} else {

  #  Map nye observationer til tibble ------------------------------------
  # Vi bygger new_temp i samme struktur som RAW-tabellen og tilføjer
  # load_timestamp, så vi kan se, hvornår opdateringen blev indlæst.
  new_temp <- purrr::map_dfr(
    x$features,
    function(feat) {
      tibble(
        stationId    = feat$properties$stationId,
        parameterId  = feat$properties$parameterId,
        observed     = lubridate::ymd_hms(feat$properties$observed, tz = "UTC"),
        value        = feat$properties$value,
        lon          = feat$geometry$coordinates[[1]],
        lat          = feat$geometry$coordinates[[2]]
      )
    }
  ) |>
    arrange(observed) |>
    mutate(load_timestamp = Sys.time())

  cat("Antal nye rækker hentet:", nrow(new_temp), "\n")

  # Upload nye rækker til RAW-tabellen ----------------------------------
  # Hvis der er nye rækker, appender vi dem til PBA01_Raw.fact_temp_dry_raw.
  if (nrow(new_temp) > 0) {
    cat("Uploader nye rækker til PBA01_Raw.fact_temp_dry_raw\n")

    dbWriteTable(
      con,
```

```
      name      = DBI::Id(schema = schema_raw, table = "fact_temp_dry_raw"),
      value     = new_temp,
      append    = TRUE,
      overwrite = FALSE
    )

    cat("Nye rækker er nu indsat i PBA01_Raw.fact_temp_dry_raw.\n")
  } else {
    cat("Der var ingen nye rækker at uploade.\n")
  }

  dbDisconnect(con)
  cat("Forbindelse til Azure SQL lukket.\n")
}
```

1.1.7 API Vejret DMI Karup station - Weather (Karup 06060)

I dette afsnit etableres det rå datagrundlag for vejrobservationer fra DMI for Flyvestation Karup (station 06060). Data hentes via metObs-API'et og gemmes ufortolket i RAW-laget i Azure SQL for at bevare en stabil og sporbar sandhedskilde. Der foretages kun nødvendige tekniske tilpasninger, mens al egentlig fortolkning udsættes til senere datalag. Processen understøtter både et første fuldt historisk load og efterfølgende inkrementelle opdateringer.

1.1.7.1 Pakker og setup

```
# Dette script:
#   - sætter DMI-API'et op (endpoint, station, parameter),
#   - definerer en hjælpefunktion til at hente ét datointerval,
#   - laver én række år-intervaller fra 2000 til i dag,
#   - looper år for år og binder alle resultater sammen til weather_karup.
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(httr2, jsonlite, dplyr, tibble, purrr, lubridate)
})
```

1.1.7.2 API-nøgle og adgangskontrol (DMI)

Dette trin sikrer, at DMI_API_KEY er korrekt sat i .Renviron, før der foretages kald til DMI's API. Hvis nøglen mangler, stoppes scriptet tidligt for at undgå fejlslagne eller uautoriserede API-kald.

```
# Drift-/ETL-kode (køres ikke ved render)

# API-nøgle ----------------------------------------------------
# DMI_API_KEY skal ligge i .Renviron.
# Hvis den ikke findes, stopper vi tidligt med en klar fejl.
dmi_api_key <- Sys.getenv("DMI_API_KEY")
if (dmi_api_key == "") {
  stop("DMI_API_KEY er ikke sat - tjek .Renviron og genstart R.")
}
```

1.1.7.3 Opsætning af faste API-parametre (DMI metObs)

Her defineres de faste parametre, der anvendes ved alle API-kald til DMI's metObs-endpoint, herunder station, parameter og base-URL. Det sikrer konsistens i dataudtrækket og gør senere ændringer nemme og kontrollerede.

```
# Opsætning ----------------------------------------------------
# Faste parametre for alle kald:
#   - base_url: metObs observation-endpoint.
#   - station_id: 06060 (Karup).
#   - param_id: weather (present weather).
base_url   <- "https://dmigw.govcloud.dk/v2/metObs/collections/observation/items"
station_id <- "06060"      # Flyvestation Karup
param_id   <- "weather"    # present weather
```

1.1.7.4 Funktion: hent vejrobservationer pr. datointerval (DMI metObs)

Denne funktion bygger et ISO-datetime-interval og henter alle observationer for den valgte station/parameter via DMI's API. Svaret foldes ud fra GeoJSON "features" til en ensartet tibble, og der returneres en tom tibble, hvis intervallet ikke giver data.

```
# Funktion til at hente ét datointerval ----------------------------------
# fetch_weather_interval:
#   - bygger et datetime-interval i ISO-format (UTC),
#   - kalder DMI API for det interval,
#   - mapper hver feature til en række i en tibble,
#   - returnerer tom tibble, hvis der ingen features er.
fetch_weather_interval <- function(start_date, end_date) {

  start_txt <- paste0(format(start_date, "%Y-%m-%d"), "T00:00:00Z")
  end_txt   <- paste0(format(end_date,   "%Y-%m-%d"), "T23:59:59Z")
```

```
  dt_range  <- paste0(start_txt, "/", end_txt)

  cat("  Henter interval:", start_txt, "→", end_txt, "\n")

  req <- request(base_url) |>
    req_url_query(
      stationId   = station_id,
      parameterId = param_id,
      datetime    = dt_range,
      limit       = 300000   # max pr. kald
    ) |>
    req_headers(
      "X-Gravitee-Api-Key" = dmi_api_key
    )

  resp <- req_perform(req)
  x    <- resp_body_json(resp, simplifyVector = FALSE)

  if (is.null(x$features) || length(x$features) == 0) {
    cat("    Ingen features i dette interval.\n")
    return(tibble())
  }

  out <- map_dfr(
    x$features,
    function(feat) {
      tibble(
        stationId    = feat$properties$stationId,
        parameterId  = feat$properties$parameterId,
        observed_raw = feat$properties$observed,
        value        = feat$properties$value,
        lon          = feat$geometry$coordinates[[1]],
        lat          = feat$geometry$coordinates[[2]]
      )
    }
  )

  cat("    Rækker hentet i dette interval:", nrow(out), "\n")
  out
}
```

1.1.7.5 Årsopdeling af datointervaller (robuste API-kald)

Den samlede periode opdeles i ét interval pr. år fra 2000 til dags dato. Det reducerer risiko for timeouts og gør datatrækket mere stabilt og reproducerbart.

```
# Drift-/ETL-kode (køres ikke ved render)


# Lav års-intervaller fra 2000-01-01 til i dag ----------------------------
# For at gøre kaldene robuste laver vi ét kald per år:
#   - start_total / end_total definerer hele perioden,
#   - years er listen over år,
#   - intervals indeholder start- og slutdato for hvert år.
start_total <- as.Date("2000-01-01")
end_total   <- Sys.Date()

years <- seq(year(start_total), year(end_total))

intervals <- map(years, function(y) {
  start_y <- as.Date(paste0(y, "-01-01"))
  end_y   <- as.Date(paste0(y, "-12-31"))

  tibble(
    start = max(start_y, start_total),
    end   = min(end_y,   end_total)
  )
}) |>
  bind_rows()

cat("Intervaller der hentes (år for år):\n")
print(intervals)
```

```
> print(intervals)
# A tibble: 27 × 2
   start      end
   <date>     <date>
 1 2000-01-01 2000-12-31
 2 2001-01-01 2001-12-31
 3 2002-01-01 2002-12-31
 4 2003-01-01 2003-12-31
 5 2004-01-01 2004-12-31
 6 2005-01-01 2005-12-31
 7 2006-01-01 2006-12-31
 8 2007-01-01 2007-12-31
 9 2008-01-01 2008-12-31
10 2009-01-01 2009-12-31
# ℹ 17 more rows
# ℹ Use `print(n = ...)` to see more rows
```

1.1.7.6 Indsamling af vejrdata på tværs af år

Her hentes vejrdata for hvert årsinterval via DMI-API'et og samles efterfølgende i ét samlet RAW-datasæt. Processen sikrer et fuldt og konsistent datagrundlag for hele perioden.

```
# Hent alle år og bind sammen ------------------------------------------
# Vi looper over alle års-intervaller med fetch_weather_interval
# og binder dem efterfølgende til ét samlet RAW-datasæt.
cat("  Starter hentning af weather for Karup...\n")

weather_list <- map2(intervals$start, intervals$end, fetch_weather_interval)

weather_karup_raw <- bind_rows(weather_list)

cat("Samlet antal rækker hentet:", nrow(weather_karup_raw), "\n")
```

1.1.7.7 Rensning og samling af RAW-vejrdata

I dette trin foretages en let teknisk rensning af de hentede observationer og data samles i én kronologisk tabel. Datasættet forbliver i RAW-laget og anvendes som grundlag for senere oversættelser og feature-engineering.

```r
# Drift-/ETL-kode (køres ikke ved render)


# Rens og lav endelig (RAW) tabel ----------------------------------------
# Vi laver kun "teknisk" rensning:
#   - observed_raw → POSIXct (UTC),
#   - vælger kun relevante kolonner,
#   - sorterer kronologisk.
# Det er stadig et RAW-lag - ingen oversættelser til kode-tabeller endnu.
weather_karup <- weather_karup_raw |>
  mutate(
    observed = ymd_hms(observed_raw, tz = "UTC")
  ) |>
  select(
    stationId,
    parameterId,
    observed,
    value,
    lon,
    lat
  ) |>
  arrange(observed)

# Kig på data (udvikling / QA)
cat("Første rækker af weather_karup:\n")
print(head(weather_karup))
View(weather_karup, title = "weather_karup - weather for Karup")
glimpse(weather_karup)
# Valgfrit: gem til senere brug lokalt
# saveRDS(weather_karup, "weather_karup_2000_nu.rds")
```

```
# A tibble: 6 × 6
  stationId parameterId observed            value   lon   lat
  <chr>     <chr>       <dttm>              <dbl> <dbl> <dbl>
1 06060     weather     2000-01-01 00:00:00    10  9.11  56.3
2 06060     weather     2000-01-01 03:00:00    10  9.11  56.3
3 06060     weather     2000-01-01 06:00:00    60  9.11  56.3
4 06060     weather     2000-01-01 09:00:00    60  9.11  56.3
5 06060     weather     2000-01-01 12:00:00    10  9.11  56.3
6 06060     weather     2000-01-01 15:00:00    10  9.11  56.3
```

1.1.7.8 RAW-load til Azure SQL: PBA01_Raw.fact_weather_raw

Dette script forbinder til Azure SQL og gennemfører et første fuldt historisk load af `weather_karup` til RAW-tabellen. Inden upload sikrer det, at tabellen findes, og en dummy-sikring stopper kørslen, hvis der allerede ligger data for station 06060 og parameter `weather` (så vi undgår dubletter).

```
# Drift-/ETL-kode (køres ikke ved render)
```

1.1.7.9 Første fulde upload af weather-data

I dette trin indlæses alle rækker fra det klargjorte RAW-datasæt i Azure SQL ved hjælp af DBI. Uploaden foretages som et append, da dummy-sikringen på forhånd har sikret, at der ikke findes eksisterende data for den pågældende station og parameter.

```
# Drift-/ETL-kode (køres ikke ved render)



# ======================================================================
# SCRIPT A - FØRSTE FULDE LOAD AF WEATHER_KARUP TIL Azure SQL (RAW-LAG)
#            → PBA01_Raw.fact_weather_raw
#            (med dummy-sikring mod dublet-upload)
# ======================================================================
# Dette script:
#   - opretter forbindelse til Azure SQL,
#   - sikrer at PBA01_Raw.fact_weather_raw findes,
#   - tjekker om der allerede er data for 06060/weather,
#   - uploader hele weather_karup første gang (fuld historisk load).
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(DBI, odbc, dplyr, lubridate)
})

cat("  Starter fuld load af weather_karup til Azure SQL...\n")

# Læs login til Azure SQL fra .Renviron ----------------------------------
# Loginoplysninger læses fra miljøvariabler, så de ikke hardcodes i scriptet.
server <- Sys.getenv("AZURE_SQL_SERVER")
db     <- Sys.getenv("AZURE_SQL_DB")
uid    <- Sys.getenv("AZURE_SQL_UID")
pwd    <- Sys.getenv("AZURE_SQL_PWD")
schema_raw <- "PBA01_Raw"
```

```r
if (server == "" || db == "" || uid == "" || pwd == "") {
  stop("AZURE_SQL_* miljøvariable er ikke sat korrekt.")
}

# Opret forbindelse til Azure SQL ----------------------------------------
# Vi åbner en ODBC-forbindelse til Azure.
# Denne bruges kun til fuld-load-delen.
cat("  Opretter forbindelse til Azure SQL...\n")

con <- dbConnect(
  drv    = odbc::odbc(),
  Driver   = "ODBC Driver 18 for SQL Server",
  Server   = server,
  Database = db,
  UID      = uid,
  PWD      = pwd,
  Encrypt  = "yes",
  TrustServerCertificate = "no",
  Authentication = "SqlPassword",
  Port     = 1433
)

cat("  Forbindelse oprettet.\n")

# Forbered data til upload -----------------------------------------------
# Vi sikrer, at weather_karup findes (API-scriptet skal være kørt først).
if (!exists("weather_karup")) {
  dbDisconnect(con)
  stop("Objektet 'weather_karup' findes ikke i R - kør API-scriptet først.")
}

# Vi tilføjer load_timestamp, så vi kan se, hvornår rækkerne blev indlæst.
weather_upload <- weather_karup |>
  mutate(
    load_timestamp = Sys.time()
  )

cat("  Klar til upload. Antal rækker i weather_upload:", nrow(weather_upload), "\n")

# Opret tabel i SQL hvis den ikke findes ---------------------------------
# Her arbejder vi nu i RAW-schemaet PBA01_Raw.
# Tabellen oprettes kun, hvis den ikke allerede findes.
```

```
cat("  Sikrer at PBA01_Raw.fact_weather_raw findes...\n")

dbExecute(con, sprintf("
IF NOT EXISTS (
  SELECT 1
  FROM INFORMATION_SCHEMA.TABLES
  WHERE TABLE_SCHEMA = '%s'
    AND TABLE_NAME   = 'fact_weather_raw'
)
BEGIN
  CREATE TABLE %s.fact_weather_raw (
    weather_id      bigint IDENTITY(1,1) PRIMARY KEY,
    stationId       varchar(10),
    parameterId     varchar(50),
    observed        datetime2,
    value           float,
    lon             float,
    lat             float,
    load_timestamp  datetime2
  );
END;
", schema_raw, schema_raw))

cat("  Tabel", paste0(schema_raw, ".fact_weather_raw"), "klar.\n")

# DUMMY-SIKRING - tjek om der allerede er data for denne station/parameter
# Dummy-sikringen sikrer, at vi kun fuld-loader én gang for 06060/weather.
antal_eksisterende <- dbGetQuery(con, sprintf("
  SELECT COUNT(1) AS n
  FROM %s.fact_weather_raw
  WHERE stationId   = '06060'
    AND parameterId = 'weather';
", schema_raw))$n[1]

if (antal_eksisterende > 0) {
  cat("\n*** DUMMY-SIKRING AKTIVERET ***\n")
  cat("Tabellen ", schema_raw, ".fact_weather_raw indeholder allerede ",
      antal_eksisterende,
      " rækker for station 06060 / weather.\n", sep = "")
  cat("Fuld-load afbrydes for at undgå dubletter.\n")
  dbDisconnect(con)
  stop("Fuld-load må kun køres på en tom tabel for denne station/parameter. Brug inkrementel
```

```
}

cat("Dummy-sikring: ingen eksisterende data for 06060 / weather - fuld-load fortsætter.\n\n")

# Upload data (første fulde load) ----------------------------------------
# Vi uploader alle rækker fra weather_upload til PBA01_Raw.fact_weather_raw.
# append = TRUE, overwrite = FALSE er sikkert, fordi vi lige har verificeret,
# at der ikke ligger data for denne station/parameter.
cat("  Uploader weather-data til PBA01_Raw.fact_weather_raw...\n")

dbWriteTable(
  con,
  name      = DBI::Id(schema = schema_raw, table = "fact_weather_raw"),
  value     = weather_upload,
  append    = TRUE,
  overwrite = FALSE
)

cat("  Første fulde load færdig. Antal rækker uploadet:", nrow(weather_upload), "\n")

dbDisconnect(con)
cat("  Forbindelse til Azure SQL lukket.\n")
```

1.1.7.10 Inkrementel opdatering af RAW weather-data

Dette script finder seneste registrerede observation i RAW-tabellen og henter kun nye DMI-målinger siden da, så tabellen kan vedligeholdes uden dubletter. De nye rækker transformeres til samme RAW-struktur og appenderes til PBA01_Raw.fact_weather_raw.

```
# Drift-/ETL-kode (køres ikke ved render)

# ============================================================================
# SCRIPT B - INKREMENTEL OPDATERING AF WEATHER-DATA I Azure SQL
#            → PBA01_Raw.fact_weather_raw
# ============================================================================
# Dette script:
#   - finder seneste observed i PBA01_Raw.fact_weather_raw,
#   - bygger et datetime-interval fra (seneste + 1 sekund) til nu,
#   - henter kun de nye observationer fra DMI API,
#   - appender de nye rækker til RAW-tabellen.
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
```

```
  pacman::p_load(DBI, odbc, dplyr, lubridate, httr2, tibble, purrr)
})

cat("  Starter inkrementel opdatering af weather for Karup (06060)...\n")

# 1) Læs login til Azure SQL ------------------------------------------------
server <- Sys.getenv("AZURE_SQL_SERVER")
db     <- Sys.getenv("AZURE_SQL_DB")
uid    <- Sys.getenv("AZURE_SQL_UID")
pwd    <- Sys.getenv("AZURE_SQL_PWD")
schema_raw <- "PBA01_Raw"

if (server == "" || db == "" || uid == "" || pwd == "") {
  stop("AZURE_SQL_* miljøvariable er ikke sat korrekt.")
}

# 2) Opret forbindelse til Azure SQL ----------------------------------------
cat("  Opretter forbindelse til Azure SQL...\n")

con <- dbConnect(
  drv   = odbc::odbc(),
  Driver   = "ODBC Driver 18 for SQL Server",
  Server   = server,
  Database = db,
  UID      = uid,
  PWD      = pwd,
  Encrypt  = "yes",
  TrustServerCertificate = "no",
  Authentication = "SqlPassword",
  Port     = 1433
)

cat("  Forbindelse oprettet.\n")

# 3) Find seneste observed i SQL for Karup weather --------------------------
# Vi finder den maksimale observed for 06060/weather - det er vores "cut".
last_obs_df <- dbGetQuery(con, sprintf("
  SELECT MAX(observed) AS last_obs
  FROM %s.fact_weather_raw
  WHERE stationId   = '06060'
    AND parameterId = 'weather';
", schema_raw))
```

```r
last_obs <- last_obs_df$last_obs[1]

if (is.na(last_obs)) {
  dbDisconnect(con)
  stop("Der er ingen data i PBA01_Raw.fact_weather_raw endnu - kør fuld-load-scriptet først.")
}

cat("  Seneste observed i SQL:", as.character(last_obs), "\n")

# 4) Opsæt DMI API ------------------------------------------------------------
dmi_api_key <- Sys.getenv("DMI_API_KEY")
if (dmi_api_key == "") {
  dbDisconnect(con)
  stop("DMI_API_KEY er ikke sat - tjek .Renviron.")
}

base_url   <- "https://dmigw.govcloud.dk/v2/metObs/collections/observation/items"
station_id <- "06060"
param_id   <- "weather"

# 5) Definér tidsinterval for nye data ----------------------------------------
# Start = seneste observed + 1 sekund, slut = nu (UTC),
# så vi undgår dubletter men ikke misser målinger.
start_time <- with_tz(last_obs, "UTC") + seconds(1)
end_time   <- with_tz(Sys.time(), "UTC")

start_txt <- format(start_time, "%Y-%m-%dT%H:%M:%SZ")
end_txt   <- format(end_time,   "%Y-%m-%dT%H:%M:%SZ")

dt_range  <- paste0(start_txt, "/", end_txt)

cat("  Henter nye data i interval:", dt_range, "\n")

# 6) Kald DMI API for nye observationer ---------------------------------------
req <- request(base_url) |>
  req_url_query(
    stationId   = station_id,
    parameterId = param_id,
    datetime    = dt_range,
    limit       = 300000
  ) |>
  req_headers(
```

```
    "X-Gravitee-Api-Key" = dmi_api_key
  )

resp <- req_perform(req)
x    <- resp_body_json(resp, simplifyVector = FALSE)

if (is.null(x$features) || length(x$features) == 0) {
  cat("  Ingen nye observationer fundet i DMI API.\n")
  dbDisconnect(con)
  cat("  Forbindelse lukket.\n")
} else {

  # Map nye observationer til tibble i samme struktur som RAW-tabellen
  new_weather <- map_dfr(
    x$features,
    function(feat) {
      tibble(
        stationId    = feat$properties$stationId,
        parameterId  = feat$properties$parameterId,
        observed     = ymd_hms(feat$properties$observed, tz = "UTC"),
        value        = feat$properties$value,
        lon          = feat$geometry$coordinates[[1]],
        lat          = feat$geometry$coordinates[[2]]
      )
    }
  ) |>
    arrange(observed) |>
    mutate(load_timestamp = Sys.time())

  cat("  Antal nye rækker hentet:", nrow(new_weather), "\n")

  if (nrow(new_weather) > 0) {
    # Append nye rækker til RAW-tabellen i PBA01_Raw
    dbWriteTable(
      con,
      name      = DBI::Id(schema = schema_raw, table = "fact_weather_raw"),
      value     = new_weather,
      append    = TRUE,
      overwrite = FALSE
    )
    cat("  Nye rækker er nu indsat i", paste0(schema_raw, ".fact_weather_raw"), "\n")
  } else {
```

```
    cat("  Der var ingen nye rækker at uploade.\n")
  }

  dbDisconnect(con)
  cat("  Forbindelse lukket.\n")
}

cat("  Inkrementel opdatering af weather færdig.\n")
```

1.1.8 Vejrkode DMI dimension

Scraping-koden til DMI's weather-koder er oprindeligt udviklet på baggrund af en dokumentationsside, hvor indholdet blev leveret som klassiske, server-renderede HTML-tabeller. I slutningen af 2025 ændrede DMI den tekniske implementering, så weather-koderne nu leveres som rå tekst (TSV) i `<textarea>`-elementer og først renderes via JavaScript i browseren.

I dette projekt er SQl/dimensionen opbygget på det daværende HTML-grundlag (før ændringen). I QMD-filen anvendes derimod en opdateret scraping-metode, som passer til den nuværende version af DMI's side, så koden fortsat kan køres og dokumenteres reproducerbart.

1.1.8.1 Pakker og datakilde

Dette afsnit indlæser de nødvendige pakker til webscraping og datarensning samt definerer URL'en til DMI's dokumentationsside for weather-koder.

```
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(
    rvest, dplyr, purrr, stringr, tidyr, tibble,
    readr, janitor
  )
}); cat("Pakker klar og loaded \n")

url <- "https://www.dmi.dk/friedata/dokumentation/meteorological-observations-data#weather"
```

1.1.8.2 Udvidelse af kodeintervaller

Denne hjælpefunktion normaliserer og udvider weather-kodeintervaller (fx 20–29) til enkeltkoder. Det sikrer en ensartet og entydig repræsentation af alle koder i RAW-dimensionen

```
# ----------------------------------------------------------------------# Hjælper: udvid interv
# ----------------------------------------------------------------------
expand_code_range <- function(x) {
  x <- stringr::str_squish(as.character(x))
  x <- stringr::str_replace_all(x, "–", "-")
  x <- stringr::str_replace_all(x, "—", "-")

  a <- suppressWarnings(as.integer(stringr::str_extract(x, "^\\d+")))
  b <- suppressWarnings(as.integer(stringr::str_extract(x, "(?<=-)\\d+$")))

  if (is.na(a)) return(integer(0))
  if (is.na(b)) return(a)
  seq.int(a, b)
}
```

1.1.8.3 Parser TSV-blok fra DMI-dokumentation

Funktionen omdanner én TSV-blok fra `<textarea>` til en ensartet tabel med `code_raw` og engelsk tekst, og logger eventuelle parsing-advarsler. Det gør scraping robust over for DMI's "pseudo-tags" og små ændringer i tabellernes struktur.

```
# ----------------------------------------------------------------------
# Parser én TSV-blok fra textarea -> tibble(code_raw, weather_text_en)
#    + fanger parsing warnings og printer dem via cat()
# ----------------------------------------------------------------------
parse_code_tsv_block <- function(txt, block_id = NA_character_) {
  txt <- as.character(txt)

  # Fjern DMI pseudo-tags inde i celler: [colspan=2], [rowspan=4] osv.
  txt <- stringr::str_replace_all(txt, "\\[[^\\]]+\\]", "")
  txt <- stringr::str_replace_all(txt, "\r", "")
  txt <- stringr::str_trim(txt)

  if (!nzchar(txt)) {
    return(tibble::tibble(block_id = block_id, code_raw = character(), weather_text_en = cha
  }

  warn_msgs <- character(0)

  df <- withCallingHandlers(
    {
      readr::read_tsv(
```

```
      I(txt),
      col_types = readr::cols(.default = readr::col_character()),
      show_col_types = FALSE,
      progress = FALSE
    ) |>
      janitor::clean_names()
  },
  warning = function(w) {
    warn_msgs <<- c(warn_msgs, conditionMessage(w))
    invokeRestart("muffleWarning")
  }
)

# Print warnings pænt (hvis nogen)
if (length(warn_msgs) > 0) {
  cat("  Parsing-advarsler i TSV-blok", block_id, "(", length(warn_msgs), "):\n", sep = "")
  cat("    - ", paste(unique(warn_msgs), collapse = "\n    - "), "\n", sep = "")
}

# readr kan også have konkrete parsing-problemer (vroom)
probs <- tryCatch(readr::problems(df), error = function(e) tibble::tibble())
if (nrow(probs) > 0) {
  cat("  Konkrete parsing-problemer i TSV-blok", block_id, ":", nrow(probs), "\n", sep = ""
  print(utils::head(probs, 10))
}

# Vi leder efter en kodekolonne
code_col <- intersect(names(df), c("code", "codes"))
if (length(code_col) == 0) {
  return(tibble::tibble(block_id = block_id, code_raw = character(), weather_text_en = cha
}
code_col <- code_col[1]

value_cols <- setdiff(names(df), code_col)

if (length(value_cols) == 0) {
  out <- df |>
    dplyr::transmute(
      block_id = block_id,
      code_raw = .data[[code_col]],
      weather_text_en = ""
    )
```

```
  } else {
    out <- df |>
      dplyr::mutate(
        weather_text_en = purrr::pmap_chr(
          dplyr::across(dplyr::all_of(value_cols)),
          ~{
            vals <- c(...)
            vals <- vals[!is.na(vals) & vals != ""]
            if (length(vals) == 0) "" else stringr::str_squish(paste(vals, collapse = " - "))
          }
        )
      ) |>
      dplyr::transmute(
        block_id = block_id,
        code_raw = .data[[code_col]],
        weather_text_en = weather_text_en
      )
  }

  out |>
    dplyr::mutate(
      code_raw = stringr::str_squish(as.character(code_raw)),
      weather_text_en = stringr::str_squish(as.character(weather_text_en))
    )
}
```

1.1.8.4 Hent WeatherCode via DMI's <textarea>

Funktionen finder alle `textarea.dmi-input`, filtrerer til blokke med `Code/Codes`, parser dem og udvider intervaller til enkeltkoder, så vi ender med en komplet tabel for weather_code 0–199 inkl. kontrol af manglende koder.

```
# ---------------------------------------------------------------------
# NY METODE (2025+): hent alle textarea.dmi-input og filtrér til WeatherCode
# ---------------------------------------------------------------------
try_get_weather_from_dmi_textarea <- function(url) {
  page <- rvest::read_html(url)

  textareas <- page |> rvest::html_elements("textarea.dmi-input")
  if (length(textareas) == 0) stop("Fandt ingen textarea.dmi-input. DMI kan have ændret sider

  tsv_blocks <- purrr::map_chr(textareas, ~ rvest::html_text(.x, trim = TRUE))
```

```
tsv_blocks <- tsv_blocks[nzchar(tsv_blocks)]

cat("  textarea-blokke fundet:", length(tsv_blocks), "\n")

# Heuristik: behold kun blokke der "ligner" kode-tabeller (har Code/Codes header-linje)
# (vi undgår parameterlisten med Name/Unit/Description osv.)
first_line <- function(x) strsplit(x, "\n", fixed = TRUE)[[1]][1]
keep <- vapply(tsv_blocks, function(x) {
  fl <- first_line(x)
  stringr::str_detect(fl, "^(Code|Codes)\\b")
}, logical(1))

tsv_blocks_codes <- tsv_blocks[keep]
cat("  Kandidat-blokke med Code/Codes header:", length(tsv_blocks_codes), "\n")

parsed_all <- purrr::imap_dfr(
  tsv_blocks_codes,
  ~ parse_code_tsv_block(.x, block_id = as.character(.y))
)

if (nrow(parsed_all) == 0) stop("Kunne ikke parse nogen kode-tabeller fra textarea-blokkene

weather_code_0_199 <- parsed_all |>
  dplyr::mutate(
    code_raw_clean = stringr::str_replace_all(code_raw, "\\s", ""),
    code_raw_clean = stringr::str_replace_all(code_raw_clean, "–", "-"),
    code_raw_clean = stringr::str_replace_all(code_raw_clean, "—", "-"),
    code_min = suppressWarnings(as.integer(stringr::str_extract(code_raw_clean, "^\\d+"))),
    code_max = suppressWarnings(as.integer(stringr::str_extract(code_raw_clean, "(?<=-)\\d+
    code_max = dplyr::if_else(is.na(code_max), code_min, code_max)
  ) |>
  dplyr::filter(!is.na(code_min)) |>
  dplyr::rowwise() |>
  dplyr::mutate(weather_code = list(seq.int(code_min, code_max))) |>
  tidyr::unnest(weather_code) |>
  dplyr::ungroup() |>
  dplyr::filter(weather_code >= 0, weather_code <= 199) |>
  dplyr::arrange(weather_code, block_id) |>
  dplyr::distinct(weather_code, .keep_all = TRUE) |>
  dplyr::transmute(
    weather_code = as.integer(weather_code),
    weather_text_en = weather_text_en,
```

```
      code_raw = code_raw,
      block_id = block_id
    )

  cat("  Weather codes (0-199) rækker:", nrow(weather_code_0_199), "\n")

  missing_codes <- setdiff(0:199, weather_code_0_199$weather_code)
  cat("  Manglende koder i 0-199:", length(missing_codes), "\n")
  if (length(missing_codes) > 0) print(missing_codes)

  weather_code_0_199
}
```

1.1.8.5 Legacy fallback: parse WeatherCode fra HTML-tabeller (før 2025)

Denne funktion forsøger den gamle html_table()-baserede scraping, men forventes ofte at give 0 rækker nu, fordi DMI's WeatherCode-sektion typisk renderes via JavaScript i stedet for server-renderet HTML.

```
# ------------------------------------------------------------------
# LEGACY METODE (før 2025): parse HTML-tabeller (dokumentation/fallback)
#    Forventning nu: ofte 0 rækker pga JS-rendering på DMI.dk
# ------------------------------------------------------------------
try_get_weather_from_html_tables_legacy <- function(url) {
  suppressPackageStartupMessages(pacman::p_load(rvest, dplyr, purrr, stringr, tidyr, tibble)

  page <- rvest::read_html(url)

  # Forsøg at finde tabeller under Weather og før næste sektion
  # (denne XPath kan være "rigtig" historisk, men giver ofte 0 på den nuværende side)
  weather_tables_nodes <- page |>
    rvest::html_elements(
      xpath = "//h3[@id='weather']/following::table
               [preceding::h3[@id='weather'] and following::h2[@id='stations']]"
    )

  cat("  Legacy: HTML-tabeller fundet:", length(weather_tables_nodes), "\n")

  if (length(weather_tables_nodes) == 0) {
    return(tibble::tibble(weather_code = integer(), weather_text_en = character(), code_raw =
  }
```

```r
parse_weather_table <- function(tbl_node) {
  tbl <- rvest::html_table(tbl_node, fill = TRUE)
  tbl <- tibble::as_tibble(tbl)
  names(tbl) <- make.unique(names(tbl))

  tbl <- tbl |>
    dplyr::rename(code_raw = 1) |>
    dplyr::mutate(code_raw = stringr::str_squish(as.character(code_raw)))

  if (ncol(tbl) == 1) {
    tbl <- tbl |> dplyr::mutate(weather_text_en = "")
  } else {
    value_cols <- names(tbl)[2:ncol(tbl)]
    tbl <- tbl |>
      dplyr::mutate(
        weather_text_en = purrr::pmap_chr(
          dplyr::across(dplyr::all_of(value_cols)),
          ~{
            vals <- c(...)
            vals <- vals[!is.na(vals) & vals != ""]
            if (length(vals) == 0) "" else stringr::str_squish(paste(vals, collapse = " - "
          }
        )
      )
  }

  tbl |>
    dplyr::select(code_raw, weather_text_en)
}

weather_codes_raw <- purrr::map_dfr(weather_tables_nodes, parse_weather_table)

weather_codes_raw |>
  dplyr::mutate(
    code_raw_clean = stringr::str_replace_all(code_raw, "\\s", ""),
    code_raw_clean = stringr::str_replace_all(code_raw_clean, "–", "-"),
    code_raw_clean = stringr::str_replace_all(code_raw_clean, "—", "-"),
    code_min = suppressWarnings(as.integer(stringr::str_extract(code_raw_clean, "^\\d+"))),
    code_max = suppressWarnings(as.integer(stringr::str_extract(code_raw_clean, "(?<=-)\\d+
    code_max = dplyr::if_else(is.na(code_max), code_min, code_max)
  ) |>
  dplyr::filter(!is.na(code_min)) |>
```

```
    dplyr::rowwise() |>
    dplyr::mutate(weather_code = list(seq.int(code_min, code_max))) |>
    tidyr::unnest(weather_code) |>
    dplyr::ungroup() |>
    dplyr::filter(weather_code >= 0, weather_code <= 199) |>
    dplyr::arrange(weather_code) |>
    dplyr::distinct(weather_code, .keep_all = TRUE) |>
    dplyr::transmute(
      weather_code = as.integer(weather_code),
      weather_text_en = weather_text_en,
      code_raw = code_raw
    )
}
```

1.1.8.6 Kørsel: hent WeatherCode 0–199 med ny metode (textarea/TSV)

Her eksekveres den opdaterede scraping, som udlæser TSV-blokkene fra DMI-sidens `textarea.dmi-input` og samler dem til én tabel med WeatherCode 0–199 til videre brug i projektet.

```
# Drift-/ETL-kode (køres ikke ved render)

# ------------------------------------------------------------------
# KØR: Primært ny metode. Legacy er kun dokumentation/fallback.
# ------------------------------------------------------------------
weather_code_0_199 <- try_get_weather_from_dmi_textarea(url)

# Hurtigt view
glimpse(weather_code_0_199)
```

```
> glimpse(weather_code_0_199)
Rows: 200
Columns: 4
$ weather_code    <int> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, …
$ weather_text_en <chr> "calm", "Clouds generally dissolving or becoming less developed", "State of s
y on the whole unchanged", "Clouds generally forming…
$ code_raw        <chr> "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "1
4", "15", "16", "17", "18", "19", "20", "21", "22", "2…
$ block_id        <chr> "3", "5", "5", "5", "5", "5", "5", "5", "5", "5", "4", "5", "5", "5", "5",
"5", "5", "5", "5", "5", "6", "6", "6", "6", "6", "4", …
```

1.1.8.7 Upload af RAW WeatherCode til Azure SQL

Her uploades den færdige RAW-dimension til PBA01_Raw.dim_weather_code_raw, som overskrives ved hver kørsel. Dermed er RAW-laget altid synkroniseret med den seneste dokumentation fra DMI.

```
# Drift-/ETL-kode (køres ikke ved render)

# ============================================================
# Upload til Azure SQL (RAW → PBA01_Raw.dim_weather_code_raw)
# ============================================================
# Til sidst uploader vi RAW-dimensionen til Azure SQL.
# Loginoplysninger hentes som miljøvariable fra .Renviron, så scriptet kan
# ligge på GitHub uden at afsløre credentials.
#
# Tabellen skrives til PBA01_Raw.dim_weather_code_raw og overskrives ved hver kørsel,
# så RAW-laget altid er synkroniseret med den seneste dokumentation fra DMI.

# Hent login fra .Renviron
server   <- Sys.getenv("AZURE_SQL_SERVER")
database <- Sys.getenv("AZURE_SQL_DB")
uid      <- Sys.getenv("AZURE_SQL_UID")
pwd      <- Sys.getenv("AZURE_SQL_PWD")

if (any(!nzchar(c(server, database, uid, pwd)))) {
  stop("Manglende miljøvariable: Tjek .Renviron og genstart R.")
}

schema <- "PBA01_Raw"          #  SKIFTET FRA 'dbo' TIL 'PBA01_Raw'
table  <- "dim_weather_code_raw"

# Opret SQL-forbindelse
con_azure <- DBI::dbConnect(
  odbc::odbc(),
  .connection_string = paste0(
    "Driver={ODBC Driver 18 for SQL Server};",
    "Server=", server, ";",
    "Database=", database, ";",
    "Uid=", uid, ";",
    "Pwd=", pwd, ";",
    "Encrypt=yes;",
    "TrustServerCertificate=no;",
    "Connection Timeout=30;"
```

```
  )
)

# Upload tabel (RAW)
DBI::dbWriteTable(
  conn       = con_azure,
  name       = DBI::Id(schema = schema, table = table),
  value      = weather_code_0_199,
  overwrite  = TRUE
)

# Validering
head(DBI::dbReadTable(con_azure, DBI::Id(schema = schema, table = table)))

DBI::dbDisconnect(con_azure)
```

1.1.9 API Befolkning Viborg (DST)

1.1.9.1 Pakker

Her loader vi de nødvendige pakker til at hente befolkningsdata via API, læse dem ind i R og lave et hurtigt RAW-tjek. Samtidig klargør vi DBI/ODBC-værktøjerne, så datasættet senere kan flyttes videre til databasen uden at blande analyse og indlæsning.

```
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(
    httr, rjstat, dplyr, DBI, odbc, tibble, rstudioapi, dbplyr, rlang
  )
});  cat("Pakker klar og loaded \n")
```

1.1.9.2 Oprettelse af tidsdimension (kvartaler)

I dette afsnit konstrueres en samlet tidsvektor med alle kvartaler fra 2008K1 til 2025K4. Strengen bruges som input til API-kaldet, så hele perioden kan hentes konsistent i én samlet forespørgsel.

```
år      <- 2008:2025
kvartal <- paste0("K", 1:4)

tid_værdier <- as.vector(t(outer(år, kvartal, paste0)))
tid_streng  <- paste(tid_værdier, collapse = ",")
```

1.1.9.3 Hent befolkningsdata fra Danmarks Statistik

Kalder FOLK1A-API'et for Viborg Kommune og henter kvartalsvise befolkningstal. Resultatet gemmes som RAW-data til senere rensning og brug i analyser.

```
# ======================================================================
# Definér API-endpoint og query-parametre
# ======================================================================

dst_url <- "https://api.statbank.dk/v1/data/FOLK1A/JSONSTAT"

dst_query <- list(
  "OMRÅDE"     = "791",           # Viborg
  "KØN"        = "TOT,1,2",       # I alt, mænd, kvinder
  "ALDER"      = "IALT",          # Alder i alt
  "CIVILSTAND" = "TOT,U,G,E,F",   # Civilstand
  "Tid"        = tid_streng
)

dst_response <- GET(
  url   = dst_url,
  query = dst_query
)

kode <- status_code(dst_response)
cat("HTTP-status fra DST:", kode, "\n")
if (kode != 200) {
  stop("Fejl ved kald til Danmarks Statistik (status = ", kode, ").")
}

dst_json <- dst_response |>
  content(as = "text", encoding = "UTF-8")

dst_parsed <- fromJSONstat(dst_json)

befolkning_viborg_raw <- dst_parsed[[1]] |>
  as_tibble()

cat("\n  Strukturen af befolkning_viborg_raw:\n")
str(befolkning_viborg_raw)
```

```
cat("\n  Første 10 rækker:\n")
print(head(befolkning_viborg_raw, 10))
```

1.1.9.4 Sikring af database-login (.Renviron)

Tjekker at alle nødvendige Azure SQL-miljøvariabler er sat korrekt. Stopper processen og guider til opsætning, hvis der mangler loginoplysninger.

```
# Drift-/ETL-kode (køres ikke ved render)

# ------------------------------------------------------------------------
# Safeguard .Renviron
# ------------------------------------------------------------------------

required_vars <- c(
  "AZURE_SQL_SERVER",
  "AZURE_SQL_DB",
  "AZURE_SQL_UID",
  "AZURE_SQL_PWD"
)

values <- Sys.getenv(required_vars)
missing_vars <- required_vars[values == ""]

if (length(missing_vars) > 0) {
  cat("  Følgende miljøvariabler mangler i .Renviron:\n")
  print(missing_vars)
  cat("\nÅbner ~/.Renviron - udfyld værdierne og gem filen.\n")

  file.edit("~/.Renviron")
  cat("  R genstartes nu via RStudio - kør scriptet igen bagefter.\n")

  if (rstudioapi::isAvailable()) {
    rstudioapi::restartSession()
  } else {
    stop("rstudioapi er ikke tilgængelig - genstart R manuelt og kør scriptet igen.")
  }
} else {
  cat("  Alle nødvendige .Renviron-variabler er sat.\n\n")
}
```

1.1.9.5 Azure SQL-forbindelse med retry

Opretter forbindelse til Azure SQL med automatisk genforsøg og stigende timeout. Sikrer robust forbindelse selv ved midlertidige netværks- eller timeout-fejl.

```
# Drift-/ETL-kode (køres ikke ved render)

# ------------------------------------------------------------------
# Azure forbindelse med automatisk retry
# ------------------------------------------------------------------

forsøg_max  <- 6
timeouts    <- c(60, 180, 200, 260, 360, 600)
delay_sec   <- 10
forsøg      <- 1
con         <- NULL

while (forsøg <= forsøg_max && is.null(con)) {

  timeout_brug <- timeouts[min(forsøg, length(timeouts))]

  cat(
    "Forsøg", forsøg,
    "på at forbinde (ConnectionTimeout =", timeout_brug, "sekunder)...\n"
  )

  con_try <- try(
    dbConnect(
      odbc::odbc(),
      driver   = "ODBC Driver 18 for SQL Server",
      server   = Sys.getenv("AZURE_SQL_SERVER"),
      database = Sys.getenv("AZURE_SQL_DB"),
      uid      = Sys.getenv("AZURE_SQL_UID"),
      pwd      = Sys.getenv("AZURE_SQL_PWD"),
      port     = 1433,
      Encrypt  = "yes",
      TrustServerCertificate = "no",
      ConnectionTimeout      = timeout_brug
    ),
    silent = TRUE
  )

  if (!inherits(con_try, "try-error")) {
```

```
    con <- con_try
    cat(
      "  Forbundet til:", Sys.getenv("AZURE_SQL_DB"),
      "på forsøg", forsøg, "\n\n"
    )
  } else {
    cat("  Forbindelsen fejlede på forsøg", forsøg, "- prøver igen.\n")

    if (forsøg == forsøg_max) {
      stop("Kunne ikke forbinde til Azure SQL efter ", forsøg_max, " forsøg.\n")
    }

    cat("Venter", delay_sec, "sekunder før næste forsøg...\n\n")
    Sys.sleep(delay_sec)
    forsøg <- forsøg + 1
  }
}
```

1.1.9.6 Første load + incremental update (uden rå SQL)

Her tjekker vi, om RAW-tabellen allerede findes i PBA01_Raw, og vælger automatisk korrekt strategi. Hvis tabellen ikke findes, oprettes den og fuld-loades; hvis den findes, indsættes kun nye kvartaler for at undgå dubletter.

```
# Drift-/ETL-kode (køres ikke ved render)

# ------------------------------------------------------------------------
# Første load + incremental update UDEN rå SQL
# ------------------------------------------------------------------------

# Vi antager schemaet hedder præcist "PBA01_Raw".
# I stedet for et separat schema-tjek bruger vi dbExistsTable
# og fanger fejl med try(): hvis schema ikke findes, vil det fejle.

table_id <- Id(
  schema = "PBA01_Raw",
  table  = "fact_befolkning_Viborg_raw"
)

cat("  Tjekker adgang til schema 'PBA01_Raw' og tabellen via dbExistsTable...\n")

exists_try <- try(
```

```
  dbExistsTable(con, table_id),
  silent = TRUE
)

if (inherits(exists_try, "try-error")) {
  dbDisconnect(con)
  stop(
    "Kan ikke tilgå 'PBA01_Raw.fact_befolkning_Viborg_raw'.\n",
    "Tjek at schemaet 'PBA01_Raw' findes, og at brugeren har rettigheder."
  )
}

tabel_findes <- isTRUE(exists_try)

if (!tabel_findes) {
  # ------------------------------------------------------------------
  # Tabel findes ikke → første fulde load
  # ------------------------------------------------------------------
  cat("  Første load: Tabel findes ikke, opretter og indsætter alle rækker...\n")

  dbWriteTable(
    conn      = con,
    name      = table_id,
    value     = befolkning_viborg_raw,
    overwrite = FALSE,
    append    = FALSE
  )

  antal_rækker  <- nrow(befolkning_viborg_raw)
  antal_værdier <- nrow(befolkning_viborg_raw) * ncol(befolkning_viborg_raw)

  cat("  Første load gennemført.\n")
  cat("  Antal rækker indsat:", antal_rækker, "\n")
  cat("  Antal værdier i tabellen:", antal_værdier, "\n\n")

} else {
  # ------------------------------------------------------------------
  # Tabel findes → incremental update (ingen dupletter)
  # ------------------------------------------------------------------
  cat("  Tabel findes - laver incremental update.\n")

  # dbplyr-reference til tabellen
```

```r
fact_tbl <- tbl(
  con,
  in_schema("PBA01_Raw", "fact_befolkning_Viborg_raw")
)

# hent et lille sample til at identificere Tid-kolonnen i SQL
sample_sql <- fact_tbl |>
  head(5) |>
  collect()

# find kolonnen der ligner YYYYKQ, fx 2008K1
tid_kolonne_sql <- names(sample_sql)[sapply(sample_sql, function(col) {
  any(grepl("^[0-9]{4}K[1-4]$", as.character(col)))
})][1]

if (is.na(tid_kolonne_sql) || tid_kolonne_sql == "") {
  dbDisconnect(con)
  stop("Kunne ikke identificere Tid-kolonnen i SQL-tabellen (mønster YYYYKQ).")
}

# find seneste kvartal i databasen (max på tid-kolonnen, uden NA-advarsel)
seneste_tid_df <- fact_tbl |>
  summarise(max_tid = max(.data[[tid_kolonne_sql]], na.rm = TRUE)) |>
  collect()

seneste_tid <- seneste_tid_df$max_tid[1]
cat("Sidst kendte kvartal i databasen:", seneste_tid, "\n")

# find tilsvarende Tid-kolonne i R-datasættet
sample_r <- head(befolkning_viborg_raw, 5)
tid_kolonne_r <- names(sample_r)[sapply(sample_r, function(col) {
  any(grepl("^[0-9]{4}K[1-4]$", as.character(col)))
})][1]

if (is.na(tid_kolonne_r) || tid_kolonne_r == "") {
  dbDisconnect(con)
  stop("Kunne ikke identificere Tid-kolonnen i R-datasættet (mønster YYYYKQ).")
}

# vælg nye rækker i R (Tid > seneste_tid)
if (is.na(seneste_tid)) {
  cat("  Tabellen findes men er tom - indsætter alle rækker fra API.\n")
```

```
    nye_rækker <- befolkning_viborg_raw
  } else {
    nye_rækker <- befolkning_viborg_raw |>
      filter(.data[[tid_kolonne_r]] > seneste_tid)
  }

  if (nrow(nye_rækker) == 0) {
    cat("  Ingen nye kvartaler at indsætte - alt er allerede i databasen.\n\n")
  } else {
    cat("  Indsætter kun nye rækker (antal):", nrow(nye_rækker), "\n")

    dbWriteTable(
      conn      = con,
      name      = table_id,
      value     = nye_rækker,
      append    = TRUE,
      overwrite = FALSE
    )

    cat("  Incremental insert gennemført.\n\n")
  }
}

# ------------------------------------------------------------------------
# Luk forbindelsen pænt
# ------------------------------------------------------------------------

dbDisconnect(con)
cat("  Forbindelsen til Azure SQL er nu lukket.\n")
```

1.2 PBA02_Clean – Transformation og klargøring af data

Fra og med PB02 påbegyndes den egentlige datatransformation og -behandling. Data trækkes direkte fra et adgangsbegrænset og betalt SQL-baseret datalager, hvorfor koden ikke kan afvikles via Quarto-rendering uden særskilt systemadgang. Dokumentationen fokuserer derfor på at påvise udførte transformationer gennem strukturelle outputs og screenshots. I denne fase renses data, datatyper behandles og standardiseres, og datasættene klargøres til videre joins og analyse.

1.3 Dimension vejrkode transformation

1.3.0.1 Pakker og miljø

Dette afsnit indlæser de nødvendige pakker til databaseadgang og datamanipulation. `pacman` anvendes for at sikre reproducerbarhed og et rent konsol-output uden unødvendig støj.

```
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(
    DBI, odbc, dplyr, tibble, stringr
  )
}); cat("Pakker klar og loaded \n")
```

1.3.0.2 Azure-forbindelse

Her oprettes forbindelsen til Azure SQL-databasen. Loginoplysninger hentes fra `.Renviron`, så credentials ikke indgår direkte i koden og løsningen forbliver sikker og reproducerbar.

```
# Drift-/ETL-kode (køres ikke ved render)

# ======================================================================
# Azure forbindelse
# ======================================================================

# Vi forbinder til den samme Azure-database som i de øvrige scripts.
# Alle credentials hentes fra .Renviron, så der ikke står passwords i koden.

con <- dbConnect(
  odbc::odbc(),
  driver   = "ODBC Driver 18 for SQL Server",
  server   = Sys.getenv("AZURE_SQL_SERVER"),
  database = Sys.getenv("AZURE_SQL_DB"),
  uid      = Sys.getenv("AZURE_SQL_UID"),
  pwd      = Sys.getenv("AZURE_SQL_PWD"),
  port     = 1433,
  Encrypt  = "yes",
  TrustServerCertificate = "no",
  ConnectionTimeout      = 30
); cat("Forbundet til:", Sys.getenv("AZURE_SQL_DB"), "\n\n")
```

1.3.0.3 Indlæs RAW-dimension fra PBA01_Raw.dim_weather_code_raw

Her hentes weather-kode-dimensionen direkte fra RAW-laget i Azure SQL (PBA01_Raw). Datasættet danner input til CLEAN-laget, hvor koderne efterfølgende standardiseres og forsynes med danske beskrivelser.

```
# Drift-/ETL-kode (køres ikke ved render)



# ======================================================================
# Hent RAW-dimension fra PBA01_Raw
# ======================================================================

# RAW-tabellen indeholder:
#   - weather_code     → DMI's vejrkode (0-199)
#   - weather_text_en  → engelsk beskrivelse
#
# I CLEAN-laget ønsker vi:
#   - vejrkode         (samme kode)
#   - vejrbeskrivelse  (REN dansk tekst, ingen engelsk)
dim_weather_code_raw <- dbReadTable(
  con,
  DBI::Id(schema = "PBA01_Raw", table = "dim_weather_code_raw")
) |>
  select(vejrkode = weather_code) |>
  arrange(vejrkode)

cat("Antal koder i RAW:", nrow(dim_weather_code_raw), "\n\n")
```

1.3.0.4 Opslagstabel for danske vejrbeskrivelser (CLEAN)

De danske oversættelser af DMI's engelske vejrbeskrivelser er udarbejdet manuelt med støtte fra GitHub Copilot som sprogligt hjælpemiddel. MS Copilot er udelukkende anvendt til formulering og konsistens i teksten – den faglige fortolkning og den endelige validering af hver vejrkode er foretaget manuelt.

```
# ======================================================================
# Opslagstabel med danske oversættelser
# ======================================================================
# Her hardcoder vi EN entydig dansk tekst pr. vejrkode (0-199).
# Denne tabel er "sandheden" i CLEAN-laget og gør, at vi ikke længere
# er afhængige af den engelske tekst fra dokumentationen.
```

```r
vejr_oversaettelse <- tibble::tribble(
  ~vejrkode, ~vejrbeskrivelse,

  0,  "Skyudvikling ikke observeret eller kan ikke observeres - karakteristisk ændring i himl
  1,  "Skyer er generelt ved at opløses eller blive mindre udviklede - karakteristisk ændring
  2,  "Himlens tilstand stort set uændret i den seneste time",
  3,  "Skyer er generelt ved at dannes eller udvikles - karakteristisk ændring i himlens tils
  4,  "Sigtbarhed reduceret af røg (fx steppe- eller skovbrande, industrirøg eller vulkansk a
  5,  "Dis",
  6,  "Udbredt støv i luften, ikke løftet af vinden ved eller nær stationen på observationsti
  7,  "Støv eller sand løftet af vinden ved eller nær stationen, men ingen veludviklede støv
  8,  "Veludviklede støv- eller sandhvirvler set ved eller nær stationen i den seneste time e
  9,  "Støv- eller sandstorm i sigte på observationstidspunktet eller ved stationen i den sen

  10, "Tåge",
  11, "Plettet lav tåge eller is-tåge ved stationen (på land   ca. 2 m, til søs   ca. 10 m)",
  12, "Mere eller mindre sammenhængende lav tåge eller is-tåge ved stationen (på land   ca. 2
  13, "Lyn synligt, men ingen torden høres",
  14, "Nedbør i sigte, men når ikke jordoverfladen eller havoverfladen",
  15, "Nedbør i sigte, når jorden eller havet, men fjernt (anslået > 5 km fra stationen)",
  16, "Nedbør i sigte, når jorden eller havet, tæt på men ikke ved selve stationen",
  17, "Tordenvejr, men ingen nedbør på observationstidspunktet",
  18, "Byger (squalls) ved eller i nærheden af stationen i den seneste time eller på observat
  19, "Skyløb/funnel cloud (tornado- eller vandsky) ved eller i nærheden af stationen i den s

  20, "Støvregn (ikke underafkølet) eller snefnug-korn, ikke som byger",
  21, "Regn (ikke underafkølet), ikke som byger",
  22, "Sne, ikke som byger",
  23, "Regn og sne eller iskorn, ikke som byger",
  24, "Underafkølet støvregn eller underafkølet regn, ikke som byger",
  25, "Regnbyge(r)",
  26, "Snebyge(r) eller byge(r) af regn og sne",
  27, "Haglbyge(r) eller byge(r) af regn og hagl",
  28, "Tåge eller is-tåge",
  29, "Tordenvejr (med eller uden nedbør)",

  30, "Svag eller moderat støv- eller sandstorm - aftaget i den seneste time",
  31, "Svag eller moderat støv- eller sandstorm - ingen væsentlig ændring i den seneste time
  32, "Svag eller moderat støv- eller sandstorm - begyndt eller tiltager i den seneste time"
  33, "Kraftig støv- eller sandstorm - aftaget i den seneste time",
  34, "Kraftig støv- eller sandstorm - ingen væsentlig ændring i den seneste time",
  35, "Kraftig støv- eller sandstorm - begyndt eller tiltager i den seneste time",
```

```
36, "Svag eller moderat føget sne, generelt lavt (under øjenhøjde)",
37, "Kraftig føgesne, generelt lavt (under øjenhøjde)",
38, "Svag eller moderat føget sne, generelt højt (over øjenhøjde)",
39, "Kraftig føgesne, generelt højt (over øjenhøjde)",

40, "Tåge eller is-tåge i det fjerne på observationstidspunktet, men ikke ved stationen i d
41, "Tåge eller is-tåge i banker/pletter",
42, "Tåge eller is-tåge, himlen synlig - er blevet tyndere i den seneste time",
43, "Tåge eller is-tåge, himlen usynlig - er blevet tyndere i den seneste time",
44, "Tåge eller is-tåge, himlen synlig - ingen væsentlig ændring i den seneste time",
45, "Tåge eller is-tåge, himlen usynlig - ingen væsentlig ændring i den seneste time",
46, "Tåge eller is-tåge, himlen synlig - er begyndt eller blevet tættere i den seneste time
47, "Tåge eller is-tåge, himlen usynlig - er begyndt eller blevet tættere i den seneste tim
48, "Tåge, der afsætter rim, himlen synlig",
49, "Tåge, der afsætter rim, himlen usynlig",

50, "Støvregn, ikke underafkølet, periodisk - svag på observationstidspunktet",
51, "Støvregn, ikke underafkølet, vedvarende - svag på observationstidspunktet",
52, "Støvregn, ikke underafkølet, periodisk - moderat på observationstidspunktet",
53, "Støvregn, ikke underafkølet, vedvarende - moderat på observationstidspunktet",
54, "Støvregn, ikke underafkølet, periodisk - kraftig (tæt) på observationstidspunktet",
55, "Støvregn, ikke underafkølet, vedvarende - kraftig (tæt) på observationstidspunktet",
56, "Underafkølet støvregn, svag",
57, "Underafkølet støvregn, moderat eller kraftig (tæt)",
58, "Støvregn og regn, svag",
59, "Støvregn og regn, moderat eller kraftig",

60, "Regn, ikke underafkølet, periodisk - svag på observationstidspunktet",
61, "Regn, ikke underafkølet, vedvarende - svag på observationstidspunktet",
62, "Regn, ikke underafkølet, periodisk - moderat på observationstidspunktet",
63, "Regn, ikke underafkølet, vedvarende - moderat på observationstidspunktet",
64, "Regn, ikke underafkølet, periodisk - kraftig på observationstidspunktet",
65, "Regn, ikke underafkølet, vedvarende - kraftig på observationstidspunktet",
66, "Underafkølet regn, svag",
67, "Underafkølet regn, moderat eller kraftig (tæt)",
68, "Regn eller støvregn og sne, svag",
69, "Regn eller støvregn og sne, moderat eller kraftig",

70, "Periodisk snefald - svagt på observationstidspunktet",
71, "Vedvarende snefald - svagt på observationstidspunktet",
72, "Periodisk snefald - moderat på observationstidspunktet",
73, "Vedvarende snefald - moderat på observationstidspunktet",
```

```
74, "Periodisk snefald - kraftigt på observationstidspunktet",
75, "Vedvarende snefald - kraftigt på observationstidspunktet",
76, "Iskrystaller (diamond dust) med eller uden tåge",
77, "Snefnug-korn (snow grains) med eller uden tåge",
78, "Isolerede stjerneformede snekrystaller med eller uden tåge",
79, "Iskorn (ice pellets)",

80, "Regnbyge(r), svag",
81, "Regnbyge(r), moderat eller kraftig",
82, "Regnbyge(r), meget kraftig/voldsom",
83, "Byge(r) af regn og sne blandet, svag",
84, "Byge(r) af regn og sne blandet, moderat eller kraftig",
85, "Snebyge(r), svag",
86, "Snebyge(r), moderat eller kraftig",
87, "Byge(r) af snefnug-korn eller små hagl, med eller uden regn eller regn/sne blandet - s
88, "Byge(r) af snefnug-korn eller små hagl, med eller uden regn eller regn/sne blandet - m
89, "Haglbyge(r), med eller uden regn eller regn/sne blandet, ikke forbundet med torden - s
90, "Haglbyge(r), med eller uden regn eller regn/sne blandet, ikke forbundet med torden - m

91, "Svag regn på observationstidspunktet - tordenvejr i den seneste time men ikke nu",
92, "Moderat eller kraftig regn på observationstidspunktet - tordenvejr i den seneste time
93, "Svag sne, regn og sne blandet eller hagl på observationstidspunktet - tordenvejr i den
94, "Moderat eller kraftig sne, regn og sne blandet eller hagl på observationstidspunktet -
95, "Tordenvejr, svagt eller moderat, uden hagl men med regn og/eller sne på observationsti
96, "Tordenvejr, svagt eller moderat, med hagl på observationstidspunktet",
97, "Tordenvejr, kraftigt, uden hagl men med regn og/eller sne på observationstidspunktet"
98, "Tordenvejr sammen med støv- eller sandstorm på observationstidspunktet",
99, "Tordenvejr, kraftigt, med hagl på observationstidspunktet",

100, "Ingen væsentligt vejr observeret",
101, "Skyer er generelt ved at opløses eller blive mindre udviklede i den seneste time",
102, "Himlens tilstand på det hele uændret i den seneste time",
103, "Skyer er generelt ved at dannes eller udvikles i den seneste time",
104, "Dis eller røg, eller støv i luften - sigtbarhed  1 km",
105, "Dis eller røg, eller støv i luften - sigtbarhed < 1 km",
106, "Reserveret kode (ikke anvendt)",
107, "Reserveret kode (ikke anvendt)",
108, "Reserveret kode (ikke anvendt)",
109, "Reserveret kode (ikke anvendt)",
110, "Tåge",
111, "Iskrystaller (diamond dust)",
112, "Fjernt lyn",
```

```
113, "Reserveret kode (ikke anvendt)",
114, "Reserveret kode (ikke anvendt)",
115, "Reserveret kode (ikke anvendt)",
116, "Reserveret kode (ikke anvendt)",
117, "Reserveret kode (ikke anvendt)",
118, "Byger (squalls)",
119, "Reserveret kode (ikke anvendt)",
120, "Tåge",
121, "Nedbør",
122, "Støvregn (ikke underafkølet) eller snefnug-korn",
123, "Regn (ikke underafkølet)",
124, "Sne",
125, "Underafkølet støvregn eller underafkølet regn",
126, "Tordenvejr (med eller uden nedbør)",
127, "Føget eller drivende sne eller sand",
128, "Føget eller drivende sne eller sand, sigtbarhed  1 km",
129, "Føget eller drivende sne eller sand, sigtbarhed < 1 km",
130, "Tåge",
131, "Tåge eller is-tåge i banker/pletter",
132, "Tåge eller is-tåge - er blevet tyndere i den seneste time",
133, "Tåge eller is-tåge - ingen væsentlig ændring i den seneste time",
134, "Tåge eller is-tåge - er begyndt eller blevet tættere i den seneste time",
135, "Tåge, der afsætter rim",
136, "Reserveret kode (ikke anvendt)",
137, "Reserveret kode (ikke anvendt)",
138, "Reserveret kode (ikke anvendt)",
139, "Reserveret kode (ikke anvendt)",
140, "Nedbør",
141, "Nedbør, svag eller moderat",
142, "Nedbør, kraftig",
143, "Flydende nedbør, svag eller moderat",
144, "Flydende nedbør, kraftig",
145, "Fast nedbør, svag eller moderat",
146, "Fast nedbør, kraftig",
147, "Underafkølet nedbør, svag eller moderat",
148, "Underafkølet nedbør, kraftig",
149, "Reserveret kode (ikke anvendt)",
150, "Støvregn",
151, "Støvregn, ikke underafkølet, svag",
152, "Støvregn, ikke underafkølet, moderat",
153, "Støvregn, ikke underafkølet, kraftig",
154, "Underafkølet støvregn, svag",
```

```
155, "Underafkølet støvregn, moderat",
156, "Underafkølet støvregn, kraftig",
157, "Støvregn og regn, svag",
158, "Støvregn og regn, moderat eller kraftig",
159, "Reserveret kode (ikke anvendt)",
160, "Regn",
161, "Regn, ikke underafkølet, svag",
162, "Regn, ikke underafkølet, moderat",
163, "Regn, ikke underafkølet, kraftig",
164, "Underafkølet regn, svag",
165, "Underafkølet regn, moderat",
166, "Underafkølet regn, kraftig",
167, "Regn (eller støvregn) og sne, svag",
168, "Regn (eller støvregn) og sne, moderat eller kraftig",
169, "Reserveret kode (ikke anvendt)",
170, "Sne",
171, "Sne, svag",
172, "Sne, moderat",
173, "Sne, kraftig",
174, "Iskorn, svage",
175, "Iskorn, moderate",
176, "Iskorn, kraftige",
177, "Snefnug-korn",
178, "Iskrystaller",
179, "Reserveret kode (ikke anvendt)",
180, "Byger eller periodisk nedbør",
181, "Regnbyge(r) eller periodisk regn, svag",
182, "Regnbyge(r) eller periodisk regn, moderat",
183, "Regnbyge(r) eller periodisk regn, kraftig",
184, "Regnbyge(r) eller periodisk regn, meget kraftig/voldsom",
185, "Snebyge(r) eller periodisk sne, svag",
186, "Snebyge(r) eller periodisk sne, moderat",
187, "Snebyge(r) eller periodisk sne, kraftig",
188, "Reserveret kode (ikke anvendt)",
189, "Hagl",
190, "Tordenvejr",
191, "Tordenvejr, svagt eller moderat, uden nedbør",
192, "Tordenvejr, svagt eller moderat, med regn- og/eller snebyger",
193, "Tordenvejr, svagt eller moderat, med hagl",
194, "Tordenvejr, kraftigt, uden nedbør",
195, "Tordenvejr, kraftigt, med regn- og/eller snebyger",
196, "Tordenvejr, kraftigt, med hagl",
```

```
  197, "Reserveret kode (ikke anvendt)",
  198, "Reserveret kode (ikke anvendt)",
  199, "Tornado"
)

cat("Antal koder i oversættelsestabel:", nrow(vejr_oversaettelse), "\n\n")
```

1.3.0.5 Sammensmeltning af RAW-vejrkoder med danske beskrivelser

Her sammenkobles RAW-dimensionen fra PBA01_Raw med den danske oversættelsestabel for at danne en fuld CLEAN-dimension. Der udføres samtidig et kvalitetstjek for at sikre, at alle vejrkoder har en entydig dansk beskrivelse.

```
# Drift-/ETL-kode (køres ikke ved render)



# ======================================================================
# Join RAW-koder med dansk tekst → CLEAN-dimension
# ======================================================================
dim_vejrkode_clean <- dim_weather_code_raw |>
  left_join(vejr_oversaettelse, by = "vejrkode")

# Kvalitetstjek: findes der koder i RAW uden dansk oversættelse?
mangler_da <- dim_vejrkode_clean |>
  filter(is.na(vejrbeskrivelse))

cat("Antal koder uden dansk tekst:", nrow(mangler_da), "\n\n")
if (nrow(mangler_da) > 0) {
  cat("  Følgende vejrkoder mangler oversættelse:\n")
  print(mangler_da$vejrkode)
}; glimpse(dim_vejrkode_clean);tibble(dim_vejrkode_clean)
```

dim_vejrkode_clean

Filter

	vejrkode	vejrbeskrivelse
1	0	Skyudvikling ikke observeret eller kan ikke observeres – kara...
2	1	Skyer er generelt ved at opløses eller blive mindre udviklede...
3	2	Himlens tilstand stort set uændret i den seneste time
4	3	Skyer er generelt ved at dannes eller udvikles – karakteristis...
5	4	Sigtbarhed reduceret af røg (fx steppe- eller skovbrande, in...
6	5	Dis
7	6	Udbredt støv i luften, ikke løftet af vinden ved eller nær stati...
8	7	Støv eller sand løftet af vinden ved eller nær stationen, men ...
9	8	Veludviklede støv- eller sandhvirvler set ved eller nær statio...
10	9	Støv- eller sandstorm i sigte på observationstidspunktet elle...
11	10	Tåge
12	11	Plettet lav tåge eller is-tåge ved stationen (på land ≤ ca. 2 m...
13	12	Mere eller mindre sammenhængende lav tåge eller is-tåge v...
14	13	Lyn synligt, men ingen torden høres
15	14	Nedbør i sigte, men når ikke jordoverfladen eller havoverfla...
16	15	Nedbør i sigte, når jorden eller havet, men fjernt (anslået > ...
17	16	Nedbør i sigte, når jorden eller havet, tæt på men ikke ved s...
18	17	Tordenvejr, men ingen nedbør på observationstidspunktet
19	18	Byger (squalls) ved eller i nærheden af stationen i den senes...
20	19	Skyløb/funnel cloud (tornado- eller vandsky) ved eller i nær...
21	20	Støvregn (ikke underafkølet) eller snefnug-korn, ikke som by...

1.3.0.6 Load til PBA02_Clean

Den færdigbearbejdede vejrkode-dimension gemmes nu i CLEAN-laget som PBA02_Clean.dim_vejrkode. Tabellen overskrives ved kørsel, så CLEAN-laget altid afspejler den seneste, validerede version.

```
# Drift-/ETL-kode (køres ikke ved render)

# ======================================================================
# Skriv CLEAN-dimension til PBA02_Clean
# ======================================================================

# Vi lægger nu den færdige dimension i CLEAN-skemaet:
```

```
#   PBA02_Clean.dim_vejrkode
#
# Schema + tabelnavn angives eksplicit med DBI::Id.
DBI::dbWriteTable(
  conn      = con,
  name      = DBI::Id(schema = "PBA02_Clean", table = "dim_vejrkode"),
  value     = dim_vejrkode_clean,
  overwrite = TRUE
)

cat("dim_vejrkode er nu uploadet til PBA02_Clean.dim_vejrkode  \n\n")
```

1.3.0.7 Afslutning og oprydning

Til sidst lukkes databaseforbindelsen korrekt for at frigive ressourcer og sikre en ren afslutning på scriptet.

```
# Drift-/ETL-kode (køres ikke ved render)

# ======================================================================
# Luk forbindelse
# ======================================================================
dbDisconnect(con)
cat("  Forbindelse lukket.\n")
```

1.4 Dimension klubinfo transformation

I dette afsnit hentes klubinformation fra SQL-databasen og transformeres med henblik på at identificere og korrigere eventuelle fejl eller inkonsistenser. Formålet er at sikre et konsistent og pålideligt datagrundlag til det videre forløb, da klubdata – herunder klubID – udgør en central nøgle, som anvendes på tværs af flere analyser i projektet.

1.4.0.1 Pakker

I dette afsnit indlæses de nødvendige R-pakker til databaseforbindelse og datamanipulation. suppressPackageStartupMessages() anvendes for at holde konsol-output ryddeligt og fokuseret.

```
# ============================================
# Pakker
# ============================================
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(DBI, odbc, dplyr, tibble, stringr)
}); cat("Pakker klar og loaded \n")
```

1.4.0.2 Forbindelse til Azure SQL

Her oprettes forbindelsen til projektets Azure SQL-database ved brug af ODBC. Logino-plysninger hentes fra .Renviron for at sikre en sikker og reproducerbar opsætning uden cre-dentials i koden.

```
# Drift-/ETL-kode (køres ikke ved render)

# ================================================
# Azure forbindelse
# ================================================
con <- dbConnect(
  odbc::odbc(),
  driver   = "ODBC Driver 18 for SQL Server",
  server   = Sys.getenv("AZURE_SQL_SERVER"),
  database = Sys.getenv("AZURE_SQL_DB"),
  uid      = Sys.getenv("AZURE_SQL_UID"),
  pwd      = Sys.getenv("AZURE_SQL_PWD"),
  port     = 1433,
  Encrypt  = "yes",
  TrustServerCertificate = "no",
  ConnectionTimeout      = 30
)
cat("Forbundet til:", Sys.getenv("AZURE_SQL_DB"), "\n\n")
```

1.4.0.3 Hent RAW-klubdimension fra Azure (PBA01_Raw)

I dette trin indlæses den rå klubdimension direkte fra RAW-laget i Azure SQL. Datasættet danner grundlag for efterfølgende rensning og standardisering af kluboplysninger, som anven-des bredt i analyserne.

```r
# Drift-/ETL-kode (køres ikke ved render)

dim_vff_klubber_raw <- dbReadTable(
  con,
  DBI::Id(schema = "PBA01_Raw", table = "dim_vff_klubber_raw")
)

cat("Rækker i dim_vff_klubber_raw:", nrow(dim_vff_klubber_raw), "\n\n")
glimpse(dim_vff_klubber_raw)
```

```
> cat("Rækker i dim_vff_klubber_raw:", nrow(dim_vff_klubber_raw), "\n\n")
Rækker i dim_vff_klubber_raw: 47

> glimpse(dim_vff_klubber_raw)
Rows: 47
Columns: 6
$ kort                <chr> "AB", "ACH", "AGF", "AHF", "AMA", "B.93", "B1909", "B1913", "BKF", "BKS", …
$ langt               <chr> "AB", "AC Horsens", "AGF", "Aarhus Fremad", "Fremad Amager", "B.93", "B190…
$ navn                <chr> "Akademisk Boldklub Gladsaxe", "Alliance Club Horsens", "Aarhus Gymnastikf…
$ stadion             <chr> "Gladsaxe Stadion", "Horsens Idrætspark)", "Vejlby Stadium", "Riisvangen S…
$ sponsor_stadion_navn <chr> "Gladsaxe Stadion", "Nordstern Arena Horsens", "Vejlby Stadium", "Riisvang…
$ tilskuere           <dbl> 13507, 10495, 12000, 5000, 7200, 4400, 6000, 2000, 12000, 4400, NA, 3000, …
```

1.4.0.4 Opbygning af CLEAN-klubdimension (stabilt KLUB-ID)

Her transformeres RAW-klubdata til en renset og konsistent klubdimension med et stabilt, entydigt klub-ID. Dimensionen standardiserer navne- og stadionoplysninger og fungerer som fælles reference på tværs af analyser og joins.

```r
# Drift-/ETL-kode (køres ikke ved render)

# ============================================================================
# 3) Byg CLEAN-dimension med stabilt KLUB-ID
# ============================================================================
dim_vff_klubber_clean <- dim_vff_klubber_raw %>%
  arrange(kort) %>%                              # stabil rækkefølge
  mutate(
    klub_nr_løbende = row_number(),             # 1, 2, 3, ...
    klub_id         = paste0(
      "KLUB",
      stringr::str_pad(klub_nr_løbende, width = 3, pad = "0")
    ),
    klub_kort          = kort,
    klub_lang          = langt,
```

```
    klub_navn_officiel = navn,
    stadion_navn       = stadion,
    stadion_sponsor    = sponsor_stadion_navn,
    stadion_kapacitet  = tilskuere
  ) %>%
  select(
    klub_id,
    klub_kort,
    klub_lang,
    klub_navn_officiel,
    stadion_navn,
    stadion_sponsor,
    stadion_kapacitet
  )

cat("CLEAN-klubber før ekstra klubber:", nrow(dim_vff_klubber_clean), "\n\n")
view(dim_vff_klubber_clean)
```

```
> cat("CLEAN-klubber før ekstra klubber:", nrow(dim_vff_klubber_clean), "\n\n")
CLEAN-klubber før ekstra klubber: 47
```

1.4.0.5 Validering mod Superliga-program (manglende klubber og inkon)

Her hentes Superliga-kampprogrammet fra PBA02_Clean og bruges som reference til at validere, om alle hjemme-/udeklubkoder findes i dim_vff_klubber_clean. Outputtet identificerer eventuelle manglende klubkoder, så klubdimensionen kan suppleres før næste join-/analyselag.

```
# Drift-/ETL-kode (køres ikke ved render)

# ======================================================================
# Sammenlign med Superliga-programmet og find manglende klubkoder
# ======================================================================
fact_sl_clean <- dbReadTable(
  con,
  DBI::Id(schema = "PBA02_Clean", table = "fact_superliga_program_clean")
)

cat("Rækker i fact_superliga_program_clean:", nrow(fact_sl_clean), "\n\n")

alle_koder_program <- union(
```

```
  unique(fact_sl_clean$hjemmehold),
  unique(fact_sl_clean$udehold)
)
alle_koder_program <- alle_koder_program[!is.na(alle_koder_program)]

manglende_koder <- setdiff(alle_koder_program, dim_vff_klubber_clean$klub_kort)
manglende_koder <- sort(manglende_koder)

cat("Antal klubkoder i Superliga-programmet:", length(alle_koder_program), "\n")
cat("Antal klubkoder der mangler i dim_vff_klubber_clean:", length(manglende_koder), "\n\n")
print(manglende_koder)
```

```
> cat("Antal klubkoder i Superliga-programmet:", length(alle_koder_program), "\n")
Antal klubkoder i Superliga-programmet: 26
> cat("Antal klubkoder der mangler i dim_vff_klubber_clean:", length(manglende_koder), "\n\n")
Antal klubkoder der mangler i dim_vff_klubber_clean: 2

> print(manglende_koder)
[1] "KBK" "SJF"
```

1.4.0.6 Tilføj dummy-klubber (supplering af manglende koder)

Hvis der findes klubkoder i Superliga-programmet, som ikke ligger i dim_vff_klubber_clean, oprettes der midlertidige "dummy"-rækker med nye KLUB-id'er. Det sikrer fuld dækning i joins, så kampprogram og andre tabeller ikke mister rækker pga. manglende klubdimension.

```
# Drift-/ETL-kode (køres ikke ved render)

# ================================================================================
# Opret dummy-klubber for manglende koder (fx KBK)
# ================================================================================
if (length(manglende_koder) > 0) {

  # start næste løbenummer efter de eksisterende
  næste_nr <- nrow(dim_vff_klubber_clean) + 1L

  supplerende_klubber <- tibble(
    klub_nr_løbende   = seq_len(length(manglende_koder)) + (næste_nr - 1L),
    klub_id           = paste0(
      "KLUB",
      stringr::str_pad(klub_nr_løbende, width = 3, pad = "0")
    ),
    klub_kort         = manglende_koder,
```

```
    klub_lang          = manglende_koder,   # indtil vi evt. ved mere
    klub_navn_officiel= manglende_koder,   # indtil vi evt. ved mere
    stadion_navn       = NA_character_,
    stadion_sponsor    = NA_character_,
    stadion_kapacitet = NA_integer_
  ) %>%
    select(
      klub_id,
      klub_kort,
      klub_lang,
      klub_navn_officiel,
      stadion_navn,
      stadion_sponsor,
      stadion_kapacitet
    )

  cat("Tilføjer", nrow(supplerende_klubber), "dummy-klub(ber):\n")
  print(supplerende_klubber)

  dim_vff_klubber_clean <- bind_rows(
    dim_vff_klubber_clean,
    supplerende_klubber
  )
}

cat("\nSamlet antal klubber i CLEAN-dimension:", nrow(dim_vff_klubber_clean), "\n\n")
glimpse(dim_vff_klubber_clean)
```

```
Tilføjer 2 dummy-klub(ber):
>
> cat("\nSamlet antal klubber i CLEAN-dimension:", nrow(dim_vff_klubber_clean), "\n\n")

Samlet antal klubber i CLEAN-dimension: 49
```

```
> glimpse(dim_vff_klubber_clean)
Rows: 49
Columns: 7
$ klub_id           <chr> "KLUB001", "KLUB002", "KLUB003", "KLUB004", "KLUB005", "KLUB006", "KLUB007", "KLUB008", "K…
$ klub_kort         <chr> "AB", "ACH", "AGF", "AHF", "AMA", "AaB", "B.93", "B1909", "B1913", "BIF", "BKF", "BKS", "B…
$ klub_lang         <chr> "AB", "AC Horsens", "AGF", "Aarhus Fremad", "Fremad Amager", "AaB", "B.93", "B1909", "B191…
$ klub_navn_officiel <chr> "Akademisk Boldklub Gladsaxe", "Alliance Club Horsens", "Aarhus Gymnastikforening", "Aarhu…
$ stadion_navn      <chr> "Gladsaxe Stadion", "Horsens Idrætspark)", "Vejlby Stadium", "Riisvangen Stadion", "Sundby…
$ stadion_sponsor   <chr> "Gladsaxe Stadion", "Nordstern Arena Horsens", "Vejlby Stadium", "Riisvangen Stadion", "Su…
$ stadion_kapacitet <dbl> 13507, 10495, 12000, 5000, 7200, 13600, 4400, 6000, 2000, 28000, 12000, 4400, NA, 3000, 40…
```

1.4.0.7 Upload af CLEAN-klubdimension og afslutning

Den færdige klubdimension skrives nu til PBA02_Clean.dim_vff_klubber_clean, så alle klubber har stabile og konsistente KLUB-ID'er. Herefter lukkes forbindelsen til Azure SQL.

```
# Drift-/ETL-kode (køres ikke ved render)

# ============================================================================
# Upload til PBA02_Clean.dim_vff_klubber_clean + luk forbindelse
# ============================================================================
dbWriteTable(
  conn      = con,
  name      = DBI::Id(schema = "PBA02_Clean", table = "dim_vff_klubber_clean"),
  value     = dim_vff_klubber_clean,
  overwrite = TRUE
)

cat("dim_vff_klubber_clean er nu uploadet til PBA02_Clean med opdaterede klubkoder og KLUB-I

dbDisconnect(con)
cat("  Forbindelse lukket.\n")
```

1.5 Superliga-program: PBA01_Raw → PBA02_Clean Transformation

Her renses Superliga-programmet fra RAW-laget, så kun sæsoner fra og med 2000/2001 medtages. Dato-feltet konverteres og fastholdes som Date, og der oprettes en date_key som tekst i format DDMMÅÅÅÅ til sikre joins i de efterfølgende lag.

1.5.0.1 Pakker og afhængigheder

Dette afsnit indlæser de nødvendige R-pakker til databaseforbindelse, datamanipulation og datohåndtering. Pakkerne sikres installeret og loadet samlet for at gøre scriptet reproducerbart og overskueligt.

```
# ============================================================================
# Pakker
# ============================================================================
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(DBI, odbc, dplyr, tibble, stringr, lubridate, purrr, rlang)
}) ; cat("Pakker klar og loaded \n")
```

1.5.0.2 Miljø- og credential-validering (.Renviron)

Dette afsnit fungerer som en sikkerhedskontrol, der verificerer, at alle nødvendige Azure-miljøvariabler er korrekt sat i .Renviron. Scriptet stopper tidligt med en klar fejl, hvis credentials mangler, for at undgå mislykkede forbindelser senere i pipeline.

```
# Drift-/ETL-kode (køres ikke ved render)

# ========================================================================
# Safeguard: tjek at .Renviron er sat korrekt op
# ========================================================================
required_vars <- c("AZURE_SQL_SERVER", "AZURE_SQL_DB", "AZURE_SQL_UID", "AZURE_SQL_PWD")
values <- Sys.getenv(required_vars)
missing_vars <- required_vars[values == ""]

if (length(missing_vars) > 0) {
  cat("  Følgende miljøvariabler mangler i .Renviron:\n")
  print(missing_vars)
  stop("Udfyld .Renviron og kør scriptet igen.")
}
```

1.5.0.3 Azure SQL-forbindelse (PBA-miljø)

Her oprettes en sikker ODBC-forbindelse til Azure SQL ved brug af credentials fra `.Renviron`. Forbindelsen anvendes til at læse RAW-data og skrive videre til CLEAN-laget.

```
# ========================================================================
# Azure forbindelse
# ========================================================================
con <- DBI::dbConnect(
  odbc::odbc(),
  driver   = "ODBC Driver 18 for SQL Server",
  server   = Sys.getenv("AZURE_SQL_SERVER"),
  database = Sys.getenv("AZURE_SQL_DB"),
  uid      = Sys.getenv("AZURE_SQL_UID"),
  pwd      = Sys.getenv("AZURE_SQL_PWD"),
  port     = 1433,
  Encrypt  = "yes",
  TrustServerCertificate = "no",
  ConnectionTimeout      = 30
)
cat("  Forbundet til:", Sys.getenv("AZURE_SQL_DB"), "\n\n")
```

1.5.0.4 Hent RAW Superliga-program (PBA01_Raw.fact_superliga_program_raw)

I dette trin indlæses Superliga-programmet direkte fra RAW-laget i Azure SQL. Datasættet danner grundlag for efterfølgende rensning, datokonvertering og filtrering til CLEAN-laget.

```
# ======================================================================
# Hent RAW Superliga-program fra PBA01_Raw
# ======================================================================
fact_superliga_program_raw <- DBI::dbReadTable(
  con,
  DBI::Id(schema = "PBA01_Raw", table = "fact_superliga_program_raw")
) %>% as_tibble()

cat("  Rækker i PBA01_Raw.fact_superliga_program_raw:", nrow(fact_superliga_program_raw), "\n
tibble(fact_superliga_program_raw)
```

```
> tibble(fact_superliga_program_raw)
# A tibble: 5,446 × 10
   season_value runde_header runde_nr ugedag_raw dato_tid_raw hold_raw resultat_raw tilskuertal_raw
          <int> <chr>           <int> <chr>      <chr>        <chr>    <chr>        <chr>
 1         2000 Runde 1             1 LÃ¸r      24/07 19:00  BIF-LBK  1-0          9.114
 2         2000 Runde 1             1 SÃ¸n      25/07 15:00  AB-AGF   1-0          2.553
 3         2000 Runde 1             1 SÃ¸n      25/07 15:00  HER-AaB  3-1          1.908
 4         2000 Runde 1             1 SÃ¸n      25/07 15:00  VB-SIF   0-4          3.141
 5         2000 Runde 1             1 SÃ¸n      25/07 17:15  VFF-FCK  2-1          2.168
 6         2000 Runde 1             1 Man      26/07 19:00  EFB- OB  2-0          7.260
 7         2000 Runde 2             2 SÃ¸n      01/08 15:00  VB-BIF   0-4          4.349
 8         2000 Runde 2             2 SÃ¸n      01/08 15:00  SIF-FCK  0-0          5.004
 9         2000 Runde 2             2 SÃ¸n      01/08 15:00  LBK-EFB  5-1          2.171
10         2000 Runde 2             2 SÃ¸n      01/08 15:00  AGF-HER  1-1          2.611
# ℹ 5,436 more rows
# ℹ 2 more variables: dommer_raw <chr>, tv_kanal_raw <chr>
# ℹ Use `print(n = ...)` to see more rows
```

1.5.0.5 Transformér RAW → CLEAN: Superliga-program (PBA01_Raw.fact_superliga_program_raw → PBA02_Clean.fact_superliga_program_clean)

Her standardiserer vi sæsonlogik, dato/tid (dato som Date), ugedag, hold/resultat, tilskuertal samt beregner vinderfelter og opretter date_key. Til sidst filtreres der til sæsoner fra og med 2000/2001, og der udvælges kun de felter, som skal bruges stabilt i CLEAN-laget.

```
# ======================================================================
# Clean: sæson, runde, ugedag, dato (Date), date_key (tekst), tid, hold, mål, vinder, tilsku
# ======================================================================

fact_superliga_program_clean <- fact_superliga_program_raw %>%
```

```
  mutate(
    # Sæson-startår: Superliga-sæson går typisk fra sommer til forår
    # 2000/2001 → season_value = 2001 → start_aar = 2000
    start_aar = if_else(!is.na(season_value) & season_value > 1991L, season_value - 1L, seas

    # Parse dato + tid fra "dd/mm HH:MM"
    dt_match = stringr::str_match(dato_tid_raw, "^(\\d{2})/(\\d{2})\\s+(\\d{2}:\\d{2})$")
  ) %>%
  mutate(
    dag    = suppressWarnings(as.integer(dt_match[, 2])),
    maaned = suppressWarnings(as.integer(dt_match[, 3])),
    klok   = dt_match[, 4],

    # År afhænger af måned: juli-december = start_aar, jan-juni = start_aar + 1
    aar = if_else(
      !is.na(maaned) & maaned >= 7L,
      start_aar,
      start_aar + 1L
    ),

    # Byg rigtig dato og tid
    kamp_dato = suppressWarnings(lubridate::make_date(year = aar, month = maaned, day = dag)
    kamp_tid  = if_else(!is.na(klok) & stringr::str_detect(klok, "^\\d{2}:\\d{2}$"),
                        paste0(klok, ":00"),
                        NA_character_)
  ) %>%
  #  Kun sæsoner fra 2000/2001 og frem
  filter(!is.na(start_aar) & start_aar >= 2000L) %>%
  # Ugedag: fix encoding og brug fulde danske navne
  mutate(
    ugedag = case_when(
      stringr::str_starts(ugedag_raw, "Man") ~ "Mandag",
      stringr::str_starts(ugedag_raw, "Tir") ~ "Tirsdag",
      stringr::str_starts(ugedag_raw, "Ons") ~ "Onsdag",
      stringr::str_starts(ugedag_raw, "Tor") ~ "Torsdag",
      stringr::str_starts(ugedag_raw, "Fre") ~ "Fredag",
      stringr::str_starts(ugedag_raw, "L")   ~ "Lørdag",
      stringr::str_starts(ugedag_raw, "S")   ~ "Søndag",
      TRUE                                   ~ ugedag_raw
    )
  ) %>%
  # Split hold og resultat + tilskuertal
```

```r
  mutate(
    # Split hjemme/ude fra "BIF-AB"
    hold_split = stringr::str_split(hold_raw, "-", n = 2),
    hjemmehold = purrr::map_chr(hold_split, ~ stringr::str_squish(.x[1] %||% NA_character_)),
    udehold    = purrr::map_chr(hold_split, ~ stringr::str_squish(.x[2] %||% NA_character_)),

    # Split mål fra "3-0"
    res_split  = stringr::str_match(resultat_raw, "^(\\d+)-(\\d+)$"),
    mål_hjemme = suppressWarnings(as.integer(res_split[, 2])),
    mål_ude    = suppressWarnings(as.integer(res_split[, 3])),

    # Tilskuertal: fjern punktummer, mellemrum, osv.
    tilskuertal = tilskuertal_raw %>%
      stringr::str_replace_all("[^0-9]", "") %>%
      na_if("") %>%
      suppressWarnings(as.integer())
  ) %>%
  # Beregn vinder og vinder_type
  mutate(
    vinder = case_when(
      mål_hjemme > mål_ude ~ hjemmehold,
      mål_ude    > mål_hjemme ~ udehold,
      TRUE ~ "Uafgjort"
    ),
    vinder_type = case_when(
      mål_hjemme > mål_ude ~ "Hjemmehold",
      mål_ude    > mål_hjemme ~ "Udehold",
      TRUE ~ "Uafgjort"
    )
  ) %>%
  transmute(
    # Sæson som tekst: "2000/2001" og frem
    sæson = paste0(start_aar, "/", start_aar + 1L),

    # Runde som rent tal
    runde = suppressWarnings(as.integer(runde_nr)),

    # Pæn ugedag
    ugedag = ugedag,

    #   dato som Date (ikke tekst)
    dato = kamp_dato,
```

```
    #   date_key som tekst (DDMMÅÅÅÅ)
    date_key = format(kamp_dato, "%d%m%Y"),

    # Tid i HH:MM:SS
    tid = kamp_tid,

    # Hold og mål
    hjemmehold,
    udehold,
    mål_hjemme,
    mål_ude,

    # Vinder + om det er hjemme/ude/uafgjort
    vinder,
    vinder_type,

    # Tilskuere
    tilskuertal,

    # Dommer og tv-kanal videreføres som de er
    dommer   = dommer_raw,
    tv_kanal = tv_kanal_raw
  )

cat("  Færdig-cleanet Superliga-program (rækker):", nrow(fact_superliga_program_clean), "\n\n
tibble(fact_superliga_program_clean)

if (interactive()) View(fact_superliga_program_clean);glimpse(fact_superliga_program_clean)
```

```
> tibble(fact_superliga_program_clean)
# A tibble: 5,248 × 15
   sæson     runde ugedag dato       date_key tid      hjemmehold udehold mål_hjemme mål_ude vinder
   <chr>     <int> <chr>  <date>     <chr>    <chr>    <chr>      <chr>         <int>   <int> <chr>
 1 2000/2001     1 Lørdag 2000-07-22 22072000 15:30:00 BIF        AGF               1       2 AGF
 2 2000/2001     1 Søndag 2000-07-23 23072000 15:00:00 AB         SIF               1       1 Uafgj…
 3 2000/2001     1 Søndag 2000-07-23 23072000 15:00:00 FCM        LBK               4       0 FCM
 4 2000/2001     1 Søndag 2000-07-23 23072000 17:05:00 FCK        SJF               5       1 FCK
 5 2000/2001     1 Mandag 2000-07-24 24072000 19:00:00 OB         VFF               0       1 VFF
 6 2000/2001     1 Onsdag 2000-07-26 26072000 19:00:00 HER        AaB               2       3 AaB
 7 2000/2001     2 Lørdag 2000-07-29 29072000 15:30:00 BIF        OB                0       0 Uafgj…
 8 2000/2001     2 Søndag 2000-07-30 30072000 15:00:00 LBK        HER               3       1 LBK
 9 2000/2001     2 Søndag 2000-07-30 30072000 15:00:00 VFF        AB                1       1 Uafgj…
10 2000/2001     2 Søndag 2000-07-30 30072000 15:00:00 AGF        FCK               1       0 AGF
# i 5,238 more rows
# i 4 more variables: vinder_type <chr>, tilskuertal <chr>, dommer <chr>, tv_kanal <chr>
# i Use `print(n = ...)` to see more rows
```

1.5.0.6 Upload af CLEAN Superliga-program til Azure SQL (PBA02_Clean)

I dette trin skrives det færdigbehandlede Superliga-program til CLEAN-laget i Azure SQL. Tabellen overskrives ved hver kørsel, så PBA02_Clean altid afspejler den seneste og konsistente version af datasættet.

```
# ======================================================================
# Upload til PBA02_Clean.fact_superliga_program_clean
# ======================================================================
DBI::dbWriteTable(
  conn      = con,
  name      = DBI::Id(schema = "PBA02_Clean", table = "fact_superliga_program_clean"),
  value     = fact_superliga_program_clean,
  overwrite = TRUE
)
cat("  fact_superliga_program_clean uploadet til PBA02_Clean  \n")
```

1.5.0.7 Luk forbindelse til Azure SQL

Til sidst lukkes forbindelsen til Azure SQL for at frigive ressourcer og sikre en korrekt afslutning af scriptet.

```
# ======================================================================
# Luk forbindelse
# ======================================================================
DBI::dbDisconnect(con)
cat("  Forbindelse lukket.\n")
```

1.6 DMI vejr obsevationer transformation

CLEAN-faktatabel med én entydig DMI-vejrobservation pr. time, hvor den seneste observation inden for hver time bevares til analysebrug.

1.6.0.1 Pakkeload

Vi loader og sikrer at vi har nødvendige pakker til transformation

```
# ======================================================================
# Pakker
# ======================================================================
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(DBI, odbc, dplyr, tibble, stringr, lubridate)
})

options(scipen = 999)
cat("Pakker er klar  \n\n")
```

1.6.0.2 Azure SQL-forbindelse (CLEAN-lag)

Her oprettes forbindelsen til Azure SQL via ODBC med credentials fra `.Renviron`, så data kan læses og skrives sikkert i CLEAN-laget.

```
# Drift-/ETL-kode (køres ikke ved render)

# ======================================================================
# Azure forbindelse
# ======================================================================
con <- DBI::dbConnect(
  odbc::odbc(),
  driver   = "ODBC Driver 18 for SQL Server",
  server   = Sys.getenv("AZURE_SQL_SERVER"),
  database = Sys.getenv("AZURE_SQL_DB"),
  uid      = Sys.getenv("AZURE_SQL_UID"),
  pwd      = Sys.getenv("AZURE_SQL_PWD"),
  port     = 1433,
  Encrypt  = "yes",
  TrustServerCertificate = "no",
  ConnectionTimeout      = 30
)
```

```
cat("Forbundet til:", Sys.getenv("AZURE_SQL_DB"), " \n\n")
on.exit(try(DBI::dbDisconnect(con), silent = TRUE), add = TRUE)
```

1.6.0.3 Hent RAW vejrdata fra Azure SQL (PBA01_Raw)

I dette trin indlæses de rå vejr-observationer direkte fra PBA01_Raw.fact_weather_raw. Datasættet bruges som grundlag for efterfølgende aggregering og rensning i CLEAN-laget.

```
# Drift-/ETL-kode (køres ikke ved render)

# ======================================================================
# Hent RAW
# ======================================================================
fact_weather_raw <- DBI::dbReadTable(
  con,
  DBI::Id(schema = "PBA01_Raw", table = "fact_weather_raw")
)

cat("Rækker i PBA01_Raw.fact_weather_raw:", nrow(fact_weather_raw), "\n\n")
stopifnot(all(c("observed","value") %in% names(fact_weather_raw)))
glimpse(fact_weather_raw)
```

```
> glimpse(fact_weather_raw)
Rows: 246,648
Columns: 8
$ weather_id     <int64> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 2…
$ stationId      <chr> "06060", "06060", "06060", "06060", "06060", "06060", "06060", "06060", "06060", "06060", "060…
$ parameterId    <chr> "weather", "weather", "weather", "weather", "weather", "weather", "weather", "weather", "weath…
$ observed       <dttm> 2000-01-01 00:00:00, 2000-01-01 03:00:00, 2000-01-01 06:00:00, 2000-01-01 09:00:00, 2000-01-0…
$ value          <dbl> 10, 10, 60, 60, 10, 10, 10, 11, 10, 10, 10, 10, 51, 10, 10, 10, 47, 28, 10, 58, 58, 0, 10, 10,…
$ lon            <dbl> 9.1138, 9.1138, 9.1138, 9.1138, 9.1138, 9.1138, 9.1138, 9.1138, 9.1138, 9.1138, 9.1138, 9.1138…
$ lat            <dbl> 56.2935, 56.2935, 56.2935, 56.2935, 56.2935, 56.2935, 56.2935, 56.2935, 56.2935, 56.2935, 56.2…
$ load_timestamp <dttm> 2025-12-06 18:40:26, 2025-12-06 18:40:26, 2025-12-06 18:40:26, 2025-12-06 18:40:26, 2025-12-0…
```

1.6.0.4 Opbyg CLEAN vejrdata med timesopløsning

Her aggregeres RAW-observationerne til én repræsentativ vejrkode pr. time ved at vælge den senest registrerede observation inden for hver time. Datasættet afgrænses til tidsrummet 08–22 og struktureres til det endelige CLEAN-format.

```
# Drift-/ETL-kode (køres ikke ved render)
# ======================================================================
# Byg CLEAN med 1 observation pr time
# ======================================================================
fact_vejr_dmi <- fact_weather_raw |>
```

```
  mutate(
    # robust datetime
    observed_dt = as.POSIXct(observed, tz = "Europe/Copenhagen"),

    # time-bucket
    observed_hour = lubridate::floor_date(observed_dt, unit = "hour"),

    # vejrkode som int
    vejrkode = as.integer(value),

    # output-format (samme som før, bare HH:00:00)
    obs_dato = format(as.Date(observed_hour), "%d-%m-%Y"),
    tid      = format(observed_hour, "%H:00:00")
  ) |>
  # forretningsfilter: 08-22
  filter(
    tid >= "08:00:00",
    tid <= "22:00:00"
  ) |>
  # vælg seneste observation i timen
  group_by(observed_hour) |>
  slice_max(order_by = observed_dt, n = 1, with_ties = FALSE) |>
  ungroup() |>
  # behold samme kolonner og samme rækkefølge som før
  transmute(
    obs_dato = obs_dato,
    tid      = tid,
    vejrkode = vejrkode
  )

cat("CLEAN fact_vejr_dmi bygget  \n")
cat("Rækker:", nrow(fact_vejr_dmi), "\n\n")
glimpse(fact_vejr_dmi)
```

```
> cat("CLEAN fact_vejr_dmi bygget ✅\n")
CLEAN fact_vejr_dmi bygget ✅
> cat("Rækker:", nrow(fact_vejr_dmi), "\n\n")
Rækker: 45103

> glimpse(fact_vejr_dmi)
Rows: 45,103
Columns: 3
$ obs_dato <chr> "01-01-2000", "01-01-2000", "01-01-2000", "01-01-…
$ tid      <chr> "10:00:00", "13:00:00", "16:00:00", "19:00:00", "…
$ vejrkode <int> 60, 10, 10, 10, 11, 10, 51, 10, 10, 10, 58, 58, 0…
```

1.6.0.5 Sanity check: entydighed pr. dato og time datakonsistens

Datasættet har ikke en ensartet tidsopløsning over hele perioden: ældre observationer er registreret med ca. 3-timers interval, mens nyere data forekommer med op til 10-minutters interval. Dette håndteres ved at aggregere til én observation pr. time i CLEAN-laget.

```
# Drift-/ETL-kode (køres ikke ved render)
# ======================================================================
# 3) Sanity: Ingen dubletter pr obs_dato+tid
# ======================================================================
dup <- fact_vejr_dmi |>
  count(obs_dato, tid, sort = TRUE) |>
  filter(n > 1)

cat("Dublet-check (obs_dato + tid):", nrow(dup), "\n")
if (nrow(dup) > 0) {
  print(head(dup, 50))
  stop("STOP: Der er dubletter i CLEAN pr obs_dato+tid. Det må ikke ske.")
}

stopifnot(!any(is.na(fact_vejr_dmi$obs_dato)))
stopifnot(!any(is.na(fact_vejr_dmi$tid)))
stopifnot(!any(is.na(fact_vejr_dmi$vejrkode)))
```

```
> cat("Dublet-check (obs_dato + tid):", nrow(dup), "\n")
Dublet-check (obs_dato + tid): 0
```

1.6.0.6 Upload af CLEAN vejrdata (PBA02_Clean.fact_vejr_dmi)

Den færdige CLEAN-fact overskrives nu i PBA02_Clean med én observation pr. time. Tabellen bevarer samme struktur som før, så efterfølgende JoinReady- og analysetrin kan køre uændret.

```
# Drift-/ETL-kode (køres ikke ved render)
# ======================================================================
# 4) Upload CLEAN-fact til PBA02_Clean (samme tabelnavn som før)
# ======================================================================

DBI::dbWriteTable(
  conn      = con,
  name      = DBI::Id(schema = "PBA02_Clean", table = "fact_vejr_dmi"),
  value     = fact_vejr_dmi,
  overwrite = TRUE
)

view(fact_vejr_dmi)

cat("\n  CLEAN-fact 'PBA02_Clean.fact_vejr_dmi' er nu opdateret (1 obs pr time).\n")
cat("JoinReady kan køre uændret.\n")
```

1.7 PBA03_join_ready

I PBA03_JoinReady udføres den sidste og mest omfattende samling af data, hvor transformationer gennemføres med henblik på at gøre analysefasen så smertefri som muligt. Det er i dette lag, at dimensioner og faktatabeller sammenkobles via stabile nøgler, og datastrukturen bringes på en form, der er direkte anvendelig i modeller, analyser og visualiseringer. Dermed minimeres yderligere databehandling i analysefasen og sikrer konsistens på tværs af det videre forløb.

1.7.1 VFF billetsalg transformation + joinklar

I dette afsnit transformerer vi billetsalgsdata fra databasens PBA01_Raw-lag og klargør datasættet til videre behandling og analyse i de efterfølgende lag.

1.7.1.1 Pakkeload

Vi loader her alle de nødvendige pakker for transformationen.

```
# ------------------------------------------------------------------
# Pakker
# ------------------------------------------------------------------
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(DBI, odbc, dplyr, rstudioapi, stringr)
}); cat("Pakker er nu klar - god arbejdslyst!\n\n")
```

1.7.1.2 Safeguard – validering af miljøvariabler (.Renviron)

Dette afsnit sikrer, at alle nødvendige Azure SQL-credentials er korrekt sat i `.Renviron`. Hvis en eller flere variabler mangler, stoppes scriptet tidligt for at undgå fejlbehæftede forbindelser og utilsigtede kørsler.

```
# Drift-/ETL-kode (køres ikke ved render)

# ------------------------------------------------------------------
# Safeguard: tjek at .Renviron er sat korrekt op
# ------------------------------------------------------------------
required_vars <- c(
  "AZURE_SQL_SERVER",
  "AZURE_SQL_DB",
  "AZURE_SQL_UID",
  "AZURE_SQL_PWD"
)

values <- Sys.getenv(required_vars)
missing_vars <- required_vars[values == ""]

if (length(missing_vars) > 0) {
  cat("  Følgende miljøvariabler mangler i .Renviron:\n")
  print(missing_vars); cat("\nÅbner ~/.Renviron - udfyld værdierne og gem filen.\n")

  file.edit("~/.Renviron")
  cat("  R genstartes nu via RStudio - kør scriptet igen bagefter.\n")

  if (rstudioapi::isAvailable()) {
    rstudioapi::restartSession()
  } else {
    stop("rstudioapi er ikke tilgængelig - genstart R manuelt og kør scriptet igen.")
  }
} else {
```

```
  cat("  Alle nødvendige .Renviron-variabler er sat.\n\n")
}
```

1.7.1.3 Azure SQL-forbindelse med automatisk retry-mekanisme

Dette afsnit opretter forbindelse til Azure SQL med indbygget retry-logik og gradvist øgede timeouts. Formålet er at sikre en robust forbindelse i tilfælde af midlertidige netværks- eller belastningsproblemer i Azure.

```
# Drift-/ETL-kode (køres ikke ved render)
# ------------------------------------------------------------------------
# Azure forbindelse med automatisk retry
# ------------------------------------------------------------------------
forsøg_max <- 6
timeouts  <- c(60, 180, 200, 260, 360, 600)
delay_sec <- 10
forsøg    <- 1
con       <- NULL

while (forsøg <= forsøg_max && is.null(con)) {

  timeout_brug <- timeouts[min(forsøg, length(timeouts))]

  cat(
    "Forsøg", forsøg,
    "på at forbinde (ConnectionTimeout =", timeout_brug, "sekunder)...\n"
  )

  con_try <- try(
    DBI::dbConnect(
      odbc::odbc(),
      driver   = "ODBC Driver 18 for SQL Server",
      server   = Sys.getenv("AZURE_SQL_SERVER"),
      database = Sys.getenv("AZURE_SQL_DB"),
      uid      = Sys.getenv("AZURE_SQL_UID"),
      pwd      = Sys.getenv("AZURE_SQL_PWD"),
      port     = 1433,
      Encrypt  = "yes",
      TrustServerCertificate = "no",
      ConnectionTimeout      = timeout_brug
    ),
    silent = TRUE
```

```
  )

  if (!inherits(con_try, "try-error")) {
    con <- con_try
    cat("  Forbundet til:", Sys.getenv("AZURE_SQL_DB"), "på forsøg", forsøg, "\n\n")
  } else {
    cat("  Forbindelsen fejlede på forsøg", forsøg, "\n")

    if (forsøg == forsøg_max) {
      stop("Kunne ikke forbinde til Azure SQL efter ", forsøg_max, " forsøg.\n")
    }

    cat("Venter", delay_sec, "sekunder før næste forsøg...\n\n")
    Sys.sleep(delay_sec)
    forsøg <- forsøg + 1
  }
}
```

1.7.1.4 Indlæsning af nødvendige tabeller fra Azure SQL

I dette afsnit indlæses nødvendige dimension- og faktatabeller fra Azure SQL. Data hentes fra henholdsvis CLEAN-laget (klubdimension) og RAW-laget (billetsalg) og danner grundlag for den efterfølgende transformation, normalisering og opbygning af JoinReady-datasættet.

```
# ------------------------------------------------------------------------
# Hent tabeller
# ------------------------------------------------------------------------
dim_vff_klubber_clean <- DBI::dbReadTable(
  conn = con,
  name = DBI::Id(schema = "PBA02_Clean", table = "dim_vff_klubber_clean")
) |> as_tibble()

fact_vff_billetsalg_raw_SQLite <- DBI::dbReadTable(
  conn = con,
  name = DBI::Id(schema = "PBA01_Raw", table = "fact_vff_billetsalg_raw_SQLite")
) |> as_tibble()

cat("  Hentet dim_vff_klubber_clean:", nrow(dim_vff_klubber_clean), "rækker\n")
cat("  Hentet fact_vff_billetsalg_raw_SQLite:", nrow(fact_vff_billetsalg_raw_SQLite), "rækker

dim_vff_klubber_clean
fact_vff_billetsalg_raw_SQLite
```

```
> dim_vff_klubber_clean
# A tibble: 49 × 7
   klub_id klub_kort klub_lang     klub_navn_officiel          stadion_navn       stadion_sponsor   stadion_kapacitet
   <chr>   <chr>     <chr>         <chr>                       <chr>              <chr>                         <dbl>
 1 KLUB001 AB        AB            Akademisk Boldklub Gladsaxe Gladsaxe Stadion   Gladsaxe Stadion              13507
 2 KLUB002 ACH       AC Horsens    Alliance Club Horsens       Horsens Idrætspark) Nordstern Arena …            10495
 3 KLUB003 AGF       AGF           Aarhus Gymnastikforening    Vejlby Stadium     Vejlby Stadium                12000
 4 KLUB004 AHF       Aarhus Fremad Aarhus Fremad               Riisvangen Stadion Riisvangen Stadi…              5000
 5 KLUB005 AMA       Fremad Amager Boldklubben Fremad Amager   Sundby Idrætspark  Sundby Idrætspark              7200
 6 KLUB006 AaB       AaB           Aalborg Boldspilklub af 1885 Aalborg Stadion  Aalborg Portland…             13600
 7 KLUB007 B.93      B.93          Boldklubben af 1893         Østerbro Stadion   Østerbro Stadion              4400
 8 KLUB008 B1909     B1909         Boldklubben 1909            Gillested Park     Gillested Park                6000
 9 KLUB009 B1913     B1913         Boldklubben 1913            Campus Road        Campus Road                   2000
10 KLUB010 BIF       Brøndby IF    Brøndbyernes Idrætsforening Brøndby Stadion    Vilfort Park                 28000
# i 39 more rows
# i Use `print(n = ...)` to see more rows
```

```
# A tibble: 259 × 8
   sæson     runde tilskuere    år hold    d10_tilskuere d7_tilskuere d3_tilskuere
   <chr>     <dbl>     <dbl> <dbl> <chr>           <dbl>        <dbl>        <dbl>
 1 2000/2001     2      2089  2000 VFF- AB          1087         1282         1898
 2 2000/2001     4      3198  2000 VFF-FCK          1346         2147         2735
 3 2000/2001     5      2378  2000 VFF-SJF          1114         1607         2166
 4 2000/2001     8      3878  2000 VFF-FCM          1871         2517         3278
 5 2000/2001     9      2017  2000 VFF-SIF           949         1103         1769
 6 2000/2001    11      2308  2000 VFF-AGF          1026         1451         1994
 7 2000/2001    13      4132  2000 VFF-AaB          2181         2810         3992
 8 2000/2001    15      2308  2000 VFF-FCK          1027         1543         2049
 9 2000/2001    17      2308  2000 VFF-AGF          1278         1437         1882
10 2000/2001    19      2134  2001 VFF-HER           953         1084         1784
# i 249 more rows
# i Use `print(n = ...)` to see more rows
```

1.7.1.5 Clean og nøgleopbygning af billetsalgsdata (JoinReady)

I dette afsnit opdeles og normaliseres holdbetegnelser fra billetsalgsdata, kendte inkonsistenser korrigeres (SDR → SJF), og der foretages opslag mod klub-dimensionen for at tilknytte stabile klub-ID'er. På baggrund heraf konstrueres et entydigt kamp-ID, som danner grundlag for videre joins og analyser i JoinReady-laget.

```
# Drift-/ETL-kode (køres ikke ved render)

# ------------------------------------------------------------------------------
# Clean: split hold + SDR->SJF + join klub_id + kamp_id
# ------------------------------------------------------------------------------
dim_klub_key <- dim_vff_klubber_clean |>
  transmute(
    klub_id,
    klub_kort_norm = str_to_upper(str_squish(klub_kort))
```

```
  )

fact_VFF_Billetsalg_join_ready <- fact_vff_billetsalg_raw_SQLite |>
  mutate(
    hold_raw = hold,
    hold     = str_to_upper(str_squish(hold)),

    hjemme_kort = str_trim(str_extract(hold, "^[^-]+")),
    ude_kort    = str_trim(str_extract(hold, "[^-]+$")),

    hjemme_kort = if_else(hjemme_kort == "SDR", "SJF", hjemme_kort),
    ude_kort    = if_else(ude_kort    == "SDR", "SJF", ude_kort),

    hjemme_kort_norm = str_to_upper(str_squish(hjemme_kort)),
    ude_kort_norm    = str_to_upper(str_squish(ude_kort))
  ) |>
  left_join(dim_klub_key, by = c("hjemme_kort_norm" = "klub_kort_norm")) |>
  rename(hjemme_klubID = klub_id) |>
  left_join(dim_klub_key, by = c("ude_kort_norm" = "klub_kort_norm")) |>
  rename(ude_klubID = klub_id) |>
  select(-hjemme_kort_norm, -ude_kort_norm) |>
  mutate(
    kamp_id = paste0(sæson, "R", runde, hjemme_klubID)
  )

# ------------------------------------------------------------------------
# Tjek + view
# ------------------------------------------------------------------------
cat("Manglende hjemme_klubID:", sum(is.na(fact_VFF_Billetsalg_join_ready$hjemme_klubID)), "\n
cat("Manglende ude_klubID:   ", sum(is.na(fact_VFF_Billetsalg_join_ready$ude_klubID)), "\n\n"

if (interactive()) View(fact_VFF_Billetsalg_join_ready)

fact_VFF_Billetsalg_join_ready
```

```
> fact_VFF_Billetsalg_join_ready
# A tibble: 259 × 14
   sæson     runde tilskuere    år hold    d10_tilskuere d7_tilskuere d3_tilskuere hold_raw hjemme_kort ude_kort
   <chr>     <dbl>     <dbl> <dbl> <chr>           <dbl>        <dbl>        <dbl> <chr>    <chr>       <chr>
 1 2000/2001     2      2089  2000 VFF- AB          1087         1282         1898 VFF- AB  VFF         AB
 2 2000/2001     4      3198  2000 VFF-FCK         1346         2147         2735 VFF-FCK  VFF         FCK
 3 2000/2001     5      2378  2000 VFF-SJF         1114         1607         2166 VFF-SJF  VFF         SJF
 4 2000/2001     8      3878  2000 VFF-FCM         1871         2517         3278 VFF-FCM  VFF         FCM
 5 2000/2001     9      2017  2000 VFF-SIF          949         1103         1769 VFF-SIF  VFF         SIF
 6 2000/2001    11      2308  2000 VFF-AGF         1026         1451         1994 VFF-AGF  VFF         AGF
 7 2000/2001    13      4132  2000 VFF-AAB         2181         2810         3992 VFF-AaB  VFF         AAB
 8 2000/2001    15      2308  2000 VFF-FCK         1027         1543         2049 VFF-FCK  VFF         FCK
 9 2000/2001    17      2308  2000 VFF-AGF         1278         1437         1882 VFF-AGF  VFF         AGF
10 2000/2001    19      2134  2001 VFF-HER          953         1084         1784 VFF-HER  VFF         HER
# i 249 more rows
# i 3 more variables: hjemme_klubID <chr>, ude_klubID <chr>, kamp_id <chr>
# i Use `print(n = ...)` to see more rows
```

1.7.1.6 Upload til JoinReady – første load og inkrementel opdatering

I dette trin indlæses de klargjorte billetsalgsdata i JoinReady-laget. Tabellen oprettes ved første kørsel med et fuldt load, mens efterfølgende kørsler udelukkende indsætter nye rækker baseret på kamp_id, så dubletter undgås og tabellen holdes opdateret.

```r
# Drift-/ETL-kode (køres ikke ved render)
# ------------------------------------------------------------------
# Upload: første load / incremental (kun nye rækker)
# ------------------------------------------------------------------
target_id <- DBI::Id(schema = "PBA03_JoinReady", table = "fact_VFF_Billetsalg_join_ready")

tabel_finds <- DBI::dbExistsTable(con, target_id)

# Nøglen for incremental (din beslutning)
key_cols <- c("kamp_id")

if (!tabel_finds) {

  DBI::dbWriteTable(
    conn      = con,
    name      = target_id,
    value     = fact_VFF_Billetsalg_join_ready,
    overwrite = FALSE,
    append    = FALSE
  )

  cat("  Første load: Tabellen er oprettet og fuldt indlæst.\n")
```

```
} else {

  eksisterende_keys <- DBI::dbGetQuery(
    con,
    "SELECT DISTINCT kamp_id FROM PBA03_JoinReady.fact_VFF_Billetsalg_join_ready"
  ) |> as_tibble()

  nye_rækker <- fact_VFF_Billetsalg_join_ready |>
    anti_join(eksisterende_keys, by = key_cols)

  if (nrow(nye_rækker) == 0) {
    cat("  Ingen nye rækker - tabellen er up-to-date.\n")
  } else {

    DBI::dbWriteTable(
      conn   = con,
      name   = target_id,
      value  = nye_rækker,
      append = TRUE
    )

    cat("  Incremental update - indsatte", nrow(nye_rækker), "nye rækker.\n")
  }
}
```

1.7.1.7 Lukning af forbindelse til Azure SQL

```
# Drift-/ETL-kode (køres ikke ved render)
# ------------------------------------------------------------------------
# Luk forbindelse
# ------------------------------------------------------------------------
DBI::dbDisconnect(con)
cat("  Forbindelse lukket.\n")
```

1.7.2 Superliga program JoinReady

I dette afsnit sammenkobles det cleanede Superliga-program med klubdimensionen for at danne et join-ready faktadatasæt. Der tilføjes entydige klub-ID'er, date_key og kamp_id, hvorefter data indlæses i PBA03_JoinReady til videre analyse og modellering.

1.7.2.1 Pakker

Her henter vi og loader de nødvendige pakker

```
# ======================================================================
# Pakker
# ======================================================================
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(DBI, odbc, dplyr, tibble, stringr, lubridate)
})
```

1.7.2.2 Azure SQL-forbindelse

Her oprettes en sikker forbindelse til den fælles Azure SQL-database ved brug af credentials fra `.Renviron`, så efterfølgende data kan hentes og skrives uden hardcodede loginoplysninger.

```
# ======================================================================
# Azure forbindelse
# ======================================================================
con <- dbConnect(
  odbc::odbc(),
  driver   = "ODBC Driver 18 for SQL Server",
  server   = Sys.getenv("AZURE_SQL_SERVER"),
  database = Sys.getenv("AZURE_SQL_DB"),
  uid      = Sys.getenv("AZURE_SQL_UID"),
  pwd      = Sys.getenv("AZURE_SQL_PWD"),
  port     = 1433,
  Encrypt  = "yes",
  TrustServerCertificate = "no",
  ConnectionTimeout      = 120
)
cat("Forbundet til:", Sys.getenv("AZURE_SQL_DB"), "\n\n")
```

1.7.2.3 Indlæsning af CLEAN Superliga-program og klub-dimension

I dette trin hentes de rensede datasæt fra PBA02_Clean-laget: Superliga-kampprogrammet samt klub-dimensionen. Disse danner grundlaget for efterfølgende joins og opbygning af join-ready tabeller.

```r
# ======================================================================
# Hent CLEAN Superliga-program + CLEAN klub-dimension
# ======================================================================
fact_sl_clean <- dbReadTable(
  con,
  DBI::Id(schema = "PBA02_Clean", table = "fact_superliga_program_clean")
)

dim_klub <- dbReadTable(
  con,
  DBI::Id(schema = "PBA02_Clean", table = "dim_vff_klubber_clean")
)

cat("Rækker hentet fra Superliga CLEAN:", nrow(fact_sl_clean), "\n")
cat("Rækker hentet fra Klub-dimension:", nrow(dim_klub), "\n\n")
tibble(fact_superliga_program_clean)
view(dim_klub)
```

```
> tibble(fact_superliga_program_clean)
# A tibble: 5,248 × 15
   sæson    runde ugedag dato       date_key tid   hjemmehold udehold mål_hjemme mål_ude vinder vinder_type tilskuertal
   <chr>    <int> <chr>  <date>     <chr>    <chr> <chr>      <chr>        <int>   <int> <chr>  <chr>        <chr>
 1 2000/20…     1 Lørdag 2000-07-22 22072000 15:3… BIF        AGF              1       2 AGF    Udehold      7806
 2 2000/20…     1 Søndag 2000-07-23 23072000 15:0… AB         SIF              1       1 Uafgj… Uafgjort     2411
 3 2000/20…     1 Søndag 2000-07-23 23072000 15:0… FCM        LBK              4       0 FCM    Hjemmehold   3706
 4 2000/20…     1 Søndag 2000-07-23 23072000 17:0… FCK        SJF              5       1 FCK    Hjemmehold  11977
 5 2000/20…     1 Mandag 2000-07-24 24072000 19:0… OB         VFF              0       1 VFF    Udehold      4788
 6 2000/20…     1 Onsdag 2000-07-26 26072000 19:0… HER        AaB              2       3 AaB    Udehold      1953
 7 2000/20…     2 Lørdag 2000-07-29 29072000 15:3… BIF        OB               0       0 Uafgj… Uafgjort     6388
 8 2000/20…     2 Søndag 2000-07-30 30072000 15:0… LBK        HER              3       1 LBK    Hjemmehold   2503
 9 2000/20…     2 Søndag 2000-07-30 30072000 15:0… VFF        AB               1       1 Uafgj… Uafgjort     2089
10 2000/20…     2 Søndag 2000-07-30 30072000 15:0… AGF        FCK              1       0 AGF    Hjemmehold   7966
# ℹ 5,238 more rows
# ℹ 2 more variables: dommer <chr>, tv_kanal <chr>
# ℹ Use `print(n = ...)` to see more rows
>
```

dim_klub

Filter

	klub_id	klub_kort	klub_lang	klub_navn_officiel	stadion_navn	stadion_sponsor	stadion_kapacitet
1	KLUB001	AB	AB	Akademisk Boldklub Gladsaxe	Gladsaxe Stadion	Gladsaxe Stadion	13507
2	KLUB002	ACH	AB AC Horsens	Alliance Club Horsens	Horsens Idrætspark)	Nordstern Arena Horsens	10495
3	KLUB003	AGF	AGF	Aarhus Gymnastikforening	Vejlby Stadium	Vejlby Stadium	12000
4	KLUB004	AHF	Aarhus Fremad	Aarhus Fremad	Riisvangen Stadion	Riisvangen Stadion	5000
5	KLUB005	AMA	Fremad Amager	Boldklubben Fremad Amager	Sundby Idrætspark	Sundby Idrætspark	7200
6	KLUB006	AaB	AaB	Aalborg Boldspilklub af 1885	Aalborg Stadion	Aalborg Portland Park	13600
7	KLUB007	B.93	B.93	Boldklubben af 1893	Østerbro Stadion	Østerbro Stadion	4400
8	KLUB008	B1909	B1909	Boldklubben 1909	Gillested Park	Gillested Park	6000
9	KLUB009	B1913	B1913	Boldklubben 1913	Campus Road	Campus Road	2000
10	KLUB010	BIF	Brøndby IF	Brøndbyernes Idrætsforening	Brøndby Stadion	Vilfort Park	28000
11	KLUB011	BKF	BK Frem	Boldklubben Frem af 1886	Valby Idrætspark	Valby Idrætspark	12000
12	KLUB012	BKS	BK Skjold	Boldklubben Skjold	Østerbro Stadion	Østerbro Stadion	4400
13	KLUB013	BRA	Brabrand IF	Brabrand Idrætsforening	Brabrand Stadion	Brabrand Stadion	*NA*
14	KLUB014	BRØ	Brønshøj BK	Brønshøj Boldklub	Tingbjerg Idrætspark	Tingbjerg Idrætspark	3000
15	KLUB015	DAL	Dalum IF	Dalum Idrætsforening	Dalum Stadion	Rema 1000 Park	4000

1.7.2.4 Join af klub-ID på hjemme- og udehold

Her kobles klub-ID'er fra klub-dimensionen på både hjemme- og udehold i Superliga-programmet. Dermed sikres entydige og stabile nøgler, som kan anvendes konsekvent i de efterfølgende join-ready og analyse-trin.

```
# ======================================================================
# Join klub-ID på hjemmehold og udehold
# ======================================================================
fact_joined <- fact_sl_clean %>%
  left_join(
    dim_klub %>% dplyr::select(klub_kort, klub_id),
    by = c("hjemmehold" = "klub_kort")
  ) %>%
  rename(klub_id_hjemme = klub_id) %>%
  left_join(
    dim_klub %>% dplyr::select(klub_kort, klub_id),
    by = c("udehold" = "klub_kort")
  ) %>%
  rename(klub_id_ude = klub_id) %>%
  mutate(
    hjemme_klubID = if_else(is.na(klub_id_hjemme), hjemmehold, klub_id_hjemme),
    ude_klubID    = if_else(is.na(klub_id_ude),    udehold,    klub_id_ude)
  ) %>%
  dplyr::select(-klub_id_hjemme, -klub_id_ude)
```

```
> tibble(fact_joined)
# A tibble: 5,248 × 18
   sæson runde ugedag dato       date_key tid   hjemmehold udehold mål_hjemme mål_ude
   <chr> <int> <chr>  <date>     <chr>    <chr> <chr>      <chr>         <int>   <int>
 1 2000…     1 Lørdag 2000-07-22 22072000 15:3… BIF        AGF               1       2
 2 2000…     1 Søndag 2000-07-23 23072000 15:0… AB         SIF               1       1
 3 2000…     1 Søndag 2000-07-23 23072000 15:0… FCM        LBK               4       0
 4 2000…     1 Søndag 2000-07-23 23072000 17:0… FCK        SJF               5       1
 5 2000…     1 Mandag 2000-07-24 24072000 19:0… OB         VFF               0       1
 6 2000…     1 Onsdag 2000-07-26 26072000 19:0… HER        AaB               2       3
 7 2000…     2 Lørdag 2000-07-29 29072000 15:3… BIF        OB                0       0
 8 2000…     2 Søndag 2000-07-30 30072000 15:0… LBK        HER               3       1
 9 2000…     2 Søndag 2000-07-30 30072000 15:0… VFF        AB                1       1
10 2000…     2 Søndag 2000-07-30 30072000 15:0… AGF        FCK               1       0
# i 5,238 more rows
# i 8 more variables: vinder <chr>, vinder_type <chr>, tilskuertal <chr>,
#   dommer <chr>, tv_kanal <chr>, hjemme_klubID <chr>, ude_klubID <chr>,
#   kamp_id <chr>
# i Use `print(n = ...)` to see more rows
```

1.7.2.5 Oprettelse af date_key

Her oprettes en standardiseret date_key i formatet DDMMÅÅÅÅ baseret på kampdatoen. Denne nøgle anvendes senere til joins mod datodimensioner og sikrer konsistent datohåndtering på tværs af datasæt.

```
# ======================================================================
# 3) Opret date_key (ddmmyyyy som tekst)
# ======================================================================
fact_joined <- fact_joined %>%
  mutate(
    date_key = format(as.Date(dato), "%d%m%Y")
  )
```

1.7.2.6 Oprettelse af kamp_id

Her konstrueres et entydigt kamp_id baseret på sæson, runde og klubidentifikatorer. Nøglen bruges som primær reference i JoinReady-laget og sikrer stabile og entydige joins på tværs af fakta- og dimensionsdata.

```
# ======================================================================
# 4) Tilføj kamp_id (sæson + "R" + runde + "KLUB046")
# ======================================================================
VFF_ID <- "KLUB046"
```

```
fact_joined <- fact_joined %>%
  mutate(
    kamp_id = paste0(sæson, "R", runde, VFF_ID, ude_klubID)
    )
view(fact_joined)
```

sæson	runde	ugedag	dato	date_key	tid	hjemmehold	udehold	mål_hjemme	mål_ude	vinder	vinder_type	tilskuertal	dommer	tv_kanal	hjemme_klubID	ude_klubID	kamp_id
2000/2001	1	Lørdag	2000-07-22	22072000	15:30:00	BIF	AGF	1	2	AGF	Udehold	7806	Johnny RÃ¸n	NA	KLUB010	KLUB003	2000/2001R1KLUB046KLUB003
2000/2001	1	Søndag	2000-07-23	23072000	15:00:00	AB	SIF	1	1	Uafgjort	Uafgjort	2411	Nicolai Vollquartz	NA	KLUB001	KLUB041	2000/2001R1KLUB046KLUB041
2000/2001	1	Søndag	2000-07-23	23072000	15:00:00	FCM	LBK	4	0	FCM	Hjemmehold	3706	Knud Erik Fisker	NA	KLUB020	KLUB033	2000/2001R1KLUB046KLUB033
2000/2001	1	Søndag	2000-07-23	23072000	17:05:00	FCK	SJF	5	1	FCK	Hjemmehold	11977	Claus Bo Larsen	TV3+	KLUB019	KLUB049	2000/2001R1KLUB046KLUB049
2000/2001	1	Mandag	2000-07-24	24072000	19:00:00	OB	VFF	0	1	VFF	Udehold	4788	Johnny RÃ¸n	TV3+	KLUB037	KLUB046	2000/2001R1KLUB046KLUB046
2000/2001	1	Onsdag	2000-07-26	26072000	19:00:00	HER	AaB	2	3	AaB	Udehold	1953	Tonny K. Poulsen	NA	KLUB025	KLUB006	2000/2001R1KLUB046KLUB006

1.7.2.7 Klargøring af join-ready tabel

Datasættet struktureres endeligt ved at placere centrale nøgler (kamp_id, date_key og klub-ID'er) konsekvent. Resultatet er en join-ready tabel, klar til brug i videre analyse og modellering uden yderligere transformationer.

```
# ======================================================================
# 5) Endelig join-ready tabel (med kamp_id)
# ======================================================================
fact_superliga_program_join_ready <- fact_joined %>%
  relocate(date_key, .after = dato) %>%
  relocate(hjemme_klubID, ude_klubID, .after = udehold) %>%
  relocate(kamp_id, .before = sæson)

if (interactive()) View(fact_superliga_program_join_ready)
```

1.7.2.8 Load til PBA03_JoinReady (første load og inkrementel opdatering)

I dette trin indlæses den join-ready tabel i PBA03_JoinReady. Tabellen oprettes ved første kørsel og opdateres herefter inkrementelt, så kun nye kampe (baseret på kamp_id) indsættes, hvilket sikrer stabil drift uden dubletter.

```
# ======================================================================
# 6) Første load + incremental update (PBA03_JoinReady)
# ======================================================================
target_schema <- "PBA03_JoinReady"
target_table  <- "fact_superliga_program_join_ready"

table_id <- DBI::Id(
```

```
  schema = target_schema,
  table  = target_table
)

tabel_findes <- DBI::dbExistsTable(con, table_id)

if (!tabel_findes) {

  cat("  Første load - opretter tabel...\n")

  DBI::dbWriteTable(
    conn      = con,
    name      = table_id,
    value     = fact_superliga_program_join_ready,
    overwrite = TRUE
  )

  cat("  Første load gennemført. Rækker:", nrow(fact_superliga_program_join_ready), "\n\n")

} else {

  cat("  Incremental update (kun nye kamp_id)...\n")

  eksisterende_ids <- DBI::dbReadTable(con, table_id) %>%
    dplyr::select(kamp_id) %>%
    mutate(kamp_id = as.character(kamp_id))

  nye_rækker <- fact_superliga_program_join_ready %>%
    mutate(kamp_id = as.character(kamp_id)) %>%
    dplyr::anti_join(eksisterende_ids, by = "kamp_id")

  if (nrow(nye_rækker) > 0) {

    DBI::dbWriteTable(
      conn      = con,
      name      = table_id,
      value     = nye_rækker,
      append    = TRUE
    )

    cat("  Indsat nye rækker:", nrow(nye_rækker), "\n\n")
```

```
  } else {
    cat("  Ingen nye rækker at indsætte.\n\n")
  }
}

# ======================================================================
# Luk forbindelse
# ======================================================================
DBI::dbDisconnect(con)
cat("  Forbindelse lukket.\n")
```

1.7.3 Vejrdata - PBA03_JoinReady (join-ready og standardiseret)

I dette afsnit samles rensede vejrdata med tilhørende vejrkode-dimension og standardiseres til ét konsistent observationsniveau. Datasættet reduceres til én repræsentativ række pr. dato via et fast tids-grid med fallback-logik og klargøres herefter til direkte brug i analyse og modellering i PBA03_JoinReady.

1.7.3.1 Pakker

Her indlæses de nødvendige R-pakker, som bruges til databaseforbindelse, datahåndtering og tekstbehandling i det efterfølgende join-ready-workflow.

```
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(DBI, odbc, dplyr, tibble, stringr)
})
```

1.7.3.2 Forbindelse til Azure SQL

I dette trin oprettes en sikker ODBC-forbindelse til det adgangsbegrænsede Azure SQL-datalager, som anvendes til at hente og skrive data i JoinReady-laget.

```
con <- dbConnect(
  odbc::odbc(),
  driver   = "ODBC Driver 18 for SQL Server",
  server   = Sys.getenv("AZURE_SQL_SERVER"),
  database = Sys.getenv("AZURE_SQL_DB"),
  uid      = Sys.getenv("AZURE_SQL_UID"),
  pwd      = Sys.getenv("AZURE_SQL_PWD"),
  port     = 1433,
```

```
  Encrypt  = "yes",
  TrustServerCertificate = "no",
  ConnectionTimeout      = 30
)
cat("Forbundet til:", Sys.getenv("AZURE_SQL_DB"), "\n\n")
```

1.7.3.3 Indlæsning af CLEAN vejrdata og kode-dimension

Her indlæses de rensede vejr-observationer samt tilhørende vejrkode-dimension fra PBA02_Clean. Disse datasæt danner grundlaget for efterfølgende kvalitetstjek, joins og standardisering i JoinReady-laget.

```
# ======================================================================
# Hent CLEAN FACT + DIM
# ======================================================================
fact_vejr_dmi <- dbReadTable(con, DBI::Id(schema = "PBA02_Clean", table = "fact_vejr_dmi"))
dim_vejrkode  <- dbReadTable(con, DBI::Id(schema = "PBA02_Clean", table = "dim_vejrkode"))

cat("Rækker i FACT (PBA02_Clean.fact_vejr_dmi):", nrow(fact_vejr_dmi), "\n")
cat("Rækker i DIM  (PBA02_Clean.dim_vejrkode):", nrow(dim_vejrkode), "\n\n")
str(fact_vejr_dmi); str(dim_vejrkode)
```

```
> str(fact_vejr_dmi); str(dim_vejrkode)
'data.frame':   45072 obs. of  3 variables:
 $ obs_dato: chr  "01-01-2000" "01-01-2000" "01-01-2000" "01-01-2000" ...
 $ tid     : chr  "10:00:00" "13:00:00" "16:00:00" "19:00:00" ...
 $ vejrkode: int  60 10 10 10 11 10 51 10 10 10 ...
'data.frame':   200 obs. of  2 variables:
 $ vejrkode       : num  0 1 2 3 4 5 6 7 8 9 ...
 $ vejrbeskrivelse: chr  "Skyudvikling ikke observeret eller kan ikke observeres - karakteristisk ændring i himlens tilstand
i den seneste time" "Skyer er generelt ved at opløses eller blive mindre udviklede - karakteristisk ændring i himlens tilsta
nd i den seneste time" "Himlens tilstand stort set uændret i den seneste time" "Skyer er generelt ved at dannes eller udvikl
es - karakteristisk ændring i himlens tilstand i den seneste time" ...
```

1.7.3.4 Kvalitetstjek af vejrkoder før join

I dette trin kontrolleres konsistensen mellem FACT- og DIM-laget ved at identificere vejrkoder, som forekommer i observationerne, men ikke har en tilsvarende beskrivelse i kode-dimensionen. Formålet er at sikre et fuldt dækkende og korrekt join uden tab af information.

```
# ==============================================================================
# Kvalitetstjek før join - manglende matches (vejrkode)
# ==============================================================================
distinct_fact_koder <- fact_vejr_dmi |>
  distinct(vejrkode) |>
```

```
  arrange(vejrkode)

distinct_dim_koder <- dim_vejrkode |>
  distinct(vejrkode) |>
  arrange(vejrkode)

cat("Antal unikke vejrkoder i FACT:", nrow(distinct_fact_koder), "\n")
cat("Antal unikke vejrkoder i DIM :", nrow(distinct_dim_koder), "\n")

faktiske_uden_dim <- distinct_fact_koder |>
  anti_join(distinct_dim_koder, by = "vejrkode")

cat("Antal vejrkoder i FACT uden match i DIM:", nrow(faktiske_uden_dim), "\n")
if (nrow(faktiske_uden_dim) > 0) {
  cat("Koder uden match i DIM:\n")
  print(faktiske_uden_dim)
}
cat("\n")
```

```
Antal unikke vejrkoder i FACT: 119
Antal unikke vejrkoder i DIM : 200
Antal vejrkoder i FACT uden match i DIM: 0
```

1.7.3.5 Join af vejrdata og kode-dimension (rå JoinReady)

Her sammenkædes vejr-observationerne fra FACT-laget med vejrkode-dimensionen for at berige data med beskrivende metadata. Samtidig oprettes tekniske nøgler (date_key og Date_time_key), så datasættet er klar til videre standardisering og brug i JoinReady-laget.

```
# ======================================================================
# Join FACT + DIM → (raw) fact_vejr_join_ready
# ======================================================================
fact_vejr_join_ready_raw <- fact_vejr_dmi |>
  left_join(dim_vejrkode, by = "vejrkode") |>
  mutate(
    obs_dato_date = as.Date(obs_dato, format = "%d-%m-%Y"),
    date_key      = format(obs_dato_date, "%d%m%Y"),
    Date_time_key = paste0(date_key, str_replace_all(tid, ":", ""))
  ) |>
```

```
  select(-obs_dato_date)

cat("Rækker i fact_vejr_join_ready_raw:", nrow(fact_vejr_join_ready_raw), "\n\n")

# Valgfri: kig på rå join
View(fact_vejr_join_ready_raw)
```

	obs_dato	tid	vejrkode	vejrbeskrivelse	date_key	Date_time_key
1	01-01-2000	10:00:00	60	Regn, ikke underafkølet, periodisk – svag på observationstid...	01012000	01012000100000
2	01-01-2000	13:00:00	10	Tåge	01012000	01012000130000
3	01-01-2000	16:00:00	10	Tåge	01012000	01012000160000
4	01-01-2000	19:00:00	10	Tåge	01012000	01012000190000
5	01-01-2000	22:00:00	11	Plettet lav tåge eller is-tåge ved stationen (på land = ca. 2 m...	01012000	01012000220000
6	02-01-2000	10:00:00	10	Tåge	02012000	02012000100000
7	02-01-2000	13:00:00	51	Støvregn, ikke underafkølet, vedvarende – svag på observati...	02012000	02012000130000
8	02-01-2000	16:00:00	10	Tåge	02012000	02012000160000
9	02-01-2000	19:00:00	10	Tåge	02012000	02012000190000
10	02-01-2000	22:00:00	10	Tåge	02012000	02012000220000
11	03-01-2000	10:00:00	58	Støvregn og regn, svag	03012000	03012000100000
12	03-01-2000	13:00:00	58	Støvregn og regn, svag	03012000	03012000130000

1.7.3.6 Kvalitetstjek af join – manglende vejrbeskrivelser

I dette trin kontrolleres resultatet af joinet for manglende værdier i *vejrbeskrivelse*. Formålet er at identificere eventuelle vejrkoder, som findes i FACT-tabellen, men ikke har en tilsvarende beskrivelse i DIM-tabellen, så datakvaliteten kan sikres før videre behandling.

```
# ======================================================================
# 5) Kvalitetstjek efter join - NA i vejrbeskrivelse
# ======================================================================
mangler_beskrivelse <- fact_vejr_join_ready_raw |>
  filter(is.na(vejrbeskrivelse)) |>
  distinct(vejrkode) |>
  arrange(vejrkode)

cat("Antal forskellige koder uden vejrbeskrivelse efter join:", nrow(mangler_beskrivelse), "
if (nrow(mangler_beskrivelse) > 0) {
  cat("Disse koder mangler tekst i DIM:\n")
  print(mangler_beskrivelse)
}
cat("\n")
```

1.7.3.7 Standardisering af vejrdata til én repræsentativ observation pr. dato

I dette trin aggregeres vejrdata, så der kun forekommer én observation pr. dato. Observationen vælges som den måling, der ligger tættest på kl. 14:00, som anvendes som et fast og sammenligneligt referencepunkt på dagen. Der prioriteres først faste observationstidspunkter kl. 08:00, 11:00, 14:00 og 17:00. For datoer, hvor disse observationer mangler, anvendes en kontrolleret fallback-strategi, hvor den nærmeste tilgængelige observation vælges. Processen sikrer et konsistent, fuldstændigt og metodisk gennemsigtigt datasæt med præcis én række pr. dato.

```
# Formål:
#   At sikre præcis én vejr-observation pr. dato ved at:
#   1) Prioritere faste observationstidspunkter (grid)
#   2) Vælge den observation, der ligger tættest på kl. 14:00
#   3) Anvende fallback-logik på dage uden grid-observationer
#
# Grid-tider: 08:00, 11:00, 14:00, 17:00
# Måltidspunkt (anker): 14:00
# ======================================================================

# Definerer de faste tidspunkter, som betragtes som strukturerede observationer
faste_tider <- c("08:00:00", "11:00:00", "14:00:00", "17:00:00")

# Definerer det tidspunkt på dagen, som alle observationer måles op imod
mål_tid     <- "14:00:00"

# Hjælpefunktion:
# Konverterer et klokkeslæt (HH:MM:SS) til sekunder siden midnat
# Dette gør det muligt at beregne numeriske tidsafstande
tid_til_sek <- function(x) {
  as.integer(substr(x, 1, 2)) * 3600 +
    as.integer(substr(x, 4, 5)) * 60 +
    as.integer(substr(x, 7, 8))
}

# Konverterer måltidspunktet (14:00) til sekunder
mål_sec <- tid_til_sek(mål_tid)

# Udtrækker alle unikke datoer i datasættet
# Denne liste fungerer som facit for, hvor mange rækker slutdatasættet skal have
dato_liste <- fact_vejr_join_ready_raw |>
  distinct(obs_dato) |>
  arrange(obs_dato)
```

```r
# Antal unikke datoer (bruges senere til konsistenskontrol)
n_dage <- nrow(dato_liste)

# Filtrerer datasættet ned til kun observationer på grid-tidspunkter
# Disse observationer prioriteres i første omgang
vejr_grid <- fact_vejr_join_ready_raw |>
  filter(tid %in% faste_tider) |>
  arrange(obs_dato, tid)

# Sikkerhedstjek:
# Der må højst være én observation pr. dato + tid i grid
# Hvis ikke, stoppes scriptet for at undgå tvetydighed
dup_grid <- vejr_grid |>
  count(obs_dato, tid) |>
  filter(n != 1)
stopifnot(nrow(dup_grid) == 0)

# For hver dato i grid:
# 1) Beregnes afstanden til kl. 14:00
# 2) Observationen tættest på 14:00 vælges
# 3) Rækken mærkes som "grid" for fuld sporbarhed
vejr_1_pr_dato_fra_grid <- vejr_grid |>
  mutate(
    tid_sec = tid_til_sek(tid),
    dist_sec = abs(tid_sec - mål_sec),
    dist_minutes_to_14 = dist_sec / 60,
    source = "grid"
  ) |>
  group_by(obs_dato) |>
  arrange(dist_sec, .by_group = TRUE) |>
  slice(1) |>
  ungroup() |>
  select(-tid_sec, -dist_sec)

# Identificerer de datoer, som ikke har nogen grid-observation
# Disse datoer kræver fallback-logik
mangler_datoer <- dato_liste |>
  anti_join(vejr_1_pr_dato_fra_grid |> distinct(obs_dato), by = "obs_dato")

# Fallback:
# For dage uden grid-observation vælges den observation,
# der (uanset tidspunkt) ligger tættest på kl. 14:00
```

```r
# Disse rækker mærkes eksplicit som "fallback"
vejr_fallback <- fact_vejr_join_ready_raw |>
  semi_join(mangler_datoer, by = "obs_dato") |>
  mutate(
    tid_sec = tid_til_sek(tid),
    dist_sec = abs(tid_sec - mål_sec),
    dist_minutes_to_14 = dist_sec / 60,
    source = "fallback"
  ) |>
  group_by(obs_dato) |>
  arrange(dist_sec, .by_group = TRUE) |>
  slice(1) |>
  ungroup() |>
  select(-tid_sec, -dist_sec)

# Samler grid-udvalg og fallback-udvalg til ét endeligt datasæt
# Sorteres kronologisk
fact_vejr_join_ready <- bind_rows(vejr_1_pr_dato_fra_grid, vejr_fallback) |>
  arrange(obs_dato)

# Endelige sikkerhedstjek:
# 1) Antal rækker skal svare til antal unikke datoer
# 2) Ingen dato må forekomme mere end én gang
stopifnot(nrow(fact_vejr_join_ready) == n_dage)
stopifnot(sum(duplicated(fact_vejr_join_ready$obs_dato)) == 0)

# Konsol-output som dokumenterer resultatet og datakvaliteten
cat("  Endeligt datasæt (1 pr dato) klar.\n")
cat("Rækker:", nrow(fact_vejr_join_ready), " | Distinct datoer:", n_dage, "\n")
cat("Fordeling source:\n")
print(dplyr::count(fact_vejr_join_ready, source))
cat("\n")

# Mulighed for manuelt eftersyn før upload / videre brug
View(fact_vejr_join_ready)
```

obs_dato	tid	vejrkode	vejrbeskrivelse	date_key	Date_time_key	dist_minutes_to_14	source
01-01-2000	13:00:00	10	Tåge	01012000	01012000130000	60	fallback
01-01-2001	13:00:00	70	Periodisk snefald – svagt på observationstidspunktet	01012001	01012001130000	60	fallback
01-01-2002	13:00:00	21	Regn (ikke underafkølet), ikke som byger	01012002	01012002130000	60	fallback
01-01-2003	13:00:00	0	Skyudvikling ikke observeret eller kan ikke observeres – kara...	01012003	01012003130000	60	fallback
01-01-2004	13:00:00	10	Tåge	01012004	01012004130000	60	fallback
01-01-2005	13:00:00	2	Himlens tilstand stort set uændret i den seneste time	01012005	01012005130000	60	fallback
01-01-2006	13:00:00	2	Himlens tilstand stort set uændret i den seneste time	01012006	01012006130000	60	fallback
01-01-2007	13:00:00	80	Regnbyge(r), svag	01012007	01012007130000	60	fallback
01-01-2008	13:00:00	2	Himlens tilstand stort set uændret i den seneste time	01012008	01012008130000	60	fallback
01-01-2009	10:00:00	11	Plettet lav tåge eller is-tåge ved stationen (på land = ca. 2 m...	01012009	01012009100000	240	fallback
01-01-2010	16:00:00	70	Periodisk snefald – svagt på observationstidspunktet	01012010	01012010160000	120	fallback
01-01-2011	16:00:00	10	Tåge	01012011	01012011160000	120	fallback
01-01-2012	13:00:00	2	Himlens tilstand stort set uændret i den seneste time	01012012	01012012130000	60	fallback
01-01-2013	13:00:00	25	Regnbyge(r)	01012013	01012013130000	60	fallback
01-01-2014	14:00:00	110	Tåge	01012014	01012014140000	0	grid
01-01-2015	14:00:00	110	Tåge	01012015	01012015140000	0	grid
01-01-2016	14:00:00	110	Tåge	01012016	01012016140000	0	grid
01-01-2018	14:00:00	161	Regn, ikke underafkølet, svag	01012018	01012018140000	0	grid
01-01-2019	14:00:00	100	Ingen væsentligt vejr observeret	01012019	01012019140000	0	grid
01-01-2020	21:00:00	110	Tåge	01012020	01012020210000	420	fallback

1.7.3.8 Upload af færdigbehandlet vejrdata og lukning af databaseforbindelse

I dette trin uploades det endelige og validerede vejr-datasæt til databasen i laget PBA03_Join-Ready. Tabellen overskrives bevidst for at sikre, at databasen altid indeholder den seneste og konsistente version af data med præcis én observation pr. dato. Efter uploaden lukkes databaseforbindelsen kontrolleret for at frigive ressourcer og afslutte ETL-processen korrekt.

```
# ======================================================================
# Upload (overwrite) til PBA03_JoinReady.fact_vejr_join_ready
# ======================================================================
DBI::dbWriteTable(
  conn      = con,
  name      = DBI::Id(schema = "PBA03_JoinReady", table = "fact_vejr_join_ready"),
  value     = fact_vejr_join_ready,
  overwrite = TRUE
)

cat("  Tabel 'PBA03_JoinReady.fact_vejr_join_ready' er nu skrevet til databasen (overwrite).\

# ======================================================================
# Luk forbindelse
# ======================================================================
```

```
dbDisconnect(con)
cat("  Forbindelse lukket.\n")
```

1.7.4 Temperaturdata: klargøring fra RAW til join-ready

Dette afsnit beskriver behandlingen af rå temperaturdata fra PBA01_Raw til en reduceret og analytisk anvendelig FACT-tabel i PBA03_JoinReady. Data opdeles i dato og tid, filtreres til et kamprelevant tidsvindue og reduceres fra høj frekvens til udvalgte heltimer, som vurderes mest repræsentative for kampafvikling. Processen er bevidst designet til at balancere datakvalitet, relevans og stabilitet i efterfølgende joins og modeller, og afsluttes med upload af en konsistent join-ready tabel, der kan anvendes direkte i analyse- og modelarbejde.

1.7.4.1 Pakker

Nødvendige pakker loades og gøres klar til anvendelse.

```
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(DBI, odbc, dplyr, tibble, tidyverse, stringr)
}); cat("Pakker er klar  \n\n")
```

1.7.4.2 Sikkerhedstjek af databasekonfiguration via .Renviron

Dette trin fungerer som et tidligt safeguard, der sikrer, at alle nødvendige miljøvariabler til Azure SQL-forbindelsen er korrekt defineret i .Renviron-filen. Inden der oprettes forbindelse til databasen, kontrolleres det eksplicit, at servernavn, databasenavn, brugernavn og adgangskode er tilgængelige. Mangler én eller flere variabler, afbrydes processen bevidst for at undgå skjulte forbindelsesfejl og uklare runtime-problemer senere i ETL-flowet.

```
# ======================================================================
# Safeguard: .Renviron
# ======================================================================
ensure_renviron <- function() {
  required_vars <- c("AZURE_SQL_SERVER","AZURE_SQL_DB","AZURE_SQL_UID","AZURE_SQL_PWD")
  values <- Sys.getenv(required_vars)
  missing_vars <- required_vars[values == ""]
  if (length(missing_vars) > 0) {
    stop("  Manglende miljøvariabler i .Renviron: ", paste(missing_vars, collapse = ", "))
  }
  cat("  .Renviron OK\n\n")
```

```
}
ensure_renviron()
```

1.7.4.3 Stabil oprettelse af forbindelse til Azure SQL med retry-logik

Dette trin etablerer en robust databaseforbindelse til Azure SQL ved hjælp af en eksplicit retry-mekanisme. Forbindelsen forsøges oprettet flere gange med gradvist længere timeouts for at håndtere midlertidige netværksproblemer og kendte Azure-fejl som 08S01 communication link failure eller timeout. Processen sikrer, at ETL-flowet enten opnår en stabil forbindelse eller stopper kontrolleret med en klar fejlmeddelelse, frem for at fejle uforudsigeligt midt i kørslen.

```
# ======================================================================# Retry-funktion: Azure
# ======================================================================
connect_azure_retry <- function(
    forsøg_max = 6,
    timeouts  = c(60, 180, 200, 260, 360, 600),
    delay_sec = 10
) {
  forsøg <- 1
  con <- NULL

  while (forsøg <= forsøg_max && is.null(con)) {
    timeout_brug <- timeouts[min(forsøg, length(timeouts))]
    cat("Forsøg", forsøg, "- ConnectionTimeout =", timeout_brug, "sekunder\n")

    con_try <- try(
      DBI::dbConnect(
        odbc::odbc(),
        driver   = "ODBC Driver 18 for SQL Server",
        server   = Sys.getenv("AZURE_SQL_SERVER"),
        database = Sys.getenv("AZURE_SQL_DB"),
        uid      = Sys.getenv("AZURE_SQL_UID"),
        pwd      = Sys.getenv("AZURE_SQL_PWD"),
        port     = 1433,
        Encrypt  = "yes",
        TrustServerCertificate = "no",
        ConnectionTimeout = timeout_brug
      ),
      silent = TRUE
    )

    if (!inherits(con_try, "try-error")) {
```

```
      cat("  Forbundet til Azure SQL på forsøg", forsøg, "\n\n")
      return(con_try)
    }

    if (forsøg == forsøg_max) stop("  Kunne ikke forbinde til Azure SQL efter ", forsøg_max,
    Sys.sleep(delay_sec)
    forsøg <- forsøg + 1
  }
}

con <- connect_azure_retry()
on.exit(try(DBI::dbDisconnect(con), silent = TRUE), add = TRUE)
```

1.7.4.4 Indlæsning af rå temperaturdata fra databasens RAW-lag

I dette trin indlæses de rå temperaturdata fra tabellen PBA01_Raw.fact_temp_dry_raw i databasen. Kun de nødvendige kolonner, observationstidspunkt og temperaturværdi, udvælges for at holde datasættet fokuseret og let at arbejde videre med. Antallet af indlæste rækker udskrives efterfølgende som et simpelt kontrolpunkt, der bekræfter, at data er hentet korrekt fra RAW-laget og står klar til videre transformation.

```
# ======================================================================
# Hent RAW temperaturdata fra PBA01_Raw.fact_temp_dry_raw
# ======================================================================
fact_temp_dry_raw1 <- DBI::dbReadTable(
  con,
  DBI::Id(schema = "PBA01_Raw", table = "fact_temp_dry_raw")
) |>
  dplyr::select(observed, value)

cat("Antal rækker i PBA01_Raw.fact_temp_dry_raw:", nrow(fact_temp_dry_raw1), "\n\n"); str(fa
```

```
> cat("Antal rækker i PBA01_Raw.fact_temp_dry_raw:", nrow(fact_temp_dry_r
aw1), "\n\n"); str(fact_temp_dry_raw1)
Antal rækker i PBA01_Raw.fact_temp_dry_raw: 1207946

'data.frame':   1207946 obs. of  2 variables:
 $ observed: POSIXct, format: "2000-01-01 00:00:00" "2000-01-01 03:00:00"
"2000-01-01 06:00:00" "2000-01-01 09:00:00" ...
 $ value   : num  1.9 2.8 3.3 4 6.2 5.5 5 3.1 3.5 4.6 ...
```

1.7.4.5 Opdeling i dato og tid samt filtrering til kamprelevant tidsvindue

I dette trin opdeles observationstidspunktet i en separat dato- og tidskomponent, hvorefter data filtreres til tidsrummet kl. 08:00–22:00. Dette interval afspejler et bevidst forretningsvalg, hvor temperaturer uden for typiske publikums- og kamprelevante tidspunkter udelades. Samtidig omdøbes og struktureres kolonnerne til et mere forretningsnært format, som er direkte egnet til videre analyse og joins.

```
# ======================================================================
# Dato, tid og filter 08-22 (forretningslogik)
# ======================================================================
fact_temperatur_dmi <- fact_temp_dry_raw1 |>
  dplyr::mutate(
    obs_dato_raw = as.Date(observed),
    tid          = format(observed, "%H:%M:%S")
  ) |>
  dplyr::filter(
    tid >= "08:00:00",
    tid <= "22:00:00"
  ) |>
  dplyr::transmute(
    obs_dato   = format(obs_dato_raw, "%d-%m-%Y"),
    tid        = tid,
    temperatur = value
  )

cat("Antal rækker efter filter 08-22:", nrow(fact_temperatur_dmi), "\n\n"); view(fact_tempera
```

obs_dato	tid	temperatur
01-01-2000	09:00:00	4.0
01-01-2000	12:00:00	6.2
01-01-2000	15:00:00	5.5
01-01-2000	18:00:00	5.0
01-01-2000	21:00:00	3.1
02-01-2000	09:00:00	6.1
02-01-2000	12:00:00	6.5
02-01-2000	15:00:00	7.2
02-01-2000	18:00:00	6.1
02-01-2000	21:00:00	3.6
03-01-2000	09:00:00	7.4
03-01-2000	12:00:00	8.4
03-01-2000	15:00:00	6.7
03-01-2000	18:00:00	5.7
03-01-2000	21:00:00	5.6
04-01-2000	09:00:00	4.4
04-01-2000	12:00:00	6.1
04-01-2000	15:00:00	3.8
04-01-2000	18:00:00	3.4
04-01-2000	21:00:00	3.1
05-01-2000	09:00:00	2.7
05-01-2000	12:00:00	5.4

1.7.4.6 Overblik over dækningsperiode i temperaturdata

Dette trin giver et hurtigt overblik over, hvilke datoer der er repræsenteret i temperaturdatasættet. Ved at konvertere datoen til et korrekt Date-format og udtrække unikke værdier skabes et klart billede af dataperiodens omfang. Antallet af distinkte datoer bruges som et simpelt, men effektivt kontrolpunkt til at vurdere datadækning og konsistens, inden de videre reduktioner og aggregeringer gennemføres.

```
# ============================================================================
#Distinct datoer (obs_dato som rigtig Date) - hurtig overblik
# ============================================================================
distinct_datoer <- fact_temperatur_dmi |>
  dplyr::mutate(obs_dato_date = as.Date(obs_dato, format = "%d-%m-%Y")) |>
  dplyr::distinct(obs_dato_date) |>
```

```
  dplyr::arrange(obs_dato_date)

cat("Antal distincte datoer i temperaturdata:", nrow(distinct_datoer), "\n\n")
```

1.7.4.7 Reduktion til observationer på hele timer

I dette trin fjernes al minutbaseret granularitet, så datasættet udelukkende består af observationer foretaget på hele timer. Formålet er at forenkle datastrukturen og sikre ensartede tidsintervaller, som er mere stabile og lettere at arbejde med i efterfølgende joins og analyser. Antallet af tilbageværende rækker udskrives som kontrol for at dokumentere effekten af reduktionen.

```
# ======================================================================
# Smid minut-data væk (kun hele timer)
# ======================================================================
temp_heltimer <- fact_temperatur_dmi |>
  dplyr::filter(substr(tid, 4, 5) == "00")

cat("Antal rækker efter filter til HELTIMER:", nrow(temp_heltimer), "\n\n")
```

1.7.4.8 Diagnostisk overblik over observationer pr. time

Dette trin anvendes som et diagnostisk kontrolpunkt, hvor antallet af temperaturobservationer opgøres for hver hele time i tidsrummet kl. 08:00–22:00. Formålet er at vurdere datadækning og identificere eventuelle skævheder eller mangler i bestemte timer. Resultatet giver et hurtigt, kvantitativt overblik over, hvilke tidspunkter der er bedst repræsenteret i datasættet, før der træffes endelige valg om tidsreduktion.

```
# ============================================================================
# 7) Diagnose: observationer pr. HEL time (08-22)
# ============================================================================
obs_pr_time <- temp_heltimer |>
  dplyr::mutate(time_hour = as.integer(substr(tid, 1, 2))) |>
  dplyr::count(time_hour, name = "antal_observationer") |>
  dplyr::arrange(dplyr::desc(antal_observationer))

cat("--- Observationer pr. HEL time (sorteret) ---\n")
print(obs_pr_time)
cat("\n")
```

Description: df [15 × 2]

time_hour <int>	antal_observationer <int>
21	9419
9	9406
12	9403
15	9403
18	9386
20	8341
16	8340
22	8339
19	8337
17	8334

1-10 of 15 rows Previous 1 2 Next

1.7.4.9 Bevidst valg af faste, kamprelevante tidspunkter

I dette trin fastlægges de endelige tidspunkter, som temperaturdata reduceres til: kl. 09:00, 12:00, 15:00 og 18:00. Valget er metodisk og forretningsmæssigt begrundet i ønsket om at repræsentere temperaturforhold før og omkring kampafvikling. Tidspunktet kl. 21:00 fravælges bevidst, da det typisk ligger efter kampens afslutning og derfor vurderes mindre relevant for analyse af tilskuere og kamprelaterede forhold.

```
# ======================================================================
# FASTE tidspunkter (BEVIDST VALG): 09, 12, 15, 18 (ikke 21)
# ======================================================================
valgte_tidspunkter <- c("09:00:00", "12:00:00", "15:00:00", "18:00:00")

cat("Bevidst valgte tidspunkter:", paste(valgte_tidspunkter, collapse = ", "), "\n")
cat("Note: 21:00:00 fravælges fordi det typisk er efter de fleste kampe.\n\n")
```

1.7.4.10 Reduktion af temperaturdata til udvalgte faste tidspunkter

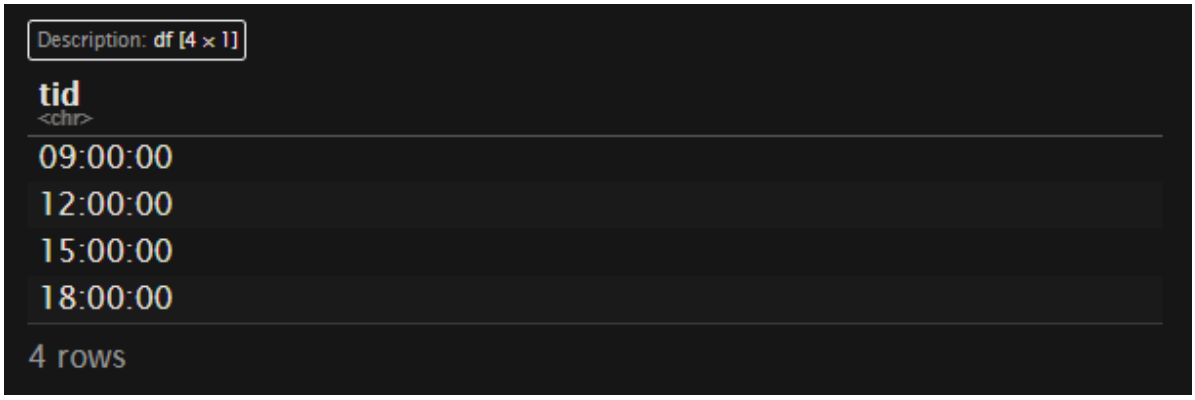
I dette trin reduceres temperaturdatasættet til udelukkende at indeholde observationer fra de fire på forhånd fastlagte tidspunkter kl. 09:00, 12:00, 15:00 og 18:00. Formålet er at skabe et kompakt og konsistent datasæt, som afspejler temperaturforhold før og omkring kampafvikling. Samtidig kontrolleres det eksplicit, at datasættet kun indeholder de ønskede

tidspunkter, hvilket sikrer metodisk stringens og forudsigelighed i de efterfølgende joins og analyser.

```
# ======================================================================
#  Reducér datasættet: behold kun de 4 tidspunkter
# ======================================================================
fact_temperatur_reduceret <- temp_heltimer |>
  dplyr::filter(tid %in% valgte_tidspunkter) |>
  dplyr::transmute(
    obs_dato   = obs_dato,
    tid        = tid,
    temperatur = temperatur
  )

cat("Antal rækker efter reduktion til 4 tidspunkter:", nrow(fact_temperatur_reduceret), "\n\n")

cat("Distinct tider i reduceret tabel:\n")
print(fact_temperatur_reduceret |> dplyr::distinct(tid) |> dplyr::arrange(tid))
cat("\n")
```



1.7.4.11 Kvalitetssikring af datadækning på valgte tidspunkter

Dette trin fungerer som et eksplicit kvalitetssikringscheck, hvor antallet af temperatur-observationer opgøres for hvert af de valgte tidspunkter. Formålet er at bekræfte, at alle fire tidspunkter er konsistent repræsenteret i datasættet, og at der ikke er utilsigtede udfald eller skævheder i datadækningen. Resultatet giver et klart overblik, som kan anvendes til at vurdere datasættets stabilitet før upload og videre anvendelse.

```
# ======================================================================
#  QA: observationer pr valgt tidspunkt
```

```
# ======================================================================
qa_valgte <- fact_temperatur_reduceret |>
  dplyr::count(tid, name = "antal_observationer") |>
  dplyr::arrange(tid)

cat("--- QA: observationer pr valgt tidspunkt ---\n")
print(qa_valgte)
cat("\n")
```

Description: df [4 × 2]

tid <chr>	antal_observationer <int>
09:00:00	9406
12:00:00	9403
15:00:00	9403
18:00:00	9386

4 rows

1.7.4.12 Visuelt eftersyn af reduceret temperatur-FACT

Dette trin giver mulighed for et manuelt, visuelt eftersyn af den reducerede temperatur-FACT-tabel. Ved at inspicere datasættet direkte kan man hurtigt verificere, at struktur, datoer, tidspunkter og temperaturværdier stemmer overens med de metodiske valg, der er foretaget. Selvom trinnet er valgfrit, er det et nyttigt supplement til de automatiske QA-checks, da det kan afsløre åbenlyse uregelmæssigheder, inden data uploades og anvendes videre.

```
# ======================================================================
#  VIEW: se den reducerede fact direkte (valgfrit men nyttigt)
# ======================================================================
View(fact_temperatur_reduceret, title = "fact_temperatur_reduceret (09/12/15/18)")
```

obs_dato	tid	temperatur
31-12-2024	09:00:00	3.6
31-12-2024	12:00:00	7.6
31-12-2024	15:00:00	7.0
31-12-2024	18:00:00	7.8
31-12-2023	09:00:00	5.7
31-12-2023	12:00:00	6.3
31-12-2023	15:00:00	5.6
31-12-2023	18:00:00	5.3
31-12-2022	09:00:00	7.8
31-12-2022	12:00:00	6.7
31-12-2022	15:00:00	5.7
31-12-2022	18:00:00	5.0
31-12-2021	09:00:00	8.3
31-12-2021	12:00:00	8.5
31-12-2021	15:00:00	8.3
31-12-2021	18:00:00	7.7
31-12-2020	09:00:00	1.0
31-12-2020	12:00:00	2.3
31-12-2020	15:00:00	1.9
31-12-2020	18:00:00	1.6
31-12-2019	09:00:00	6.1
31-12-2019	12:00:00	6.9

1.7.4.13 Diagnose af fuldstændighed på dato- og tidsniveau

Dette trin opbygger et særskilt diagnose-datasæt, der viser antallet af observationer for hver kombination af dato og valgt tidspunkt. Formålet er at skabe fuld transparens omkring datadækningen og gøre det muligt at identificere dage, hvor én eller flere af de faste tidspunkter mangler. Derudover udtrækkes eksplicit en oversigt over datoer, som ikke har alle fire tidspunkter repræsenteret, hvilket giver et klart beslutningsgrundlag for eventuel efterbehandling eller metodisk dokumentation.

```
# ======================================================================
#  Diagnose-datasæt: dato + tid + antal observationer
# ======================================================================
obs_pr_dato_tid <- fact_temperatur_reduceret |>
  dplyr::mutate(
```

```
    obs_dato_date = as.Date(obs_dato, format = "%d-%m-%Y")
  ) |>
  dplyr::count(obs_dato_date, tid, name = "antal_obs") |>
  dplyr::arrange(obs_dato_date, tid)

cat("Diagnose-datasæt (dato+tid+antal_obs) klar    Rækker:", nrow(obs_pr_dato_tid), "\n\n")
View(obs_pr_dato_tid, title = "DIAGNOSE: dato + tid + antal_obs")

# Ekstra: datoer der mangler mindst ét af de 4 tidspunkter
mangler_pr_dato <- obs_pr_dato_tid |>
  dplyr::group_by(obs_dato_date) |>
  dplyr::summarise(
    antal_tidspunkter = dplyr::n_distinct(tid),
    .groups = "drop"
  ) |>
  dplyr::filter(antal_tidspunkter < 4) |>
  dplyr::arrange(obs_dato_date)

cat("Antal datoer der IKKE har alle 4 tidspunkter:", nrow(mangler_pr_dato), "\n\n")
View(mangler_pr_dato, title = "Datoer med manglende tidspunkter (<4)")
```

```
Diagnose-datasæt (dato+tid+antal_obs) klar ✅  Rækker: 37598

Antal datoer der IKKE har alle 4 tidspunkter: 192
```

1.7.4.14 Robust upload af temperaturdata til Azure SQL

I dette trin uploades den reducerede temperatur-FACT-tabel til Azure SQL i laget PBA03_JoinReady ved hjælp af en robust retry-mekanisme. Uploaden er designet til at håndtere kendte og midlertidige forbindelsesproblemer, herunder 08S01 communication link failure, ved automatisk at gentage forsøget med korte pauser. Processen sikrer, at tabellen enten skrives korrekt til databasen eller fejler kontrolleret med en klar fejlmeddelelse, så dataintegriteten bevares og ETL-flowet afsluttes pålideligt.

```
# ======================================================================
#  Upload til Azure - robust retry (håndter 08S01 / link failure)
# ======================================================================
upload_retry <- function(df, forsøg_max = 5, delay_sec = 10) {
```

```
  forsøg <- 1
  repeat {
    cat("Upload-forsøg", forsøg, "...\n")
    ok <- try(
      DBI::dbWriteTable(
        conn      = con,
        name      = DBI::Id(schema = "PBA03_JoinReady", table = "fact_temperatur_join_ready")
        value     = df,
        overwrite = TRUE
      ),
      silent = TRUE
    )

    if (!inherits(ok, "try-error")) {
      cat("  Upload gennemført\n\n")
      break
    }

    if (forsøg >= forsøg_max) {
      stop("  Upload fejlede efter ", forsøg_max, " forsøg.\n\nFejl:\n", ok)
    }

    cat("  Upload fejlede - prøver igen om", delay_sec, "sekunder\n\n")
    Sys.sleep(delay_sec)
    forsøg <- forsøg + 1
  }
}

upload_retry(fact_temperatur_reduceret)
```

1.7.4.15 Luk forbindelsen til SQL Azure

```
# ======================================================================
# 15) Luk forbindelse
# ======================================================================
DBI::dbDisconnect(con)
cat("  Forbindelse til Azure SQL lukket.\n")
```