

## 2. Del Analyse, ML Model + Shiny

Jan Maurycy Pedersen, Jolene Kaye Jensen, Lukas Beslic Christiansen



# Indholdsfortegnelse

<b>1 Stam-baseline som analytisk referencepunkt</b>	<b>6</b>
1.1 Pakker og globale indstillinger	6
1.2 Sikker og reproducerbar databaseopsætning	7
1.3 Robust oprettelse af Azure SQL-forbindelse	7
1.4 Indlæsning af data fra Azure	8
1.5 Afgrænsning til Viborg FF's hjemmebanekampe	11
1.6 Sammenkædning af kampprogram og VFF-data	12
1.7 Sanity check af join og tidskonvertering	13
1.8 Validering af fuld join-dækning før baseline-modellering	14
1.9 Opbygning af fælles baseline-datasæt	14
1.10 Gemning og fastlæggelse af baseline	16
<b>2 Stam-baseline med kamptid</b>	<b>16</b>
2.1 Pakker og globale indstillinger	17
2.2 Sikker og reproducerbar Azure-forbindelse	17
2.3 Indlæsning af join-ready data fra Azure	19
2.4 Afgrænsning til Viborg FF's hjemmebanekampe	19
2.5 Sammenkædning af kampprogram og VFF-data	20
2.6 Sanity check af join og tidsfelter	22
2.7 Bekræftelse af fuld join-dækning	23
2.8 Konstruktion af stam-baseline (baseline1)	23
2.9 Gemning af baseline som fælles input til videre analyse	25
<b>3 ML model D3</b>	<b>25</b>
3.1 Setup: pakker og globale indstillinger	26
3.2 Hjælpefunktioner: miljøtjek og robust Azure SQL-forbindelse	26
3.2.1 Hjælpefunktion: sikker tabelindlæsning (retry + logging)	28
3.2.2 Hjælpefunktion: parse HH:MM(:SS) til time (0-23)	28
3.2.3 Hjælpefunktioner: evalueringsmål til modelperformance	29
3.2.4 Hjælpefunktioner: afgrænsning af output og robust kolonnevalg	29
3.2.5 Hjælpefunktion: fjern kolonner uden variation (0-varians)	30
3.2.6 Hjælpefunktion: ens dummy-kodning i train/test (model.matrix)	30
3.2.7 Hjælpefunktioner: tidsblokke til CV, sæson-start og robust faktorhåndtering	31
3.3 Connection: valider miljø og opret databaseforbindelse	31
3.4 Indlæsning af baseline: RDS, kvalitetstjek og konstruktion af kampstartstid	32
3.5 Baseline QA: deduplikering af kamp_id (én række pr. kamp)	33
3.6 Feature joins: berrig baseline med eksterne og kontekstuelle forklaringsvariabler	34
3.6.1 Helligdage: opbygning af helligdagsindikator og merge til baseline	34
3.6.2 SAH: indikator og antal håndboldkampe pr. dato (join til baseline)	35
3.6.3 Befolkning: match til kampdato med "nærmeste tidligere" observation	35
3.6.4 Vejr: match nærmeste observation til kickoff og konstruer model-venlige vejrvariable	36
3.6.5 Temperatur før kickoff (step-back samme dag)	38
3.6.6 Placering før kamp (feature til baseline)	39
3.6.7 Modstander (feature til baseline)	40
3.6.8 Samlet QA af baseline (tjek af manglende værdier og entydighed)	41
3.6.9 Billetsalg (D3/D7/D10): robust udtræk og "1 række pr. kamp"	42
3.6.10 Step-vækster + model-datasæt (D3): features, join og endelig klargøring	42
3.7 Tidssplit: kronologisk 80/20 opdeling i træning og test	44
3.8 Predictor pool: dynamisk valg af forklaringsvariabler	44

3.9	Best Subset . . . . .	49
3.9.1	Modelperformance for D3-modeller . . . . .	51
3.10	Regulariserede modeller (Ridge og Lasso) . . . . .	52
3.11	Fair sammenligning af D3-modeller med og uden befolkningstal . . . . .	57
3.11.1	Vurdering af FAIR POP-testen . . . . .	59
3.11.2	Betaler det sig at miste data for at beholde befolkningstal? . . . . .	59
3.12	Grafisk sammenligning og modelrepræsentation af de bedste D3-modeller . . . . .	60
3.13	Operationalisering af D3-modeller via Shiny-applikation . . . . .	69
3.14	To scenarier for tilskuertal i Superligaen baseret på D3-modellen . . . . .	77
3.14.1	Scenarie 1 – Høj interesse og næsten fyldt stadion . . . . .	77
3.14.2	Scenarie 2 – Lav efterspørgsel og behov for salgsindsats . . . . .	78
<b>4</b>	<b>ML model D7</b>	<b>78</b>
4.1	Pakker og globale indstillinger . . . . .	79
4.2	Hjælpefunktioner . . . . .	79
4.2.1	.Renviron . . . . .	79
4.2.2	Robust databaseforbindelse . . . . .	80
4.2.3	Sikker indlæsning af databaser . . . . .	80
4.2.4	Normalisering af tidsformater . . . . .	81
4.2.5	Udtræk af timekomponent . . . . .	81
4.2.6	Parsing af observationsdatoer . . . . .	81
4.2.7	Udledning af sæsonens startår . . . . .	81
4.2.8	Evalueringsmål . . . . .	81
4.2.9	Hjælpefunktioner til datarensning . . . . .	82
4.2.10	Samlet feature-testfunktion . . . . .	82
4.2.11	Miljø- og forbindelsestjek . . . . .	83
4.2.12	Indlæsning af baseline-datasæt . . . . .	84
4.2.13	Klargøring af kampstartstid og baseline-kvalitet . . . . .	84
4.2.14	Features & variabler . . . . .	85
4.2.14.1	Helligdage . . . . .	85
4.2.14.2	Samtidige håndboldkampe (SAH) . . . . .	86
4.2.14.3	Befolkning . . . . .	86
4.2.14.4	Vejrdata (nærmeste observation) . . . . .	87
4.2.14.5	Sammenkobling af vejrddata . . . . .	88
4.2.14.6	Robust håndtering af manglende vejrddata . . . . .	89
4.2.14.7	Temperatur før kamp . . . . .	90
4.2.14.8	Sammenkobling og håndtering af temperaturdata . . . . .	91
4.2.14.9	Billetsalg og vækstfeatures . . . . .	92
4.2.14.10	Konstruktion af D7-billetsalgsfeatures . . . . .	92
4.2.15	Diagnostik og datakvalitetstjek . . . . .	93
4.2.16	Endeligt D7-analysedatasæt . . . . .	93
4.2.17	Opsætning af baseline til modeltests . . . . .	94
4.2.18	Nulmodel (baseline) . . . . .	94
4.2.18.1	Feature-test: Helligdag . . . . .	95
4.2.18.2	Feature-test: Samtidig håndboldkamp (SAH) . . . . .	96
4.2.18.3	Feature-test: Temperatur (fair sammenligning) . . . . .	98
4.2.18.4	Feature-test: Placering før kamp . . . . .	101
4.2.18.5	Feature-test: Befolkning (fair test og omkostning) . . . . .	104
4.2.18.6	Samlet oversigt over variabeltests . . . . .	107
4.2.18.7	Endelig D7-model: tidssplit og modelsammenligning . . . . .	109
4.2.18.8	Hjælpere til ligningsformidling (OLS) . . . . .	110
4.2.18.9	Hjælpere til ligningsformidling (Ridge/Lasso) . . . . .	110

4.2.18.10	Tidssplit til modelvalidering . . . . .	112
4.2.18.11	Endelig OLS-model og testperformance . . . . .	112
4.2.19	Designmatricer til Ridge og Lasso . . . . .	113
4.2.19.1	Ridge-model og testperformance . . . . .	113
4.2.19.2	Lasso-model og testperformance . . . . .	114
4.2.20	Klargøring af plot-data og residualer . . . . .	116
4.2.20.1	Ligninger for de endelige modeller . . . . .	117
4.2.20.2	Plot: Faktiske vs. forudsagte tilskuere (OLS) . . . . .	117
4.2.20.3	Forberedelse af input-features til D7-modellen . . . . .	122
4.2.20.4	Prediction-helper til D7 (OLS) . . . . .	123
4.2.20.5	Prediction-helper til D7 (Ridge/Lasso) . . . . .	124
4.2.21	Afgrænsning af forudsigelser til stadionkapacitet . . . . .	124
4.2.22	Klargøring til Shiny-applikation . . . . .	124
4.2.23	Valg af standardmodel til Shiny . . . . .	125
4.2.24	Hjælpefunktioner til sæsoninput i Shiny . . . . .	127
4.2.25	Visualisering af bedste D7-model (OLS/Lasso) . . . . .	128
4.2.26	Shiny-applikation til D7-prognoser . . . . .	128
4.2.26.1	Serverlogik til Shiny-applikationen . . . . .	131
4.2.26.2	Case: D7-prediction som aktivt beslutningsværktøj . . . . .	133

## 5 ML model D10 134

5.0.1	Pakker og opsætning . . . . .	135
5.0.2	Hjælpefunktioner . . . . .	135
5.0.2.1	Miljøvariabler og adgang . . . . .	135
5.0.2.2	Sikker datalæsning fra SQL . . . . .	136
5.0.2.3	Dublet-håndtering pr kamp . . . . .	136
5.0.2.4	Sikker datokonvertering . . . . .	137
5.0.2.5	Sæsonens startår . . . . .	137
5.0.2.6	Negativ afskæring . . . . .	137
5.0.2.7	Modelmål for præcision . . . . .	137
5.0.3	Forbindelse til datagrundlag . . . . .	138
5.0.4	Indlæsning af baseline . . . . .	138
5.0.5	Dublettjek af baseline . . . . .	139
5.0.6	Sikring af kamp_tid . . . . .	139
5.0.7	Oprettelse af kamp_time_h . . . . .	140
5.0.8	Features . . . . .	140
5.0.8.1	Helligdage . . . . .	140
5.0.8.2	Håndboldkamp samme dag (SAH) . . . . .	140
5.0.8.3	Befolkning . . . . .	141
5.0.8.4	Vejr . . . . .	142
5.0.8.5	Temperatur . . . . .	144
5.0.8.6	Billetsalg . . . . .	146
5.0.9	D10-datasæt og analysegrundlag . . . . .	146
5.0.10	Testpipeline for D10-modellen . . . . .	148
5.0.10.1	Hjælpefunktioner til modeltest . . . . .	148
5.0.10.2	Enkelt variabeltest mod nulmodel . . . . .	148
5.0.11	Klargøring af D10-basisdatasæt . . . . .	149
5.0.12	D10 nulmodel . . . . .	150
5.0.13	Test af ekstra forklarende variable . . . . .	150
5.0.13.1	Test af placering før kamp . . . . .	154
5.0.13.2	Test af befolkningstal i D10-modellen (cost–benefit) . . . . .	155

5.1	OLS & Ridge & Lasso . . . . .	161
5.1.1	Tidsbaseret trænings- og testsplit (D10) . . . . .	162
5.1.2	Estimering af D10 OLS-baseline . . . . .	163
5.1.3	Regulariserede modeller: Ridge og Lasso . . . . .	163
5.1.3.1	Modelperformance og valg af bedste D10-model . . . . .	166
5.2	D10-model Endelig . . . . .	166
5.2.1	Hjælpefunktioner til modelligninger i plots . . . . .	166
5.2.2	Klargøring af D10-modelleringsdatasæt . . . . .	168
5.2.3	Tidsbaseret trænings- og testsplit . . . . .	168
5.2.4	Endelig D10 OLS-model . . . . .	169
5.2.5	Designmatricer til Ridge- og Lasso-modeller . . . . .	170
5.3	Ridge-regression (D10) . . . . .	170
5.4	Lasso-regression (D10) . . . . .	171
5.5	Samlet performance for D10-modeller . . . . .	172
5.6	Visualisering og gemning af D10-resultater . . . . .	173
5.7	Produktlag og interaktiv prognose (Shiny) . . . . .	183
5.8	Scenarie: Prognose som beslutningsværktøj før kampdag . . . . .	190
<b>6</b>	<b>LONG-model uden leakage</b>	<b>191</b>
6.1	Pakker og globale indstillinger . . . . .	192
6.2	Hjælpefunktioner . . . . .	192
6.2.0.1	Validering af miljøvariabler . . . . .	192
6.2.0.2	Azure SQL-forbindelse med retry . . . . .	192
6.2.0.3	Sikker indlæsning af tabel fra SQL . . . . .	193
6.2.0.4	Udtræk af sæsonens startår . . . . .	194
6.2.0.5	Evalueringsmål for modelperformance . . . . .	194
6.2.0.6	Year-split til træning og test . . . . .	194
6.2.0.7	Opbygning af modelmatrix . . . . .	195
6.2.0.8	Tilpasning af feature-kolonner . . . . .	195
6.2.0.9	Evaluering af modeloutput . . . . .	195
6.2.0.10	Robust gennemsnitsberegning . . . . .	195
6.2.0.11	Robust medianberegning . . . . .	196
6.2.0.12	Robust standardafvigelse . . . . .	196
6.2.0.13	Rolling historik uden leakage . . . . .	196
6.3	Initialisering af miljø og databaseforbindelse . . . . .	196
6.4	Indlæsning, validering og klargøring af baseline . . . . .	197
6.4.0.1	Indlæsning af baseline-fil . . . . .	197
6.4.0.2	Klargøring af baseline-data . . . . .	197
6.4.0.3	Validering af baseline-struktur . . . . .	197
6.4.0.4	Tjek for dubletter i baseline . . . . .	198
6.4.0.5	Håndtering af dubletter i baseline . . . . .	198
6.5	Feature load & transform . . . . .	198
6.5.1	Helligdage . . . . .	198
6.5.2	SAH-variabler sat til default . . . . .	199
6.5.3	Feature: befolkningstal (LOCF) . . . . .	199
6.5.4	Indlæsning af befolkningsdata . . . . .	199
6.5.5	Klargøring og filtrering af befolkningsdata . . . . .	199
6.5.6	Kobling af befolkningstal til kampe . . . . .	200
6.5.7	Kalenderbaserede features . . . . .	200
6.5.8	Tilskuer-historik med lag uden leakage . . . . .	201
6.5.8.1	Sæsonbaseret tilskuerhistorik . . . . .	201

6.5.9	Billetedata og sæsonbaseret historik uden leakage . . . . .	202
6.5.9.1	Indlæsning af billetsalgsdata . . . . .	202
6.5.9.2	Validering af billetsalgsdata . . . . .	202
6.5.9.3	Aggregering af billet salg pr. kamp . . . . .	203
6.5.9.4	Kobling af billet salg til baseline . . . . .	203
6.5.9.5	Sæsonbaserede billetaggregater . . . . .	203
6.5.9.6	Kobling af sæsonbaseret billehistorik . . . . .	204
6.5.9.7	Rolling kamp-historik uden leakage . . . . .	205
6.5.9.8	Rolling historikfeatures pr. kamp . . . . .	205
6.5.9.9	Filtrering til komplet LONG-datasæt . . . . .	206
6.5.9.10	Datasæt med og uden befolkningstal . . . . .	206
6.5.9.11	Definition af LONG feature-sæt . . . . .	207
6.5.10	Endelig modelkørsel . . . . .	208
6.5.10.1	Evalueringsfunktion . . . . .	212
6.5.10.2	Robust prediktionsfunktion baseret på bedste model . . . . .	213
6.6	Introduktion til scenarie-prognose-shiny app (LONG) . . . . .	216
6.6.1	Pakker . . . . .	216
6.6.2	Afledte hjælpefunktioner til scenarieinput . . . . .	216
6.6.3	Historiske referenceværdier og fallback-logik . . . . .	217
6.6.4	Datadrevet måned- og ugedagskorrektur samt scenarie-app . . . . .	218
6.7	En organisatorisk beslutning baseret på scenarie-prognosen . . . . .	223

## 1 Stam-baseline som analytisk referencepunkt

Dette QMD udgør projektets centrale stampunkt og etablerer den fælles baseline, som al efterfølgende analyse bygger videre på. Baseline fungerer ikke som en færdig eller låst model, men som et metodisk referencepunkt, der løbende udvides og nuanceres i analysen. Ved at starte fra et enkelt, lækagefrit grundsetup kan nye variable og antagelser tilføjes trinvis, så ændringer i modeladfærd og forklaringskraft kan forstås, dokumenteres og sammenlignes systematisk på tværs af prognosehorisonter. Desuden anvender vi de første to baselines til manuel validering og tjek og konsolidering af kode

### 1.1 Pakker og globale indstillinger

I dette indledende trin indlæses de nødvendige pakker på en måde, der sikrer et ensartet og reproducerbart setup på tværs af maskiner. Startup-beskeder undertrykkes for at holde output roligt og egnet til rapportering, mens globale indstillinger justeres, så numeriske værdier præsenteres uden videnskabelig notation. Dette skaber et stabilt og læsbart fundament for det videre analysearbejde.

```
# Vi undertrykker startup-messages for at holde output roligt og læsbart i en
# rapportkontekst. pacman::p_load sikrer ens setup på tværs af maskiner.
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(DBI, odbc, dplyr, lubridate, rstudioapi, hms)
})

# scipen = 999 fjerner videnskabelig notation, så tal fremstår mere intuitive.
options(scipen = 999)

cat("Pakker er nu klar - god arbejdslyst!\n\n")
```

Pakker er nu klar - god arbejdslyst!

## 1.2 Sikker og reproducerbar databaseopsætning

Dette trin sikrer en reproducerbar og sikker opsætning af forbindelsen til Azure SQL ved at anvende miljøvariabler til håndtering af credentials. Scriptet kontrollerer tidligt, om alle nødvendige variabler er til stede, og guider brugeren til korrekt opsætning, hvis noget mangler. Ved at fejle tidligt og tydeligt undgås skjulte forbindelsesfejl senere i analysen, og koden kan køres konsistent på tværs af maskiner uden hardcodede adgangsplysninger.

```
# Vi bruger miljøvariabler til credentials fra renviro, så scriptet kan køres på tværs af
# maskiner uden hardcodede nøgler. Hvis noget mangler, fejler vi tidligt og tydeligt.
```

```
ensure_renviro <- 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) {
    cat(" Følgende miljøvariabler mangler i .Renviron:\n")
    print(missing_vars)
    cat("\nÅbner ~/.Renviron - udfyld værdierne, gem, og genstart R.\n")
    file.edit("~/Renviron")

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

## 1.3 Robust oprettelse af Azure SQL-forbindelse

I dette trin etableres forbindelsen til Azure SQL ved hjælp af en bevidst robust retry-strategi. Da Azure-forbindelser i praksis kan fejle sporadisk på grund af timeouts, netværksforhold eller cold starts, forsøges forbindelsen oprettet flere gange med gradvist stigende timeouts. Tilgangen reducerer risikoen for tilfældige afbrydelser under rapportkørsler og sikrer, at analysen enten opnår en stabil databaseforbindelse eller stopper kontrolleret med en klar fejlmeddelelse.

```
DBI::dbDisconnect(con)
```

```
# Vi bruger en retry-funktion, fordi Azure-forbindelser i praksis kan fejle
# sporadisk (timeout, netværk, cold start). Metoden øger robusthed og minimerer
# "tilfældige fejl" under rapportkørsler.
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, "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")
      return(con)
    }

    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
  }

  stop("Uventet: forlod retry-loop uden forbindelse.")
}

ensure_renviro()
con <- connect_azure_retry()
on.exit(try(DBI::dbDisconnect(con), silent = TRUE), add = TRUE)

```

## 1.4 Indlæsning af data fra Azure

I dette trin indlæses de nødvendige datasæt direkte fra Azure SQL ved hjælp af `dbReadTable` uden brug af SQL-queries. Hele Superliga-programmet anvendes som strukturelt grundlag, mens Viborg FF's hjemmekampe med tilskuertal udgør målvariablen i analysen. Ved at trække data fra join-ready-laget sikres det, at datasættene allerede

er konsistente og klar til videre modellering, samtidig med at antallet af rækker udskrives som et simpelt kontrolpunkt for korrekt indlæsning.

```
# =====
# Hent data fra Azure (ingen SQL-queries, kun dbReadTable)
# =====
# program1 indeholder hele Superliga-programmet (mange klubber).
# VFF1 indeholder Viborg FF hjemmekampe med tilskuere/billetsalg (vores target).
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"),
  Encrypt  = "yes"
)

schema_program <- "PBA03_JoinReady"
schema_vff      <- "PBA03_JoinReady"
tbl_program     <- "fact_superliga_program_join_ready"
tbl_vff         <- "fact_VFF_Billetsalg_join_ready"

program1 <- DBI::dbReadTable(con, DBI::Id(schema = schema_program, table = tbl_program)) |> as_tibble()

VFF1      <- DBI::dbReadTable(con, DBI::Id(schema = schema_vff, table = tbl_vff)) |> as_tibble()

cat("program1 rækker:", nrow(program1), " | VFF1 rækker:", nrow(VFF1), "\n\n")
```

```
program1 rækker: 5248 | VFF1 rækker: 259
```

```
str(program1)
```

```
tibble [5,248 x 18] (S3: tbl_df/tbl/data.frame)
 $ kamp_id      : chr [1:5248] "2000/2001R1KLUB010" "2000/2001R1KLUB001" "2000/2001R1KLUB020" "2000/2001R1KLUB002" ...
 $ sæson        : chr [1:5248] "2000/2001" "2000/2001" "2000/2001" "2000/2001" ...
 $ runde        : int [1:5248] 1 1 1 1 1 1 2 2 2 2 ...
 $ ugedag        : chr [1:5248] "Lørdag" "Søndag" "Søndag" "Søndag" ...
 $ dato         : Date[1:5248], format: "2000-07-22" "2000-07-23" ...
 $ date_key     : chr [1:5248] "22072000" "23072000" "23072000" "23072000" ...
 $ tid          : chr [1:5248] "15:30:00" "15:00:00" "15:00:00" "17:05:00" ...
 $ hjemmehold   : chr [1:5248] "BIF" "AB" "FCM" "FCK" ...
 $ udehold      : chr [1:5248] "AGF" "SIF" "LBK" "SJF" ...
 $ hjemme_klubID: chr [1:5248] "KLUB010" "KLUB001" "KLUB020" "KLUB019" ...
 $ ude_klubID   : chr [1:5248] "KLUB003" "KLUB040" "KLUB033" "KLUB042" ...
 $ mål_hjemme   : int [1:5248] 1 1 4 5 0 2 0 3 1 1 ...
 $ mål_ude      : int [1:5248] 2 1 0 1 1 3 0 1 1 0 ...
 $ vinder       : chr [1:5248] "AGF" "Uafgjort" "FCM" "FCK" ...
 $ vinder_type  : chr [1:5248] "Udehold" "Uafgjort" "Hjemmehold" "Hjemmehold" ...
 $ tilskuertal  : chr [1:5248] "7806" "2411" "3706" "11977" ...
```

```
$ dommer      : chr [1:5248] "Johnny RÃ¸n" "Nicolai Vollquartz" "Knud Erik Fisker" "Claus Bo Larsen"
$ tv_kanal    : chr [1:5248] NA NA NA "TV3+" ...
```

```
str(VFF1)
```

```
tibble [259 x 14] (S3: tbl_df/tbl/data.frame)
 $ sæson      : chr [1:259] "2000/2001" "2000/2001" "2000/2001" "2000/2001" ...
 $ rundede    : num [1:259] 2 4 5 8 9 11 13 15 17 19 ...
 $ tilskuere  : num [1:259] 2089 3198 2378 3878 2017 ...
 $ år        : num [1:259] 2000 2000 2000 2000 2000 ...
 $ hold       : chr [1:259] "VFF- AB" "VFF-FCK" "VFF-SJF" "VFF-FCM" ...
 $ d10_tilskuere: num [1:259] 1087 1346 1114 1871 949 ...
 $ d7_tilskuere : num [1:259] 1282 2147 1607 2517 1103 ...
 $ d3_tilskuere : num [1:259] 1898 2735 2166 3278 1769 ...
 $ hold_raw   : chr [1:259] "VFF- AB" "VFF-FCK" "VFF-SJF" "VFF-FCM" ...
 $ hjemme_kort : chr [1:259] "VFF" "VFF" "VFF" "VFF" ...
 $ ude_kort   : chr [1:259] "AB" "FCK" "SJF" "FCM" ...
 $ hjemme_klubID: chr [1:259] "KLUB046" "KLUB046" "KLUB046" "KLUB046" ...
 $ ude_klubID  : chr [1:259] "KLUB001" "KLUB019" "KLUB042" "KLUB020" ...
 $ kamp_id    : chr [1:259] "2000/2001R2KLUB046" "2000/2001R4KLUB046" "2000/2001R5KLUB046" "2000/2001R8KLUB046" ...
```

```
> str(program1)
tibble [5,248 x 18] (S3: tbl_df/tbl/data.frame)
 $ kamp_id    : chr [1:5248] "2000/2001R1KLUB010" "2000/2001R1KLUB001" "2000/2001R1KLUB020" "2000/2001R1KLUB019" ...
 $ sæson      : chr [1:5248] "2000/2001" "2000/2001" "2000/2001" "2000/2001" ...
 $ rundede    : int [1:5248] 1 1 1 1 1 1 2 2 2 2 ...
 $ ugedag     : chr [1:5248] "Lørdag" "Søndag" "Søndag" "Søndag" ...
 $ dato       : Date[1:5248], format: ...
 $ date_key   : chr [1:5248] "22072000" "23072000" "23072000" "23072000" ...
 $ tid       : chr [1:5248] "15:30:00" "15:00:00" "15:00:00" "17:05:00" ...
 $ hjemmehold : chr [1:5248] "BIF" "AB" "FCM" "FCK" ...
 $ udehold    : chr [1:5248] "AGF" "SIF" "LBK" "SJF" ...
 $ hjemme_klubID: chr [1:5248] "KLUB010" "KLUB001" "KLUB020" "KLUB019" ...
 $ ude_klubID  : chr [1:5248] "KLUB003" "KLUB040" "KLUB033" "KLUB042" ...
 $ mål_hjemme : int [1:5248] 1 1 4 5 0 2 0 3 1 1 ...
 $ mål_ude    : int [1:5248] 2 1 0 1 1 3 0 1 1 0 ...
 $ vinder     : chr [1:5248] "AGF" "Uafgjort" "FCM" "FCK" ...
 $ vinder_type : chr [1:5248] "Udehold" "Uafgjort" "Hjemmehold" "Hjemmehold" ...
 $ tilskuertal : chr [1:5248] "7806" "2411" "3706" "11977" ...
 $ dommer     : chr [1:5248] "Johnny RÃ¸n" "Nicolai Vollquartz" "Knud Erik Fisker" "Claus Bo Larsen" ...
 $ tv_kanal   : chr [1:5248] NA NA NA "TV3+" ...
```

```
> str(VFF1)
tibble [259 x 14] (S3: tbl_df/tbl/data.frame)
 $ sæson      : chr [1:259] "2000/2001" "2000/2001" "2000/2001" "2000/2001" ...
 $ rundede    : num [1:259] 2 4 5 8 9 11 13 15 17 19 ...
 $ tilskuere  : num [1:259] 2089 3198 2378 3878 2017 ...
 $ år        : num [1:259] 2000 2000 2000 2000 2000 ...
 $ hold       : chr [1:259] "VFF- AB" "VFF-FCK" "VFF-SJF" "VFF-FCM" ...
 $ d10_tilskuere: num [1:259] 1087 1346 1114 1871 949 ...
 $ d7_tilskuere : num [1:259] 1282 2147 1607 2517 1103 ...
 $ d3_tilskuere : num [1:259] 1898 2735 2166 3278 1769 ...
 $ hold_raw   : chr [1:259] "VFF- AB" "VFF-FCK" "VFF-SJF" "VFF-FCM" ...
 $ hjemme_kort : chr [1:259] "VFF" "VFF" "VFF" "VFF" ...
 $ ude_kort   : chr [1:259] "AB" "FCK" "SJF" "FCM" ...
 $ hjemme_klubID: chr [1:259] "KLUB046" "KLUB046" "KLUB046" "KLUB046" ...
 $ ude_klubID  : chr [1:259] "KLUB001" "KLUB019" "KLUB042" "KLUB020" ...
 $ kamp_id    : chr [1:259] "2000/2001R2KLUB046" "2000/2001R4KLUB046" "2000/2001R5KLUB046" "2000/2001R8KLUB046" ...
```

## 1.5 Afgrænsning til Viborg FF's hjemmebanekampe

I dette trin afgrænses Superliga-programmet til udelukkende at omfatte Viborg FF's hjemmebanekampe. Da datasættet med billetsalg allerede kun indeholder VFF's hjemmekampe, anvendes dette filter for at sikre strukturel konsistens mellem kampprogram og målvariabel. Transformationen holdes bevidst minimal for at undgå utilsigtet tab af rækker, som tidligere er observeret ved mere aggressive deduplikeringer, og antallet af tilbageværende kampe anvendes som et simpelt kontrolpunkt.

```
# =====
# Afgrænsning: VFF hjemmebanekampe i programmet
# =====
# VFF1 indeholder kun VFF hjemmekampe. Programmet indeholder alt. Derfor
# filtrerer vi programmet til de rækker, hvor hjemme_klubID er Viborg FF.
#
# Vigtigt: I vores tidligere fejlsøgning viste det sig, at aggressive distinct() kunne "spise" rækker
# Her holder vi derfor transformationen minimal og kontrollerer match senere.
VFF_HOME_ID <- "KLUB046"

program2 <- program1 |>
  filter(hjemme_klubID == VFF_HOME_ID) |>
  transmute(
    kamp_id,
    dato,
    ugedag,
    tid
  )

cat("program2 (VFF hjemme) rækker:", nrow(program2), "\n")
```

```
program2 (VFF hjemme) rækker: 256
```

```
tibble(program2)
```

```
# A tibble: 256 x 4
  kamp_id      dato      ugedag tid
  <chr>      <date>    <chr> <chr>
1 2000/2001R2KLUB046 2000-07-30 Søndag 15:00:00
2 2000/2001R4KLUB046 2000-08-13 Søndag 15:00:00
3 2000/2001R5KLUB046 2000-08-20 Søndag 15:00:00
4 2000/2001R8KLUB046 2000-09-10 Søndag 15:00:00
5 2000/2001R9KLUB046 2000-09-17 Søndag 17:05:00
6 2000/2001R11KLUB046 2000-10-01 Søndag 15:00:00
7 2000/2001R13KLUB046 2000-10-22 Søndag 15:00:00
8 2000/2001R15KLUB046 2000-11-05 Søndag 15:00:00
9 2000/2001R17KLUB046 2000-11-19 Søndag 15:00:00
10 2000/2001R19KLUB046 2001-03-11 Søndag 15:00:00
# i 246 more rows
```

```
> tibble(program2)
# A tibble: 256 × 4
  kamp_id      dato      ugedag tid
  <chr>      <date>    <chr> <chr>
1 2000/2001R2KLUB046 2000-07-30 Søndag 15:00:00
2 2000/2001R4KLUB046 2000-08-13 Søndag 15:00:00
3 2000/2001R5KLUB046 2000-08-20 Søndag 15:00:00
4 2000/2001R8KLUB046 2000-09-10 Søndag 15:00:00
5 2000/2001R9KLUB046 2000-09-17 Søndag 17:05:00
6 2000/2001R11KLUB046 2000-10-01 Søndag 15:00:00
7 2000/2001R13KLUB046 2000-10-22 Søndag 15:00:00
8 2000/2001R15KLUB046 2000-11-05 Søndag 15:00:00
9 2000/2001R17KLUB046 2000-11-19 Søndag 15:00:00
10 2000/2001R19KLUB046 2001-03-11 Søndag 15:00:00
# i 246 more rows
# i Use `print(n = ...)` to see more rows
```

## 1.6 Sammenkædning af kampprogram og VFF-data

I dette trin kobles kampprogrammets oplysninger på datasættet med Viborg FF's hjemmekampe ved hjælp af `kamp_id` som fælles nøgle. VFF-datasættet behandles som master, så alle VFF-kampe bevares, mens `dato`, `ugedag` og `kickoff-tid` kun tilføjes, hvor der findes et gyldigt match i programmet. Datatyper håndteres eksplicit for at sikre, at `kampdato` og `kampstarttid` kan anvendes robust i den videre analyse og modellering.

```
# =====
# Join: tilføj programoplysninger til VFF1 (kamp_id som nøgle)
# =====
# Vi anvender left_join, fordi VFF1 er vores master: Vi vil bevare alle VFF-kampe
# og kun tilføje dato/ugedag/tid fra programmet hvor der findes match.
#
# Datatyper:
# - dato kommer fra Azure som Date og kan derfor beholdes som Date.
# - tid kommer som character og konverteres til hms, så den kan bruges til
#   at udlede kickoff-time robust.
VFF1_med_program <- VFF1 |>
  mutate(kamp_id = as.character(kamp_id)) |>
  left_join(
    program2 |> mutate(kamp_id = as.character(kamp_id)),
    by = "kamp_id"
  ) |>
  mutate(
    kamp_dato = as.Date(dato),
    kamp_tid = hms::as_hms(tid),
    kamp_time = as.integer(lubridate::hour(kamp_tid))
  )
View(VFF1_med_program);str(VFF1_med_program)
```

```
tibble [259 x 20] (S3: tbl_df/tbl/data.frame)
 $ sæson      : chr [1:259] "2000/2001" "2000/2001" "2000/2001" "2000/2001" ...
 $ runde      : num [1:259] 2 4 5 8 9 11 13 15 17 19 ...
 $ tilskuere  : num [1:259] 2089 3198 2378 3878 2017 ...
 $ år        : num [1:259] 2000 2000 2000 2000 2000 ...
 $ hold       : chr [1:259] "VFF- AB" "VFF-FCK" "VFF-SJF" "VFF-FCM" ...
 $ d10_tilskuere: num [1:259] 1087 1346 1114 1871 949 ...
 $ d7_tilskuere : num [1:259] 1282 2147 1607 2517 1103 ...
 $ d3_tilskuere : num [1:259] 1898 2735 2166 3278 1769 ...
 $ hold_raw   : chr [1:259] "VFF- AB" "VFF-FCK" "VFF-SJF" "VFF-FCM" ...
 $ hjemme_kort : chr [1:259] "VFF" "VFF" "VFF" "VFF" ...
 $ ude_kort   : chr [1:259] "AB" "FCK" "SJF" "FCM" ...
 $ hjemme_klubID: chr [1:259] "KLUB046" "KLUB046" "KLUB046" "KLUB046" ...
 $ ude_klubID  : chr [1:259] "KLUB001" "KLUB019" "KLUB042" "KLUB020" ...
 $ kamp_id     : chr [1:259] "2000/2001R2KLUB046" "2000/2001R4KLUB046" "2000/2001R5KLUB046" "2000/2001R8KLUB046" ...
 $ dato        : Date[1:259], format: "2000-07-30" "2000-08-13" ...
 $ ugedag      : chr [1:259] "Søndag" "Søndag" "Søndag" "Søndag" ...
 $ tid         : chr [1:259] "15:00:00" "15:00:00" "15:00:00" "15:00:00" ...
 $ kamp_dato   : Date[1:259], format: "2000-07-30" "2000-08-13" ...
 $ kamp_tid    : 'hms' num [1:259] 15:00:00 15:00:00 15:00:00 15:00:00 ...
 ..- attr(*, "units")= chr "secs"
 $ kamp_time   : int [1:259] 15 15 15 15 17 15 15 15 15 15 ...
```

```
> view(VFF1_med_program);str(VFF1_med_program)
tibble [259 x 20] (S3: tbl_df/tbl/data.frame)
 $ sæson      : chr [1:259] "2000/2001" "2000/2001" "2000/2001" "2000/2001" ...
 $ runde      : num [1:259] 2 4 5 8 9 11 13 15 17 19 ...
 $ tilskuere  : num [1:259] 2089 3198 2378 3878 2017 ...
 $ år        : num [1:259] 2000 2000 2000 2000 2000 ...
 $ hold       : chr [1:259] "VFF- AB" "VFF-FCK" "VFF-SJF" "VFF-FCM" ...
 $ d10_tilskuere: num [1:259] 1087 1346 1114 1871 949 ...
 $ d7_tilskuere : num [1:259] 1282 2147 1607 2517 1103 ...
 $ d3_tilskuere : num [1:259] 1898 2735 2166 3278 1769 ...
 $ hold_raw   : chr [1:259] "VFF- AB" "VFF-FCK" "VFF-SJF" "VFF-FCM" ...
 $ hjemme_kort : chr [1:259] "VFF" "VFF" "VFF" "VFF" ...
 $ ude_kort   : chr [1:259] "AB" "FCK" "SJF" "FCM" ...
 $ hjemme_klubID: chr [1:259] "KLUB046" "KLUB046" "KLUB046" "KLUB046" ...
 $ ude_klubID  : chr [1:259] "KLUB001" "KLUB019" "KLUB042" "KLUB020" ...
 $ kamp_id     : chr [1:259] "2000/2001R2KLUB046" "2000/2001R4KLUB046" "2000/2001R5KLUB046" "2000/2001R8KLUB046" ...
 $ dato        : Date[1:259], format: "2000-07-30" "2000-08-13" "2000-08-20" "2000-09-10" ...
 $ ugedag      : chr [1:259] "Søndag" "Søndag" "Søndag" "Søndag" ...
 $ tid         : chr [1:259] "15:00:00" "15:00:00" "15:00:00" "15:00:00" ...
 $ kamp_dato   : Date[1:259], format: "2000-07-30" "2000-08-13" "2000-08-20" "2000-09-10" ...
 $ kamp_tid    : 'hms' num [1:259] 15:00:00 15:00:00 15:00:00 15:00:00 ...
 ..- attr(*, "units")= chr "secs"
 $ kamp_time   : int [1:259] 15 15 15 15 17 15 15 15 15 15 ...
```

## 1.7 Sanity check af join og tidskonvertering

Dette trin fungerer som et afgørende sanity check efter joinet mellem kampprogram og VFF-data. Her kontrolleres det, om der opstår manglende værdier i kampdato, kampstarttid eller afledt kamp-time, hvilket typisk indikerer manglende match på kamp\_id eller uventede datatyper. I baseline-kontekst er fuld dækning afgørende, da manglende programoplysninger ellers kan føre til modellering på ufuldstændige eller misvisende data.

```
# Sanity check:
# Hvis join/konvertering giver NA på kamp_dato/kamp_tid, betyder det typisk
# manglende match på kamp_id eller uventet format. I baseline-kontekst ønsker vi
# fuld dækning, fordi vi ellers modellerer på "halv data".
join_diag <- VFF1_med_program |>
  summarise(
    rækker_total    = n(),
    uden_kamp_dato   = sum(is.na(kamp_dato)),
    uden_kamp_tid    = sum(is.na(kamp_tid)),
    uden_kamp_time   = sum(is.na(kamp_time))
  )

print(join_diag)
```

```
# A tibble: 1 x 4
  rækker_total uden_kamp_dato uden_kamp_tid uden_kamp_time
      <int>         <int>         <int>         <int>
1         259             0             0             0
```

## 1.8 Validering af fuld join-dækning før baseline-modellering

Dette afsluttende trin dokumenterer og håndhæver, at joint mellem billetsalgsdata og kampprogram er fuldstændigt. Konklusionen baseres på join-diagnostikken og viser, at alle VFF-kampe har gyldige værdier for kampdato og kampstarttid. Dermed er de centrale kalenderoplysninger fuldt tilgængelige, og baseline-modellen kan bygges uden risiko for at inkludere rækker med manglende programdata. Skulle der mod forventning opstå manglende match, stoppes processen eksPLICIT for at forhindre modellering på ufuldstændige data.

```
# Join-diagnostikken viser fuld dækning: 0 uden_kamp_dato, 0 uden_kamp_tid og
# 0 uden_kamp_time. Det betyder, at alle VFF-kampe i billetsalgsdatasættet har
# fundet et match i kampprogrammet på kamp_id, og at de centrale kalenderfelter
# derfor er komplet tilgængelige for baseline-modellering. Dermed undgår vi at
# bygge modellen på rækker med manglende programoplysninger.

if (join_diag$uden_kamp_dato > 0 || join_diag$uden_kamp_tid > 0) {
  cat("\n Manglende match efter join. Udsnit af problemrækker:\n\n")
  print(
    VFF1_med_program |>
      filter(is.na(kamp_dato) | is.na(kamp_tid)) |>
      select(kamp_id, sæson, runde, hjemme_klubID, ude_klubID, dato, tid, ugedag) |>
      head(20)
  )
  stop("Stopper: join-dækning fejlede. Baseline må ikke bygges på NA-join.")
}
```

## 1.9 Opbygning af fælles baseline-datasæt

I dette trin konstrueres det fælles baseline-datasæt, som fungerer som det analytiske referencepunkt for den videre modellering. Datasættet indeholder udelukkende variable, der er kendt før kampafvikling, og dermed undgås informationslækage allerede i udgangspunktet. Ved at kombinere sæson, runde, kampstarttid og ugedag med tilskuertallet

skabes et enkelt, men informativt grundlag, der kan indfange strukturelle og kalenderrelaterede forskelle mellem kampene. Baseline1 giver samtidig et klart overblik over variationen i tilskuertal og dokumenterer, at der er tilstrækkelig spredning i data til, at meningsfuld modellering kan gennemføres.

```
# =====
# 5) Baseline-datasæt (baseline1)
# =====
# Baseline konstrueres med variable, der er kendt før kampafvikling:
# - sæson: strukturelle forskelle mellem år (sportslig status, stadion, økonomi)
# - runde: position i sæsonforløb (momentum, vigtighed, vejr sæsonmæssigt)
# - kamp_time: kickoff-slot (tilgængelighed, tv-slot, transport)
# - ugedag: testes som kalender-effekt, men forventes ikke nødvendigvis stærk
#
# Vi filtrerer til rækker med responsvariabel (tilskuere) samt valid kamp_dato.
baseline1 <- VFF1_med_program |>
  filter(!is.na(kamp_dato), !is.na(tilskuere)) |>
  transmute(
    kamp_id    = as.character(kamp_id), # <-- TILFØJET: kamp_id fra VFF1 (nøgle, ikke feature)
    kamp_dato,
    sæson      = as.factor(sæson),
    runde      = as.integer(runde),
    ugedag     = as.factor(ugedag),
    kamp_time  = as.integer(kamp_time),
    tilskuere  = as.numeric(tilskuere)
  )

cat("\nbaseline1 rækker:", nrow(baseline1), "\n")
```

```
baseline1 rækker: 259
```

```
print(str(baseline1))
```

```
tibble [259 x 7] (S3: tbl_df/tbl/data.frame)
 $ kamp_id   : chr [1:259] "2000/2001R2KLUB046" "2000/2001R4KLUB046" "2000/2001R5KLUB046" "2000/2001R8K
 $ kamp_dato: Date[1:259], format: "2000-07-30" "2000-08-13" ...
 $ sæson     : Factor w/ 16 levels "2000/2001","2001/2002",...: 1 1 1 1 1 1 1 1 1 1 ...
 $ runde     : int [1:259] 2 4 5 8 9 11 13 15 17 19 ...
 $ ugedag    : Factor w/ 6 levels "Fredag","Lørdag",...: 5 5 5 5 5 5 5 5 5 5 ...
 $ kamp_time : int [1:259] 15 15 15 15 17 15 15 15 15 15 ...
 $ tilskuere : num [1:259] 2089 3198 2378 3878 2017 ...
NULL
```

```
print(summary(baseline1$tilskuere))
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1074	3314	4545	4779	6012	9523

```
View(baseline1)
```

```
# Baseline1 består af 259 observationer (VFF-hjemmekampe). Tilskuertallet varierer  
# fra 1074 til 9523, med median 4545 og gennemsnit 4778,7. Spredningen er tydelig,  
# hvilket gør det relevant at etablere en baseline-model, der kan forklare en del  
# af variationen ved hjælp af simple, tidligt kendte variable.
```

```
baseline1 rækker: 259  
tibble [259 × 7] (S3: tbl_df/tbl/data.frame)  
$ kamp_id   : chr [1:259] "2000/2001R2KLUB046" "2000/2001R4KLUB046" "2000/2001R5KLUB046" "2000/2001R8KLUB046" ...  
$ kamp_dato : Date[1:259], format: "2000-07-30" "2000-08-13" "2000-08-20" "2000-09-10" ...  
$ sæson     : Factor w/ 16 levels "2000/2001","2001/2002",...: 1 1 1 1 1 1 1 1 1 1 ...  
$ runde     : int [1:259] 2 4 5 8 9 11 13 15 17 19 ...  
$ ugedag    : Factor w/ 6 levels "Fredag","Lørdag",...: 5 5 5 5 5 5 5 5 5 5 ...  
$ kamp_time : int [1:259] 15 15 15 15 17 15 15 15 15 15 ...  
$ tilskuere : num [1:259] 2089 3198 2378 3878 2017 ...  
NULL  
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.  
  1074   3314   4545   4779   6012   9523
```

## 1.10 Gemning og fastlæggelse af baseline

I dette trin gemmes baseline-datasættet lokalt som et fælles udgangspunkt for alle prognosemodeller, så senere scripts kan genbruge det samme grundlag uden at gentage datatrin.

```
baseline_dir <- "C:/Users/janpe/OneDrive/Skrivebord/PBA Dataanalyse/01_Første semester/1 Semester proje  
  
# Opret mappen hvis den ikke findes (reproducerbarhed)  
dir.create(baseline_dir, recursive = TRUE, showWarnings = FALSE)  
  
baseline_file <- file.path(baseline_dir, "baseline_azure.rds")  
  
saveRDS(baseline1, baseline_file)  
  
cat("  Gemte baseline til:", baseline_file, "\n")
```

```
  Gemte baseline til: C:/Users/janpe/OneDrive/Skrivebord/PBA Dataanalyse/01_Første semester/1 Semester p
```

```
cat("  Working directory:", normalizePath(getwd()), "\n")
```

```
  Working directory: C:\Users\janpe\OneDrive\Skrivebord\PBA Dataanalyse\01_Første semester\1 Semester p
```

## 2 Stam-baseline med kamptid

Dette afsnit etablerer et fælles metodisk mere detaljeret startpunkt, som fungerer som afsæt for den videre analyse. Baseline anvendes ikke som en endelig model, men som et referencegrundlag, der løbende udvides og justeres, så effekten af nye variable og antagelser kan vurderes systematisk på tværs af prognosehorisonter.

## 2.1 Pakker og globale indstillinger

Her etableres et stabilt og reproducerbart analyse-setup ved at indlæse de nødvendige pakker på tværs af maskiner. Startup-beskeder undertrykkes for at holde output roligt i console og globale indstillinger justeres, så numeriske værdier præsenteres klart og uden videnskabelig notation.

```
# Vi undertrykker startup-messages for at holde output roligt og læsbart i en
# rapportkontekst. pacman::p_load sikrer ens setup på tværs af maskiner.
suppressPackageStartupMessages({
  if (!requireNamespace("pacman", quietly = TRUE)) install.packages("pacman")
  pacman::p_load(DBI, odbc, dplyr, lubridate, rstudioapi, hms, stringr)
})

# scipen = 999 fjerner videnskabelig notation, så tal fremstår mere intuitive.
options(scipen = 999)

cat("Pakker er nu klar - god arbejdslyst!\n\n")
```

Pakker er nu klar - god arbejdslyst!

## 2.2 Sikker og reproducerbar Azure-forbindelse

Dette trin sikrer både korrekt konfiguration af miljøvariabler og en stabil forbindelse til Azure SQL. Credentials håndteres via .Renviron for at undgå hardcoding og sikre reproducerbarhed på tværs af maskiner, mens en eksplicit retry-mekanisme reducerer risikoen for sporadiske forbindelsesfejl under analyse- og rapportkørsler.

```
DBI::dbDisconnect(con)

# =====
# Safeguard: .Renviron + Azure forbindelse (reproducerbarhed)
# =====
# Vi bruger miljøvariabler til credentials, så scriptet kan køres på tværs af
# maskiner uden hardcodede nøgler. Hvis noget mangler, fejler vi tidligt og
# tydeligt.
ensure_renviro <- 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) {
    cat(" Følgende miljøvariabler mangler i .Renviron:\n")
    print(missing_vars)
    cat("\nÅbner ~/.Renviron - udfyld værdierne, gem, og genstart R.\n")
    file.edit("~/Renviron")

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

```

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

# Vi bruger en retry-funktion, fordi Azure-forbindelser i praksis kan fejle
# sporadisk (timeout, netværk, cold start). Metoden øger robusthed og minimerer
# "tilfældige fejl" under rapportkørsler.
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, "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")
      return(con)
    }

    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
  }

  stop("Uventet: forlod retry-loop uden forbindelse.")
}

```

```

}

ensure_renviro()
con <- connect_azure_retry()
on.exit(try(DBI::dbDisconnect(con), silent = TRUE), add = TRUE)

```

## 2.3 Indlæsning af join-ready data fra Azure

I dette trin indlæses de nødvendige datasæt direkte fra Azure SQL ved hjælp af `dbReadTable` uden brug af SQL-queries. Superliga-programmet anvendes som strukturelt grundlag, mens Viborg FF's hjemmekampe med tilskuertal udgør målvariablen i analysen. Data hentes fra join-ready-laget for at sikre konsistens og klarhed i det videre analysearbejde.

```

# =====
# Hent data fra Azure (ingen SQL-queries, kun dbReadTable)
# =====
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"),
  Encrypt  = "yes"
)

# program1 indeholder hele Superliga-programmet (mange klubber).
# VFF1 indeholder Viborg FF hjemmekampe med tilskuere/billetsalg (vores target).
schema_program <- "PBA03_JoinReady"
schema_vff      <- "PBA03_JoinReady"
tbl_program     <- "fact_superliga_program_join_ready"
tbl_vff         <- "fact_VFF_Billetsalg_join_ready"

program1 <- DBI::dbReadTable(con, DBI::Id(schema = schema_program, table = tbl_program)) |> as_tibble()
VFF1     <- DBI::dbReadTable(con, DBI::Id(schema = schema_vff,      table = tbl_vff))      |> as_tibble()

cat("program1 rækker:", nrow(program1), " | VFF1 rækker:", nrow(VFF1), "\n\n")

```

```

program1 rækker: 5248 | VFF1 rækker: 259

```

## 2.4 Afgrænsning til Viborg FF's hjemmekampe

Her afgrænses kampprogrammet til udelukkende at omfatte Viborg FF's hjemmekampe, så det matcher billet-salgsdatasættet. Transformationen holdes bevidst minimal for at undgå utilsigtet tab af rækker, hvilket tidligere har vist sig at kunne opstå ved mere aggressive deduplikeringer.

```

# =====
# Afgrænsning: VFF hjemmekampe i programmet
# =====

```

```
# VFF1 indeholder kun VFF hjemmekampe. Programmet indeholder alt. Derfor
# filtrerer vi programmet til de rækker, hvor hjemme_klubID er Viborg FF.
#
# Vigtigt: I vores tidligere fejlsøgning viste det sig, at aggressive distinct()
# kunne "spise" rækker i praksis, afhængigt af hvordan data er blevet skrevet.
# Her holder vi derfor transformationen minimal og kontrollerer match senere.
VFF_HOME_ID <- "KLUB046"

program2 <- program1 |>
  filter(hjemme_klubID == VFF_HOME_ID) |>
  transmute(
    kamp_id,
    dato,
    ugedag,
    tid
  )

cat("program2 (VFF hjemme) rækker:", nrow(program2), "\n")
```

```
program2 (VFF hjemme) rækker: 256
```

## 2.5 Sammenkædning af kampprogram og VFF-data

I dette trin sammenkædes kampprogrammets oplysninger med VFF's billetsalgsdata ved hjælp af `kamp_id` som fælles nøgle. VFF-datasættet behandles som master, så alle hjemmekampe bevares, mens `dato` og `kickoff-tid` tilføjes, hvor der findes et match. Tidsfeltet parses robust for at håndtere forskellige formater og sikre korrekt afledning af kampstarttid, som senere anvendes i både baseline og feature-joins.

```
# =====
# Join: tilføj programoplysninger til VFF1 (kamp_id som nøgle)
# =====
# Vi anvender left_join, fordi VFF1 er vores master: Vi vil bevare alle VFF-kampe
# og kun tilføje dato/ugedag/tid fra programmet hvor der findes match.
#
# Datatyper:
# - dato kommer fra Azure som Date og kan derfor beholdes som Date.
# - tid kommer som character (eller nogle gange "grint" format) og konverteres til hms,
#   så den kan bruges til at udlede kickoff-time robust.
#

parse_tid_robust <- function(x) {
  # håndter NA
  if (all(is.na(x))) return(hms::as_hms(NA))

  # hvis det er numeric (Excel-tid), så er det typisk fraktion af døgn
  if (is.numeric(x)) {
    secs <- round(x * 86400)
    return(hms::as_hms(secs))
  }
}
```

```

x_chr <- stringr::str_trim(as.character(x))
x_chr <- ifelse(grepl("[0-9]{1,2}$", x_chr), paste0(x_chr, ":00:00"), x_chr)
x_chr <- ifelse(grepl("[0-9]{1,2}:[0-9]{2}$", x_chr), paste0(x_chr, ":00"), x_chr)

t <- suppressWarnings(hms::as_hms(x_chr))

# fallback: prøv lubridate::hm hvis as_hms ikke kan
if (all(is.na(t))) {
  hm_try <- suppressWarnings(lubridate::hm(x_chr))
  if (!all(is.na(hm_try))) t <- hms::as_hms(format(hm_try, "%H:%M:%S"))
}

return(t)
}

VFF1_med_program <- VFF1 |>
  mutate(kamp_id = as.character(kamp_id)) |>
  left_join(
    program2 |> mutate(kamp_id = as.character(kamp_id)),
    by = "kamp_id"
  ) |>
  mutate(
    kamp_dato = as.Date(dato),
    kamp_tid = parse_tid_robust(tid),
    kamp_time = as.integer(lubridate::hour(kamp_tid)),
    # kamp_dt_local er nøglen du skal bruge til vejrs senere
    kamp_dt_local = as.POSIXct(
      paste(kamp_dato, format(kamp_tid, "%H:%M:%S")),
      tz = "Europe/Copenhagen"
    )
  )

str(VFF1_med_program)

```

```

tibble [259 x 21] (S3: tbl_df/tbl/data.frame)
 $ sæson      : chr [1:259] "2000/2001" "2000/2001" "2000/2001" "2000/2001" ...
 $ runde      : num [1:259] 2 4 5 8 9 11 13 15 17 19 ...
 $ tilskuere  : num [1:259] 2089 3198 2378 3878 2017 ...
 $ år        : num [1:259] 2000 2000 2000 2000 2000 ...
 $ hold       : chr [1:259] "VFF- AB" "VFF-FCK" "VFF-SJF" "VFF-FCM" ...
 $ d10_tilskuere: num [1:259] 1087 1346 1114 1871 949 ...
 $ d7_tilskuere : num [1:259] 1282 2147 1607 2517 1103 ...
 $ d3_tilskuere : num [1:259] 1898 2735 2166 3278 1769 ...
 $ hold_raw   : chr [1:259] "VFF- AB" "VFF-FCK" "VFF-SJF" "VFF-FCM" ...
 $ hjemme_kort : chr [1:259] "VFF" "VFF" "VFF" "VFF" ...
 $ ude_kort   : chr [1:259] "AB" "FCK" "SJF" "FCM" ...
 $ hjemme_klubID: chr [1:259] "KLUB046" "KLUB046" "KLUB046" "KLUB046" ...
 $ ude_klubID  : chr [1:259] "KLUB001" "KLUB019" "KLUB042" "KLUB020" ...
 $ kamp_id    : chr [1:259] "2000/2001R2KLUB046" "2000/2001R4KLUB046" "2000/2001R5KLUB046" "2000/2001R6KLUB046" ...
 $ dato       : Date[1:259], format: "2000-07-30" "2000-08-13" ...
 $ ugedag     : chr [1:259] "Søndag" "Søndag" "Søndag" "Søndag" ...

```

```

$ tid          : chr [1:259] "15:00:00" "15:00:00" "15:00:00" "15:00:00" ...
$ kamp_dato    : Date[1:259], format: "2000-07-30" "2000-08-13" ...
$ kamp_tid     : 'hms' num [1:259] 15:00:00 15:00:00 15:00:00 15:00:00 ...
  ..- attr(*, "units")= chr "secs"
$ kamp_time    : int [1:259] 15 15 15 15 17 15 15 15 15 15 ...
$ kamp_dt_local: POSIXct[1:259], format: "2000-07-30 15:00:00" "2000-08-13 15:00:00" ...

```

```

> str(VFF1_med_program)
tibble [259 × 21] (S3: tbl_df/tbl/data.frame)
 $ sæson      : chr [1:259] "2000/2001" "2000/2001" "2000/2001" "2000/2001" ...
 $ runde      : num [1:259] 2 4 5 8 9 11 13 15 17 19 ...
 $ tilskuere  : num [1:259] 2089 3198 2378 3878 2017 ...
 $ år        : num [1:259] 2000 2000 2000 2000 2000 ...
 $ hold       : chr [1:259] "VFF- AB" "VFF-FCK" "VFF-SJF" "VFF-FCM" ...
 $ d10_tilskuere: num [1:259] 1087 1346 1114 1871 949 ...
 $ d7_tilskuere: num [1:259] 1282 2147 1607 2517 1103 ...
 $ d3_tilskuere: num [1:259] 1898 2735 2166 3278 1769 ...
 $ hold_raw   : chr [1:259] "VFF- AB" "VFF-FCK" "VFF-SJF" "VFF-FCM" ...
 $ hjemme_kort: chr [1:259] "VFF" "VFF" "VFF" "VFF" ...
 $ ude_kort   : chr [1:259] "AB" "FCK" "SJF" "FCM" ...
 $ hjemme_klubID: chr [1:259] "KLUB046" "KLUB046" "KLUB046" "KLUB046" ...
 $ ude_klubID  : chr [1:259] "KLUB001" "KLUB019" "KLUB042" "KLUB020" ...
 $ kamp_id     : chr [1:259] "2000/2001R2KLUB046" "2000/2001R4KLUB046" "2000/2001R5KLUB046" "2000/2001R8KLUB046" ...
 $ dato       : Date[1:259], format: "2000-07-30" "2000-08-13" "2000-08-20" "2000-09-10" ...
 $ ugedag     : chr [1:259] "Søndag" "Søndag" "Søndag" "Søndag" ...
 $ tid        : chr [1:259] "15:00:00" "15:00:00" "15:00:00" "15:00:00" ...
 $ kamp_dato   : Date[1:259], format: "2000-07-30" "2000-08-13" "2000-08-20" "2000-09-10" ...
 $ kamp_tid    : 'hms' num [1:259] 15:00:00 15:00:00 15:00:00 15:00:00 ...
  ..- attr(*, "units")= chr "secs"
 $ kamp_time   : int [1:259] 15 15 15 15 17 15 15 15 15 15 ...
 $ kamp_dt_local: POSIXct[1:259], format: "2000-07-30 15:00:00" "2000-08-13 15:00:00" "2000-08-20 15:00:00" "2000-09-10 15:00:00" ...

```

## 2.6 Sanity check af join og tidsfelter

Dette trin kontrollerer, at jointet mellem kampprogram og VFF-data er fuldstændigt, og at alle centrale tidsfelter er korrekt dannet. Eventuelle NA-værdier i kampdato, kampstartstid eller afledte tidsvariable indikerer typisk manglende match eller formatproblemer og er uacceptable i baseline-kontekst, hvor fuld datadækning er et krav.

```

# Sanity check:
# Hvis join/konvertering giver NA på kamp_dato/kamp_tid, betyder det typisk
# manglende match på kamp_id eller uventet format. I baseline-kontekst ønsker vi
# fuld dækning, fordi vi ellers modellerer på "halv data".

```

```

join_diag <- VFF1_med_program |>
  summarise(
    rækker_total      = n(),
    uden_kamp_dato    = sum(is.na(kamp_dato)),
    uden_kamp_tid     = sum(is.na(kamp_tid)),
    uden_kamp_time    = sum(is.na(kamp_time)),
    uden_kamp_dt_loc  = sum(is.na(kamp_dt_local))
  )

```

```
View(join_diag)
```

```
> print(join_diag)
# A tibble: 1 × 5
  rækker_total uden_kamp_dato uden_kamp_tid uden_kamp_time uden_kamp_dt_loc
      <int>         <int>         <int>         <int>         <int>
1         259             0             0             0             0
```

## 2.7 Bekræftelse af fuld join-dækning

Join-diagnostikken viser fuld dækning uden manglende værdier i kampdato, kickoff-tid eller lokal kamp-datetide. Det betyder, at alle VFF-hjemmekampe har et entydigt match i kampprogrammet, og at de centrale kalender- og tidsfelter er komplet tilgængelige. Dermed kan baseline bygges på et fuldt og konsistent datagrundlag uden risiko for modellering på ufuldstændige rækker.

```
# Join-diagnostikken viser fuld dækning: 0 uden_kamp_dato, 0 uden_kamp_tid og
# 0 uden_kamp_time. Det betyder, at alle VFF-kampe i billetsalgsdatasættet har
# fundet et match i kampprogrammet på kamp_id, og at de centrale kalenderfelter derfor er komplet tilgængelige.

if (join_diag$uden_kamp_dato > 0 || join_diag$uden_kamp_tid > 0 || join_diag$uden_kamp_dt_loc > 0) {
  cat("\n Manglende match efter join. Udsnit af problemrækker:\n\n")
  print(
    VFF1_med_program |>
      filter(is.na(kamp_dato) | is.na(kamp_tid) | is.na(kamp_dt_local)) |>
      select(kamp_id, sæson, runde, hjemme_klubID, ude_klubID, dato, tid, ugedag) |>
      head(20)
  )
  stop("Stopper: join-dækning fejlede. Baseline må ikke bygges på NA-join.")
}
```

## 2.8 Konstruktion af stam-baseline (baseline1)

I dette trin opbygges baseline1 som et fælles stam-datasæt baseret udelukkende på information, der er kendt før kampafvikling. Datasættet er bevidst konstrueret som et datagrundlag – ikke som en færdig model – og indeholder både nøglefelter og centrale kalender- og tidsvariable, så efterfølgende feature-scripts kan arbejde videre uden at gentage joins. Baseline1 fungerer dermed som det stabile referencepunkt for al videre analyse og modellering.

```
# =====
# Baseline-datasæt (baseline1)
# =====
#
# VIGTIGT:
# - Baseline er et dataset. Ikke en model.
# - Derfor gemmer vi også stamdatafelter (kamp_dato, kamp_tid, kamp_dt_local)
#   så alle feature-scripts kan arbejde videre uden at gen-joine programmet igen.
#
# Vi filtrerer til rækker med responsvariabel (tilskuere) samt valid kamp_dato.
baseline1 <- VFF1_med_program |>
  filter(!is.na(kamp_dato), !is.na(tilskuere)) |>
  transmute(
    kamp_id      = as.character(kamp_id), # nøgle
```

```

kamp_dato      = kamp_dato,          # dato (Date)
kamp_tid       = kamp_tid,           # tid (hms)
kamp_dt_local  = kamp_dt_local,      # fuld datetime (POSIXct, Europe/Copenhagen)
kamp_time      = as.integer(kamp_time), # time som tal

sæson         = as.factor(sæson),
runde         = as.integer(runde),
ugedag        = as.factor(ugedag),
tilskuere     = as.numeric(tilskuere)
)

cat("\nbaseline1 rækker:", nrow(baseline1), "\n")

```

baseline1 rækker: 259

```
print(str(baseline1))
```

```

tibble [259 x 9] (S3: tbl_df/tbl/data.frame)
 $ kamp_id      : chr [1:259] "2000/2001R2KLUB046" "2000/2001R4KLUB046" "2000/2001R5KLUB046" "2000/2001R8KLUB046" ...
 $ kamp_dato    : Date[1:259], format: "2000-07-30" "2000-08-13" ...
 $ kamp_tid     : 'hms' num [1:259] 15:00:00 15:00:00 15:00:00 15:00:00 ...
 ..- attr(*, "units")= chr "secs"
 $ kamp_dt_local: POSIXct[1:259], format: "2000-07-30 15:00:00" "2000-08-13 15:00:00" ...
 $ kamp_time    : int [1:259] 15 15 15 15 17 15 15 15 15 ...
 $ sæson       : Factor w/ 16 levels "2000/2001","2001/2002",...: 1 1 1 1 1 1 1 1 1 1 ...
 $ runde       : int [1:259] 2 4 5 8 9 11 13 15 17 19 ...
 $ ugedag      : Factor w/ 6 levels "Fredag","Lørdag",...: 5 5 5 5 5 5 5 5 5 5 ...
 $ tilskuere   : num [1:259] 2089 3198 2378 3878 2017 ...
NULL

```

```
print(summary(baseline1$tilskuere))
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1074	3314	4545	4779	6012	9523

```

baseline1 rækker: 259
> print(str(baseline1))
tibble [259 x 9] (S3: tbl_df/tbl/data.frame)
 $ kamp_id      : chr [1:259] "2000/2001R2KLUB046" "2000/2001R4KLUB046" "2000/2001R5KLUB046" "2000/2001R8KLUB046" ...
 $ kamp_dato    : Date[1:259], format: "2000-07-30" "2000-08-13" "2000-08-20" "2000-09-10" ...
 $ kamp_tid     : 'hms' num [1:259] 15:00:00 15:00:00 15:00:00 15:00:00 ...
 ..- attr(*, "units")= chr "secs"
 $ kamp_dt_local: POSIXct[1:259], format: "2000-07-30 15:00:00" "2000-08-13 15:00:00" "2000-08-20 15:00:00" "2000-09-10 15:00:00" ...
 $ kamp_time    : int [1:259] 15 15 15 15 17 15 15 15 15 ...
 $ sæson       : Factor w/ 16 levels "2000/2001","2001/2002",...: 1 1 1 1 1 1 1 1 1 1 ...
 $ runde       : int [1:259] 2 4 5 8 9 11 13 15 17 19 ...
 $ ugedag      : Factor w/ 6 levels "Fredag","Lørdag",...: 5 5 5 5 5 5 5 5 5 5 ...
 $ tilskuere   : num [1:259] 2089 3198 2378 3878 2017 ...
NULL
> print(summary(baseline1$tilskuere))
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 1074   3314   4545   4779   6012   9523

```

## 2.9 Gemning af baseline som fælles input til videre analyse

I dette trin gemmes stam-baseline lokalt som RDS-filer, så alle prognose- og feature-scripts kan tage udgangspunkt i det samme datasæt uden at gentage joins og typekonverteringer. Ved at gemme både en klassisk baseline og en eksPLICIT navngivet “future baseline feature” sikres et klart og ensartet udgangspunkt for videre modellering på tværs af prognosehorisonter.

```
# =====
# Gem baseline lokalt (genbrug i D10/D7/D3 og lang horisont)
# =====
# Vi gemmer baseline1 som RDS i projektets Baseline-mappe, så alle prognose-scripts
# kan starte fra samme datasæt uden at gentage joins og typekonvertering.
# Der bruges en absolut sti, så output placeres entydigt uafhængigt af working directory.

baseline_dir <- "C:/Users/janpe/OneDrive/Skrivebord/PBA Dataanalyse/01_Første semester/1 Semester proje

# Opret mappen hvis den ikke findes (reproducerbarhed)
dir.create(baseline_dir, recursive = TRUE, showWarnings = FALSE)

# Gem den "klassiske" baseline (som før)
baseline_file <- file.path(baseline_dir, "baseline_azure.rds")
saveRDS(baseline1, baseline_file)
cat(" Gemte baseline til:", baseline_file, "\n")
```

Gemte baseline til: C:/Users/janpe/OneDrive/Skrivebord/PBA Dataanalyse/01\_Første semester/1 Semester p

```
cat(" Working directory:", normalizePath(getwd()), "\n")
```

Working directory: C:\Users\janpe\OneDrive\Skrivebord\PBA Dataanalyse\01\_Første semester\1 Semester p

```
# Gem en "future baseline feature" (samme baseline, men tydeligt navngivet til feature-scripts)
# NOTE:
# - Den her fil er din nye standard input til feature scripts (vejr, mm.)
future_baseline_file <- file.path(baseline_dir, "future_baseline_feature_azure.rds")
saveRDS(baseline1, future_baseline_file)
cat(" Gemte FUTURE baseline feature til:", future_baseline_file, "\n")
```

Gemte FUTURE baseline feature til: C:/Users/janpe/OneDrive/Skrivebord/PBA Dataanalyse/01\_Første semest

## 3 ML model D3

Denne analyse implementerer en reproducerbar pipeline til at forudsige tilskuertal (D3) for VFF-kampe baseret på et samlet kampdatasæt. Data hentes fra Azure SQL og en lokal baseline (RDS), hvorefter centrale felter valideres, dubletter fjernes, og kampstartstid standardiseres til en numerisk timevariabel. Herefter konstrueres et feature-sæt ved at join'e helligdage, SAH-håndboldaktivitet, befolkningstal (matchet som nærmeste tidligere dato), vejr (nærmeste faste observationstid) og temperatur (seneste observation før kickoff). Yderligere inkluderes sportslige og kontekstuelle variable, herunder placering før kamp og modstander, med robust håndtering af faktorniveauer mellem træning og test. Det endelige analysedatasæt sorteres kronologisk og opdeles i et tidsbaseret 80/20 train-test split for at minimere

risikoen for fremtidskig. Modeller estimeres og sammenlignes på samme split via (1) stepwise OLS med AIC, (2) best subset selection med BIC og (3) ridge/lasso (glmnet) med krydsvalidering. For glmnet anvendes tidsblok-folds i CV for at bevare den tidslige struktur under tuning. Modelperformance evalueres konsekvent på testdata med RMSE, MAE, MSE og  $R^2$ , og resultater organiseres i strukturerede output-objekter til videre rapportering. Der indgår desuden en valgfri “fair” sammenligning, hvor modeller med og uden befolkning estimeres på samme N for en retvisende effektvurdering. Afslutningsvist genereres figurer (Predicted vs Actual) med modeltekst, og der etableres en Shiny-prototype til interaktivt modelvalg og prædiktio.

### 3.1 Setup: pakker og globale indstillinger

Kort sagt initialiseres analyse-miljøet ved først at sikre, at alle nødvendige R-pakker er tilgængelige, hvorefter de indlæses uden støjende opstartsbeskeder. Afslutningsvist justeres numerisk visning (så store tal ikke vises i videnskabelig notation), og der printes en kort statusmeddelelse, som bekræfter at miljøet er klar til resten af pipeline’en.

```
# Formål: sikre at alle afhængigheder findes, og at miljøet er ens hver gang scriptet køres.
needed_pkgs <- c("DBI","odbc","dplyr","lubridate","stringr","tibble","tidyr","ggplot2","leaps","glmnet")
# Find pakker der mangler (tjekker uden at load dem)
missing_pkgs <- needed_pkgs[
  !vapply(needed_pkgs, requireNamespace, logical(1), quietly = TRUE)]
# Stop tidligt hvis noget mangler (fail fast), så resten af pipeline'en ikke kører halvt
if (length(missing_pkgs) > 0)
  stop(" Manglende pakker: ", paste(missing_pkgs, collapse = ", "))
# Indlæs pakker (undertryk startup-messages for mere læsbart output)
suppressPackageStartupMessages({
  library(DBI); library(odbc)
  library(dplyr); library(tidyr)
  library(lubridate); library(stringr)
  library(tibble); library(ggplot2)
  library(leaps); library(glmnet)
})
# Global indstilling: undgå scientific notation i print (mere læsbart i rapport/log)
options(scipen = 999)
# Log: bekræft at setup er gennemført
cat("Pakker er klar \n\n")
```

Pakker er klar

### 3.2 Hjælpefunktioner: miljøtjek og robust Azure SQL-forbindelse

Før den egentlige dataindsamling defineres to centrale hjælpefunktioner. Først valideres, at nødvendige login-oplysninger til databasen findes som miljøvariabler i .Renviron, så forbindelsen kan etableres sikkert og reproducerbart. Dernæst etableres en robust forbindelsesfunktion til Azure SQL, som forsøger flere gange med stigende timeouts og korte pauser mellem forsøg, hvilket reducerer risikoen for, at midlertidige netværks- eller serverproblemer stopper hele pipeline’en.

```
# =====
# Hjælpefunktioner (ÉN gang)
# =====
ensure_renviron <- function() {
  # Definér hvilke miljøvariabler der skal være sat i .Renviron
```

```

required_vars <- c("AZURE_SQL_SERVER","AZURE_SQL_DB","AZURE_SQL_UID","AZURE_SQL_PWD")
# Hent værdierne fra miljøet
values <- Sys.getenv(required_vars)
# Find hvilke der mangler (tom streng)
missing_vars <- required_vars[values == ""]
# Stop tidligt hvis noget mangler (så vi ikke forsøger at forbinde med tomme credentials)
if (length(missing_vars) > 0) {
  stop(" Manglende miljøvariabler i .Renviron: ", paste(missing_vars, collapse = ", "))
} # Log: alt OK
cat(" .Renviron OK\n\n")
}

connect_azure_retry <- function(
  # Maks antal forbindelsesforsøg
  forsøg_max = 6,
  # Timeouts pr. forsøg (stigende, så vi giver mere tid ved senere forsøg)
  timeouts = c(60, 180, 200, 260, 360, 600),
  # Pause mellem forsøg (sekunder)
  delay_sec = 10
) {
  # Sørg for at miljøvariabler findes før vi forsøger at forbinde
  ensure_renviron()
  # Tæller for antal forsøg
  forsøg <- 1
  # Loop indtil connection lykkes eller vi rammer max forsøg
  while (forsøg <= forsøg_max) {
    # Vælg timeout til dette forsøg (brug sidste værdi hvis forsøg > længde(timeouts))
    timeout_brug <- timeouts[min(forsøg, length(timeouts))]
    # Log hvilket forsøg og timeout vi bruger
    cat("Forsøg", forsøg, "- ConnectionTimeout =", timeout_brug, "sekunder\n")
    # Forsøg at oprette connection (try så scriptet ikke stopper ved første fejl)
    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"),
        Encrypt = "yes",
        TrustServerCertificate = "no",
        ConnectionTimeout = timeout_brug
      ),
      silent = TRUE
    )
    # Hvis connection lykkes: returnér connection-objektet
    if (!inherits(con_try, "try-error")) {
      cat(" Forbundet til Azure SQL\n\n")
      return(con_try)
    }
    #Hvis sidste forsøg også fejler: stop med tydelig fejlbesked
  }
}

```

```

    if (forsøg == forsøg_max) stop(" Kunne ikke forbinde til Azure      SQL efter ", forsøg_max, " f
    # Vent og prøv igen
    Sys.sleep(delay_sec)
    forsøg <- forsøg + 1
  }
}

```

### 3.2.1 Hjælpefunktion: sikker tabelindlæsning (retry + logging)

Funktionen indkapsler indlæsning af tabeller fra databasen i en robust “retry”-mekanisme. Den forsøger at læse den ønskede tabel et fast antal gange, logger en kort fejlbesked ved hvert mislykket forsøg og venter kort mellem forsøgene. Hvis alle forsøg fejler, stoppes scriptet med en tydelig fejl, så pipeline’en ikke fortsætter på et ufuldstændigt datagrundlag.

```

# Læs en tabel robust med retry + fejllogging (stopper kun efter sidste forsøg)
safe_dbReadTable <- function(con, schema, table, forsøg_max = 4,      delay_sec = 3) {
  # Forsøg at læse tabellen op til forsøg_max gange
  for (i in seq_len(forsøg_max)) {
    # Forsøg at læse schema.table (try = ingen hard stop ved fejl)
    out <- try(DBI::dbReadTable(con, DBI::Id(schema = schema, table =      table)), silent = TRUE)
    # Hvis det lykkes: returnér data.frame med det samme
    if (!inherits(out, "try-error")) return(out)
    msg <- as.character(out)
    cat(" dbReadTable fejlede (forsøg", i, "af", forsøg_max, "):",      schema, ".", table, "\n")
    # Hvis det fejler: udtræk fejltekst og log kort besked (første 240      tegn)
    cat(substr(msg, 1, 240), "...\\n\\n")
    if (i < forsøg_max) Sys.sleep(delay_sec)
  }
  # Hvis alle forsøg fejler: stop med tydelig fejl
  stop(" Kunne ikke læse tabel: ", schema, ".", table)
}

```

### 3.2.2 Hjælpefunktion: parse HH:MM(:SS) til time (0–23)

Denne funktion bruges til at udtrække timen fra tidsstreng, så du kan lave robuste “nær kickoff”-matches. Den normaliserer først formatet (tilføjer :00 hvis sekunder mangler), og returnerer derefter timen som integer. Hvis formatet ikke kan parses, returneres NA.

```

# Hjælpefunktion: udtræk time fra tidsstreng (HH:MM eller HH:MM:SS)
time_chr_to_hour <- function(x) {

  # Sørg for at input er trimmed character
  x <- str_trim(as.character(x))

  # Hvis formatet er HH:MM, så gør det til HH:MM:SS ved at tilføje :00
  x <- if_else(str_detect(x, "^\\d{1,2}:\\d{2}$"), paste0(x, ":00"), x)

  # Tjek hvilke elementer der nu matcher HH:MM:SS
  ok <- str_detect(x, "^\\d{1,2}:\\d{2}:\\d{2}$")
}

```

```

# Output: default NA, udfyldt kun hvor parsing er mulig
out <- rep(NA_integer_, length(x))
out[ok] <- as.integer(str_extract(x[ok], "\\d{1,2}"))

out
}

```

### 3.2.3 Hjælpefunktioner: evalueringsmål til modelperformance

Her defineres en lille samling standardmål til at evaluere og sammenligne modeller. Funktionerne beregner henholdsvis RMSE, MAE og MSE som fejlmål samt  $R^2$  som forklaringsgrad, hvor der indbygges robusthed over for manglende værdier (NA). I  $R^2$ -funktionen håndteres specialtilfældet, hvor variationen i de observerede værdier er nul ( $sst = 0$ ), så der ikke opstår division med nul.

```

# RMSE: kvadratrods af gennemsnitlig kvadreret fejl (straffer store fejl relativt hårdt)
rmse_vec <- function(y, yhat) sqrt(mean((as.numeric(y) - as.numeric(yhat))^2, na.rm = TRUE))

# MAE: gennemsnitlig absolut fejl (mere robust end RMSE overfor outliers)
mae_vec <- function(y, yhat) mean(abs(as.numeric(y) - as.numeric(yhat)), na.rm = TRUE)

# MSE: gennemsnitlig kvadreret fejl (samme som RMSE men uden kvadratrods)
mse_vec <- function(y, yhat) mean((as.numeric(y) - as.numeric(yhat))^2, na.rm = TRUE)

# R²: 1 - (SSE/SST), dvs. hvor stor del af variationen i y modellen forklarer
# Håndterer specialtilfælde hvor SST = 0 (ingen variation i y) for at undgå division med 0
r2_vec <- function(y, yhat) {
  y <- as.numeric(y); yhat <- as.numeric(yhat)
  sse <- sum((y - yhat)^2, na.rm = TRUE)
  sst <- sum((y - mean(y, na.rm = TRUE))^2, na.rm = TRUE)
  if (isTRUE(all.equal(sst, 0))) return(NA_real_)
  1 - (sse / sst)
}

```

### 3.2.4 Hjælpefunktioner: afgrænsning af output og robust kolonnevalg

Disse to funktioner bruges som små, praktiske byggesten i pipeline'en. Den første sikrer, at numeriske outputs (fx predictioner) holdes inden for et realistisk interval ved at klippe værdier under 0 og over en valgt maksimumgrænse. Den anden finder den første kolonne i et datasæt, der matcher en liste af mulige kolonnenavne, hvilket gør koden robust over for variationer i navngivning på tværs af tabeller og versioner.

```

# Hjælpefunktion: klip værdier til et realistisk interval [0, cap]
# Formål: undgå negative predictioner og urealistisk høje værdier i output
clip_0_cap <- function(x, cap = 10000) pmax(pmin(as.numeric(x), as.numeric(cap)), 0)

# Hjælpefunktion: find første eksisterende kolonne blandt flere mulige navne
# Formål: gøre koden robust hvis kolonnenavne varierer mellem tabeller/versioner
find_first_col <- function(df, candidates) {
  hit <- candidates[candidates %in% names(df)][1]
  if (is.na(hit)) return(NA_character_)
}

```

```
hit
}
```

### 3.2.5 Hjælpefunktion: fjern kolonner uden variation (0-varians)

Funktionen bruges til at rense en designmatrix (fx fra `model.matrix`) ved at fjerne kolonner, som ikke varierer i data. Det er især relevant før variabelselektion eller regulariserede modeller, fordi 0-varians-kolonner ikke bidrager med information og kan skabe unødvendig kompleksitet. Hvis input er tomt eller `NULL`, returneres objektet uændret.

```
# Hjælpefunktion: fjern kolonner uden variation (0-varians)
# Formål: droppe uinformative features (fx dummy-kolonner der altid er 0/1 konstant),
# så model-fitting bliver mere stabilt og mindre støjfyldt.
drop_zero_variance_cols <- function(X) {
  # Hvis X er NULL eller tom (ingen kolonner), returnér uændret
  if (is.null(X) || ncol(X) == 0) return(X)
  # Beregn varians pr. kolonne (NA ignoreres)
  v <- apply(X, 2, function(col) stats::var(as.numeric(col), na.rm = TRUE))
  # Behold kun kolonner hvor variansen er defineret og større end 0
  keep <- !(is.na(v) | v == 0)
  # Returnér den filtrerede matrix (drop=FALSE bevarer matrix-format
  X[, keep, drop = FALSE]
}
```

### 3.2.6 Hjælpefunktion: ens dummy-kodning i train/test (model.matrix)

Formålet er at sikre, at train og test får præcis de samme dummy-kolonner, så modeller ikke fejler eller bliver inkonsistente pga. forskellige faktorniveauer. Det løses ved at kombinere datasættene før `model.matrix`, fjerne intercept-kolonnen og derefter splitte designmatricen tilbage i train og test.

```
# Hjælpefunktion: ens model.matrix for train/test (samme dummy-kolonner)
# Formål: undgå at train og test får forskellige kolonner pga. forskellige faktor-niveauer.
# Metode: kombiner train+test, byg model.matrix én gang, split derefter tilbage.
make_mm_train_test <- function(train_df, test_df, mm_formula) {
  # Tilføj en split-markør, så vi kan splitte korrekt efter bind_rows()
  train_df$.split <- "train"
  test_df$.split <- "test"
  # Kombiner data for at sikre identisk dummy-kodning på tværs af train/test
  combo <- bind_rows(train_df, test_df)
  # Byg designmatrix ud fra samme formel på hele datasættet
  X_combo <- stats::model.matrix(mm_formula, data = combo)
  # Fjern intercept-kolonne hvis den er med (mange workflows håndterer intercept separat)
  if ("(Intercept)" %in% colnames(X_combo)) {
    X_combo <- X_combo[, colnames(X_combo) != "(Intercept)", drop = FALSE]
  }
  # Split designmatrix tilbage til train og test (samme kolonner i begge)
  X_train <- X_combo[combo$.split == "train", , drop = FALSE]
  X_test <- X_combo[combo$.split == "test", , drop = FALSE]

  # Returnér begge matricer samlet
```

```
list(X_train = X_train, X_test = X_test)
}
```

### 3.2.7 Hjælpfunktioner: tidsblokke til CV, sæson-start og robust faktorhåndtering

Her defineres tre små hjælpefunktioner, som sikrer en mere realistisk validering og mere stabile modelkørsler. Først oprettes et foldid som sammenhængende tidsblokke til krydsvalidering, så træningen ikke blandes tilfældigt på tværs af tid. Dernæst udtrækkes sæsonens startår som numerisk variabel til modeller, hvor faktorer kan give unødigt mange dummyer. Til sidst standardiseres faktorniveauer mellem træning og test ved at mappe ukendte eller manglende niveauer til en fælles “ANDRE”-kategori, så predict() ikke fejler.

```
# CV foldid som tidsblokke (undgår random fremtidskig i træning)
# Formål: lave fold-inddeling til CV, hvor observationer holdes i sammenhængende tidsblokke.
# Dette reducerer leakage ift. tid (dvs. at "fremtiden" ikke blandes ind i træningen tilfældigt).
make_time_block_foldid <- function(n, k = 10) {
  # Hvis datasættet er mindre end antal folds, så sænk k (men mindst 2)
  if (n < k) k <- max(2, min(k, n))
  # Block size så alle rækker bliver fordelt på ~k blokke
  block_size <- ceiling(n / k)
  # Byg foldid: 1,1,1,...,2,2,2,... (tidsblokke) og klip til længde n
  rep(seq_len(k), each = block_size)[seq_len(n)]
}

# Udtræk sæsonens startår (fx "2022/2023" -> 2022)
# Formål: konvertere sæson til en numerisk variabel (ofte for at undgå mange dummy-kolonner).

season_start_year <- function(x) suppressWarnings(as.integer(substr(as.character(x), 1, 4)))

# Robust faktorhåndtering mellem train/test
# Formål: sikre at test ikke indeholder nye/ukendte niveauer som får predict()/model.matrix til at fejle
# Metode: niveauer defineres ud fra train; ukendte eller NA i test mappes til "ANDRE".
safe_factor_with_other <- function(train_vec, test_vec, other_level = "ANDRE") {
  train_chr <- as.character(train_vec)
  test_chr <- as.character(test_vec)
  lvls <- sort(unique(train_chr))
  if (!(other_level %in% lvls)) lvls <- c(lvls, other_level)
  test_chr[is.na(test_chr) | !(test_chr %in% lvls)] <- other_level
  list(
    train = factor(train_chr, levels = lvls),
    test = factor(test_chr, levels = lvls)
  )
}
```

## 3.3 Connection: valider miljø og opret databaseforbindelse

Denne del af scriptet initialiserer forbindelsen til Azure SQL. Først verificeres, at nødvendige miljøvariabler er sat korrekt i .Renviron, så credentials ikke hardcodes i koden. Derefter oprettes forbindelsen med retry-logik, og der registreres en “cleanup”-handling, så forbindelsen altid lukkes pænt igen ved afslutning eller fejl, hvilket gør pipeline’en mere stabil og reproducerbar.

```
# Validér at nødvendige miljøvariabler til Azure SQL findes (stopper hvis noget mangler)
ensure_renviro()
```

```
.Renviron OK
```

```
# Opret forbindelse til Azure SQL (med retry-logik i connect_azure_retry)
con <- connect_azure_retry()
```

```
.Renviron OK
```

```
Forsøg 1 - ConnectionTimeout = 60 sekunder
Forbundet til Azure SQL
```

```
# Sørg for at forbindelsen altid lukkes pænt ved script-exit (også ved fejl)
# add = TRUE gør, at denne on.exit ikke overskriver andre on.exit-calls
```

### 3.4 Indlæsning af baseline: RDS, kvalitetstjek og konstruktion af kampstarttid

I dette afsnit indlæses baseline-datasættet fra en lokal RDS-fil, og nøglevariable standardiseres til korrekte datatyper, så datasættet kan bruges konsistent i resten af pipeline'en. Herefter kontrolleres, at de nødvendige kolonner findes, og der udskrives en kort status for antal rækker og unikke kamp-id'er. Hvis kampstarttidspunktet (kamp\_tid) mangler i baseline-filen, hentes det fra et sekundært baseline-feature-datasæt og joines ind på kamp\_id, således at kamp-tid altid er tilgængelig. Til sidst konverteres kamp-tid til en numerisk timevariabel (kamp\_time\_h), og datasættet kvalitetssikres ved at identificere og håndtere dubletter, hvor der vælges én række pr. kamp\_id (prioriteret efter færrest manglende værdier).

```
# Sæt mappe og filnavn til baseline-RDS (lokal fil)
baseline_dir <- "C:/Users/janpe/OneDrive/Skrivebord/PBA Dataanalyse/01_Første semester/1 Semester proj
baseline_file <- file.path(baseline_dir, "baseline_azure.rds")
# Stop hvis baseline-filen ikke findes (fail fast)
stopifnot(file.exists(baseline_file))

# Indlæs baseline og standardisér centrale datatyper
# - kamp_id: character (sikrer stabile joins)
# - kamp_dato: Date (sikrer stabile dato-joins)
baseline <- readRDS(baseline_file) %>%
  mutate(
    kamp_id = as.character(kamp_id),
    kamp_dato = as.Date(kamp_dato)
  )
# Log: print størrelse og antal unikke kampe
stopifnot(all(c("kamp_id", "kamp_dato", "tilskuere") %in% names(baseline)))
# Log: print størrelse og antal unikke kampe
cat("Baseline loaded   Rækker:", nrow(baseline),
    "Unikke kamp_id:", n_distinct(baseline$kamp_id), "\n\n")
```

```
Baseline loaded   Rækker: 259 Unikke kamp_id: 254
```

```
# sikr kamp_tid
if (!("kamp_tid" %in% names(baseline))) {
  cat(" baseline_azure.rds har ikke 'kamp_tid'. Henter fra future_baseline_feature_azure.rds...\n")
  feature_baseline_file <- file.path(baseline_dir, "future_baseline_feature_azure.rds")
  stopifnot(file.exists(feature_baseline_file))

  fb <- readRDS(feature_baseline_file) %>%
    mutate(kamp_id = as.character(kamp_id), kamp_dato = as.Date(kamp_dato)) %>%
    arrange(kamp_id) %>%
    group_by(kamp_id) %>%
    slice(1) %>%
    ungroup() %>%
    select(kamp_id, kamp_tid)

  baseline <- baseline %>% left_join(fb, by = "kamp_id")
  rm(fb)
  cat(" kamp_tid joinet ind\n\n")
}

baseline <- baseline %>%
  mutate(
    kamp_tid_chr = str_trim(as.character(kamp_tid)),
    kamp_tid_chr = if_else(str_detect(kamp_tid_chr, "^\\d{1,2}:\\d{2}$"), paste0(kamp_tid_chr, ":00"),
    kamp_time_h = time_chr_to_hour(kamp_tid_chr)
  )

cat("Kamp-rækker uden parsebar kamp_time_h:", sum(is.na(baseline$kamp_time_h)), "\n\n")
```

Kamp-rækker uden parsebar kamp\_time\_h: 0

### 3.5 Baseline QA: deduplikering af kamp\_id (én række pr. kamp)

Dette afsnit kvalitetssikrer baseline ved at finde kamp-id'er, der forekommer flere gange. Hvis der findes dubletter, vælges den “bedste” række pr. kamp\_id ved at prioritere observationen med færrest manglende værdier, så datatabet minimeres. Til sidst valideres det med et tjek, at baseline nu har præcis én række pr. kamp.

```
# dedup kamp_id
dup_ids <- baseline %>% count(kamp_id, sort = TRUE) %>% filter(n > 1)
cat("--- BASELINE QA: DUPLIKAT-TJEK ---\n")
```

--- BASELINE QA: DUPLIKAT-TJEK ---

```
cat("Antal kamp_id med dubletter:", nrow(dup_ids), "\n")
```

Antal kamp\_id med dubletter: 5

```

if (nrow(dup_ids) > 0) {
  baseline2 <- baseline
  baseline2$.na_count <- rowSums(is.na(baseline2))
  baseline <- baseline2 %>%
    arrange(kamp_id, .na_count) %>%
    group_by(kamp_id) %>%
    slice(1) %>%
    ungroup() %>%
    select(-.na_count)
  stopifnot(nrow(baseline) == n_distinct(baseline$kamp_id))
  cat(" Baseline deduplikeret (1 række pr kamp_id)\n\n")
} else {
  cat(" Ingen dubletter fundet\n\n")
}

```

Baseline deduplikeret (1 række pr kamp\_id)

### 3.6 Feature joins: berrig baseline med eksterne og kontekstuelle forklaringsvariable

I dette afsnit udvides baseline-datasættet med et sæt forklaringsvariable, der forventes at påvirke tilskuertallet. Først konstrueres en helligdagsindikator på datoniveau og joines ind på kampdato. Dernæst beregnes samtidige SAH-håndboldkampe pr. dato (både indikator og antal) og joines på kampdato. Herefter tilføjes Viborgs befolkningstal ved at matche hver kampdato til nærmeste tidligere registrering, så variabelen er tidskonsistent. Endelig joines vejr og temperatur på kampniveau ved at vælge den observation, der ligger tættest på kickoff inden for faste tidsgrid (vejr) samt den seneste temperaturmåling før kickoff samme dag (temperatur). Der etableres samtidig robuste “mangler”-indikatorer og modelvenlige kodninger (fx vejrcode\_model), så manglende/ikke-observerbare observationer håndteres eksplicit i senere modellering.

#### 3.6.1 Helligdage: opbygning af helligdagsindikator og merge til baseline

Her hentes en helligdagskalender fra databasen og reduceres til ét simpelt flag pr. dato (1 = helligdag). Flaget joines derefter ind i kampdata på kampdato, så hver kamp får en entydig variabel. Manglende match efter join fortolkes som “ikke helligdag” og omkodes derfor til 0, så variabelen er klar til modellering uden NA’er.

```

hellig_raw <- safe_dbReadTable(con,"PBA01_Raw", "dim_helligdage_dkk_raw")
stopifnot(all(c("dato","helligdag_navn") %in% names(hellig_raw)))

hellig_min <- hellig_raw %>%
  mutate(hellig_dato = as.Date(dato)) %>%
  filter(!is.na(hellig_dato)) %>%
  group_by(hellig_dato) %>%
  summarise(er_helligdag = 1L, .groups = "drop")

baseline <- baseline %>%
  left_join(hellig_min, by = c("kamp_dato" = "hellig_dato")) %>%
  mutate(er_helligdag = if_else(is.na(er_helligdag), 0L, er_helligdag))

cat("Helligdage jointet \n\n")

```

Helligdage jointet

### 3.6.2 SAH: indikator og antal håndboldkampe pr. dato (join til baseline)

Her konstrueres et simpelt event-signal for SAH på dagsniveau. Først hentes håndboldkampene, hvorefter der aggregeres pr. dato til (1) en binær indikator for om der afvikles mindst én kamp og (2) et antal kampe den pågældende dag. Disse features joines derefter ind i baseline på kampdato, og manglende værdier efter join omkodes til 0, så variablerne kan anvendes direkte i modellering.

```
# 4B) SAH
sah_raw <- safe_dbReadTable(con, "PBA02_Clean", "fact_håndboldkampe_SAH_clean")
stopifnot(all(c("kamp_dato", "kamp_tid", "Event") %in% names(sah_raw)))

sah_min <- sah_raw %>%
  mutate(sah_dato = as.Date(kamp_dato)) %>%
  filter(!is.na(sah_dato)) %>%
  group_by(sah_dato) %>%
  summarise(
    er_håndboldkamp_SAH = 1L,
    antal_håndboldkampe = n(),
    .groups = "drop"
  )

baseline <- baseline %>%
  left_join(sah_min, by = c("kamp_dato" = "sah_dato")) %>%
  mutate(
    er_håndboldkamp_SAH = if_else(is.na(er_håndboldkamp_SAH), 0L, er_håndboldkamp_SAH),
    antal_håndboldkampe = if_else(is.na(antal_håndboldkampe), 0L, antal_håndboldkampe)
  )

cat("Håndbold SAH joinet \n\n")
```

Håndbold SAH joinet

### 3.6.3 Befolkning: match til kampdato med “nærmeste tidligere” observation

Her hentes Viborgs befolkningsdata og renses til én samlet tidsserie (kun totaler for køn og civilstand). Derefter matches hver kamp til den seneste befolkningsobservation, der ligger på eller før kampdatoen, så vi undgår at bruge “fremtidige” værdier. Til sidst joines befolkningstal ind på kampniveau via kamp\_id, så baseline fortsat har én række pr. kamp.

```
# 4C) Befolkning (closest tidligere dato)
bef_raw <- safe_dbReadTable(con, "PBA03_JoinReady", "fact_Viborg_befolkning_join_ready")
stopifnot(all(c("køn", "civilstand", "dato", "befolkningstal") %in% names(bef_raw)))

bef_all <- bef_raw %>%
  mutate(
    køn = str_to_lower(trimws(as.character(køn))),
    civilstand = str_to_lower(trimws(as.character(civilstand))),
    dato = as.Date(dato),
    befolkningstal = as.numeric(befolkningstal)
  ) %>%
```

```

filter(
  !is.na(dato), !is.na(befolkningstal),
  køn %in% c("i alt", "alt"),
  civilstand %in% c("i alt", "alt")
) %>%
distinct(dato, .keep_all = TRUE) %>%
arrange(dato) %>%
transmute(bef_dato = dato, befolkningstal = befolkningstal)

baseline_bef <- baseline %>%
transmute(kamp_id, kamp_dato) %>%
left_join(bef_all, join_by(closest(kamp_dato >= bef_dato)))

baseline <- baseline %>%
left_join(baseline_bef %>% select(kamp_id, befolkningstal), by = "kamp_id")

cat("Befolkning joinet \n\n")

```

Befolkning joinet

### 3.6.4 Vejr: match nærmeste observation til kickoff og konstruér model-venlige vejrvariable

Her hentes vejrdato og renses til et format, der kan matches stabilt til hver kamp. Matching sker ved først at begrænse til faste observationstidspunkter (08, 11, 14, 17) og derefter vælge den observation, der ligger tættest på kampens kickoff-time. Hvis der ikke findes en grid-observation for kampdatoen, anvendes en fallback (første observation på dato). Til sidst joines vejr ind på kampniveau, og der oprettes eksplicitte “mangler”-indikatorer samt kodninger (vejrkode\_model og vejrbeskrivelse\_model), så manglende/ikke-observerbare forhold håndteres konsistent i modelleringen.

```

# 4D) VEJR (nærmeste blandt 8/11/14/17)
cat("\n--- FEATURE: VEJR ---\n")

```

--- FEATURE: VEJR ---

```

vejr_sql <- safe_dbReadTable(con, "PBA03_JoinReady", "fact_vejr_join_ready")
stopifnot(all(c("obs_dato", "tid", "vejrkode", "vejrbeskrivelse") %in% names(vejr_sql)))

vejr_grid_tider_h <- c(8L, 11L, 14L, 17L)

vejr_pre <- vejr_sql %>%
mutate(
  vejr_dato = if (inherits(obs_dato, "Date")) obs_dato else as.Date(obs_dato, format = "%d-%m-%Y"),
  vejr_tid_chr = str_trim(as.character(tid)),
  vejr_tid_chr = if_else(str_detect(vejr_tid_chr, "^\\d{1,2}:\\d{2}$"), paste0(vejr_tid_chr, ":00"),
  vejr_time_h = time_chr_to_hour(vejr_tid_chr),
  vejrkode = suppressWarnings(as.integer(vejrkode)),
  vejrbeskrivelse = as.character(vejrbeskrivelse)
) %>%

```

```

filter(!is.na(vejr_dato), !is.na(vejr_time_h))

vejr_grid <- vejr_pre %>%
  filter(vejr_time_h %in% vejr_grid_tider_h) %>%
  group_by(vejr_dato, vejr_time_h) %>%
  slice(1) %>%
  ungroup()

vejr_fallback_1 <- vejr_pre %>%
  arrange(vejr_dato, vejr_time_h) %>%
  group_by(vejr_dato) %>%
  slice(1) %>%
  ungroup()

baseline_key <- baseline %>% transmute(kamp_id, kamp_dato, kamp_time_h)

kandidater_vejr <- baseline_key %>%
  filter(!is.na(kamp_dato), !is.na(kamp_time_h)) %>%
  inner_join(vejr_grid %>% select(vejr_dato, vejr_time_h, vejrcode, vejrbeskrivelse),
    by = c("kamp_dato" = "vejr_dato")) %>%
  mutate(dist_hours_to_kickoff = abs(kamp_time_h - vejr_time_h)) %>%
  group_by(kamp_id) %>%
  arrange(dist_hours_to_kickoff, .by_group = TRUE) %>%
  slice(1) %>%
  ungroup() %>%
  transmute(
    kamp_id,
    vejr_tid_h = vejr_time_h,
    vejrcode,
    vejrbeskrivelse,
    dist_hours_to_kickoff,
    vejr_match_type = "grid_nearest"
  )

mangler_vejr <- baseline_key %>%
  anti_join(kandidater_vejr %>% select(kamp_id), by = "kamp_id")

fallback_vejr <- mangler_vejr %>%
  left_join(vejr_fallback_1 %>% select(vejr_dato, vejr_time_h, vejrcode, vejrbeskrivelse),
    by = c("kamp_dato" = "vejr_dato")) %>%
  transmute(
    kamp_id,
    vejr_tid_h = vejr_time_h,
    vejrcode,
    vejrbeskrivelse,
    dist_hours_to_kickoff = NA_integer_,
    vejr_match_type = "fallback_dato"
  )

vejr_match <- bind_rows(kandidater_vejr, fallback_vejr)
stopifnot(sum(duplicated(vejr_match$kamp_id)) == 0)

```

```
baseline <- baseline %>%
  left_join(vejrr_match, by = "kamp_id") %>%
  mutate(
    vejrr_mangler_obs = if_else(is.na(vejrrkode), 1L, 0L),
    vejrr_ikke_observerbar = if_else(!is.na(vejrrkode) & vejrrkode == 0, 1L, 0L),
    vejrr_mangler_info = if_else(vejrr_mangler_obs == 1L | vejrr_ikke_observerbar == 1L, 1L, 0L),
    vejrrkode_model = if_else(vejrr_mangler_info == 1L, -1L, as.integer(vejrrkode)),
    vejrrbeskrivelse_model = case_when(
      vejrr_mangler_obs == 1L ~ "UKENDT",
      vejrr_ikke_observerbar == 1L ~ "IKKE_OBSERVERBAR",
      TRUE ~ as.character(vejrrbeskrivelse)
    )
  )

cat("Vejrr joinet    NA vejrrkode:", sum(is.na(baseline$vejrrkode)), "\n\n")
```

Vejrr joinet NA vejrrkode: 38

### 3.6.5 Temperatur før kickoff (step-back samme dag)

Denne kode henter temperaturmålinger og knytter dem til hver kamp ved at vælge den seneste måling samme dag før kickoff. Hvis der ikke findes en måling før kickoff, markeres det med en match-type og NA-værdier. Til sidst joines temperaturen ind i baseline, og der laves model-klare kolonner.

```
# 4E) Temperatur (step-back samme dag: 18→15→12→09 før kickoff)
cat("\n--- FEATURE: TEMPERATUR ---\n")
```

--- FEATURE: TEMPERATUR ---

```
temp_sql <- safe_dbReadTable(con, "PBA03_JoinReady", "fact_temperatur_join_ready")
stopifnot(all(c("obs_dato", "tid", "temperatur") %in% names(temp_sql)))

temp_tider_h <- c(9L, 12L, 15L, 18L)

temp_grid <- temp_sql %>%
  mutate(
    temp_dato = if (inherits(obs_dato, "Date")) obs_dato else as.Date(obs_dato, format = "%d-%m-%Y"),
    temp_tid_chr = str_trim(as.character(tid)),
    temp_tid_chr = if_else(str_detect(temp_tid_chr, "^\\d{1,2}:\\d{2}$"), paste0(temp_tid_chr, ":00"),
    temp_time_h = time_chr_to_hour(temp_tid_chr),
    temperatur = suppressWarnings(as.numeric(temperatur))
  ) %>%
  filter(!is.na(temp_dato), !is.na(temp_time_h), temp_time_h %in% temp_tider_h) %>%
  group_by(temp_dato, temp_time_h) %>%
  slice(1) %>%
  ungroup()

temp_kandidater <- baseline_key %>%
```

```

inner_join(temp_grid %>% select(temp_dato, temp_time_h, temperatur),
           by = c("kamp_dato" = "temp_dato")) %>%
filter(!is.na(kamp_time_h), temp_time_h <= kamp_time_h) %>%
mutate(temp_dist_hours_to_kickoff = kamp_time_h - temp_time_h)

temp_valgt <- temp_kandidater %>%
  group_by(kamp_id) %>%
  arrange(desc(temp_time_h), .by_group = TRUE) %>%
  slice(1) %>%
  ungroup() %>%
  transmute(
    kamp_id,
    temp_tid_h = temp_time_h,
    temperatur,
    temp_dist_hours_to_kickoff = as.integer(temp_dist_hours_to_kickoff),
    temp_match_type = "step_back_same_day"
  )

temp_mangler <- baseline_key %>%
  anti_join(temp_valgt %>% select(kamp_id), by = "kamp_id") %>%
  transmute(
    kamp_id,
    temp_tid_h = NA_integer_,
    temperatur = NA_real_,
    temp_dist_hours_to_kickoff = NA_integer_,
    temp_match_type = case_when(
      is.na(kamp_time_h) ~ "NO_KICKOFF_TIME",
      TRUE ~ "NO_TEMP_BEFORE_KICKOFF"
    )
  )

temp_match <- bind_rows(temp_valgt, temp_mangler)
stopifnot(sum(duplicated(temp_match$kamp_id)) == 0)

baseline <- baseline %>%
  left_join(temp_match, by = "kamp_id") %>%
  mutate(
    temp_mangler_obs = if_else(is.na(temperatur), 1L, 0L),
    temperatur_model = as.numeric(temperatur),
    temp_match_type_model = if_else(is.na(temp_match_type), "NO_TEMP", as.character(temp_match_type))
  )

cat("Temperatur joinet    NA temperatur:", sum(is.na(baseline$temperatur)), "\n\n")

```

```
Temperatur joinet    NA temperatur: 1
```

### 3.6.6 Placering før kamp (feature til baseline)

Denne del af pipeline henter holdets placering før kamp fra en rensset tabel og gør den klar til at blive joinet ind i baseline. Den reducerer data til én række pr. kamp\_id, så joinet ikke kan skabe dubletter i dit modelgrundlag. Til sidst udskrives en statuslinje, der viser hvor mange kampe der ender med manglende placering (NA).

```
# 4F) Placering
plac_raw <- safe_dbReadTable(con, "PBA02_Clean", "fact_vff_rundeplaceringer_clean") %>%
  mutate(kamp_id = as.character(kamp_id))
stopifnot(all(c("kamp_id", "placering_før_kamp") %in% names(plac_raw)))

plac_1row <- plac_raw %>%
  select(kamp_id, placering_før_kamp) %>%
  mutate(placering_før_kamp = suppressWarnings(as.integer(placering_før_kamp))) %>%
  arrange(kamp_id) %>%
  group_by(kamp_id) %>%
  slice(1) %>%
  ungroup()

stopifnot(sum(duplicated(plac_1row$kamp_id)) == 0)
baseline <- baseline %>% left_join(plac_1row, by = "kamp_id")
cat("Placering jointet    NA placering_før_kamp:", sum(is.na(baseline$placering_før_kamp)), "\n")
```

```
Placering jointet    NA placering_før_kamp: 5
```

### 3.6.7 Modstander (feature til baseline)

Denne kode henter kamp-programdata og finder automatisk den kolonne, der indeholder modstanderens navn, selv hvis kolonnenavnet varierer. Modstander-navnet renses (fjern overflødige mellemrum og tomme værdier sættes til NA), og der sikres én række pr. kamp\_id før join. Til sidst joines modstander ind i baseline, konverteres til faktor, og der logges både NA'er og antal unikke modstandere.

```
# 4G) Modstander
mod_raw <- safe_dbReadTable(con, "PBA03_JoinReady", "fact_superliga_program_join_ready") %>%
  mutate(kamp_id = as.character(kamp_id))
stopifnot("kamp_id" %in% names(mod_raw))

candidate_cols <- c("modstander", "udehold", "away_team", "modstander_navn", "udehold_navn", "away_team_navn")
found <- candidate_cols[candidate_cols %in% names(mod_raw)][1]
if (is.na(found)) {
  cat(" Kunne ikke finde en modstander-kolonne automatisk.\nKolonner:\n")
  print(names(mod_raw))
  stop("Ret candidate_cols eller vælg korrekt kolonne.")
}

mod_1row <- mod_raw %>%
  select(kamp_id, modstander_raw = all_of(found)) %>%
  mutate(
    modstander = str_squish(as.character(modstander_raw)),
    modstander = if_else(is.na(modstander) | modstander == "", NA_character_, modstander)
  ) %>%
  select(kamp_id, modstander) %>%
  arrange(kamp_id) %>%
  group_by(kamp_id) %>%
  slice(1) %>%
  ungroup()
```

```
stopifnot(sum(duplicated(mod_1row$kamp_id)) == 0)

baseline <- baseline %>%
  left_join(mod_1row, by = "kamp_id") %>%
  mutate(modstander = as.factor(modstander))

cat("Modstander joinet    NA modstander:", sum(is.na(baseline$modstander)), "\n")
```

```
Modstander joinet    NA modstander: 0
```

```
cat("Antal unikke modstandere:", n_distinct(as.character(baseline$modstander)), "\n\n")
```

```
Antal unikke modstandere: 22
```

### 3.6.8 Samlet QA af baseline (tjek af manglende værdier og entydighed)

Denne blok laver en hurtig kvalitetssikring af dit feature-sæt ved at tælle rækker, unikke kamp-id'er og antal manglende værdier pr. nøglefeature. Det hjælper dig med at opdage datalæk, dubletter eller kolonner der ikke blev joinet korrekt, før du går videre til modellering. Til sidst "låser" du datasættet som baseline\_d3 og rydder op ved at fjerne baseline fra miljøet.

```
# 4H) Samlet QA
qa_d3 <- baseline %>%
  summarise(
    rækker_total = n(),
    unikke_kamp_id = n_distinct(kamp_id),
    uden_kamp_time_h = sum(is.na(kamp_time_h)),
    uden_helligdag = sum(is.na(er_helligdag)),
    uden_sah_flag = sum(is.na(er_håndboldkamp_SAH)),
    uden_befolkning = sum(is.na(befolkningstal)),
    uden_vejrkode = sum(is.na(vejrkode)),
    uden_temp = sum(is.na(temperatur)),
    uden_placering = sum(is.na(placering_før_kamp)),
    uden_modstander = sum(is.na(modstander))
  )

print(qa_d3)
```

```
# A tibble: 1 x 10
  rækker_total unikke_kamp_id uden_kamp_time_h uden_helligdag uden_sah_flag
      <int>         <int>         <int>         <int>         <int>
1         254         254             0             0             0
# i 5 more variables: uden_befolkning <int>, uden_vejrkode <int>,
#   uden_temp <int>, uden_placering <int>, uden_modstander <int>
```

```
cat("\nbaseline_d3 (features) klar \n\n")
```

baseline\_d3 (features) klar

```
baseline_d3 <- baseline
rm(baseline)
```

### 3.6.9 Billetsalg (D3/D7/D10): robust udtræk og “1 række pr. kamp”

Denne kode henter billetsalgsdata og finder automatisk de korrekte kolonner til D3, D7 og D10, selv hvis kolonnenavnene varierer. Derefter reduceres data til én række pr. kamp\_id ved at tage den højeste observerede værdi pr. kamp (praktisk hvis der ligger flere snapshots). Til sidst renses output, så umulige værdier som Inf bliver til NA, hvilket gør datasættet stabilt til analyse og modellering.

```
# =====
# 5) Billetsalg -> analysis_df_d3 + df_d3_all (ren, stabil)
# =====
tickets_raw <- safe_dbReadTable(con, "PBA03_JoinReady", "fact_VFF_Billetsalg_join_ready") %>%
  mutate(kamp_id = as.character(kamp_id))

col_d3 <- find_first_col(tickets_raw, c("d3_tilskuere","d3","tilskuere_d3","billetter_d3"))
col_d7 <- find_first_col(tickets_raw, c("d7_tilskuere","d7","tilskuere_d7","billetter_d7"))
col_d10 <- find_first_col(tickets_raw, c("d10_tilskuere","d10","tilskuere_d10","billetter_d10"))
if (is.na(col_d3)) stop(" Kan ikke finde D3-kolonne i billetsalg-tabellen. Kig på names(tickets_raw).")

tickets_1row <- tickets_raw %>%
  group_by(kamp_id) %>%
  summarise(
    d3 = suppressWarnings(max(as.numeric(.data[[col_d3]]), na.rm = TRUE)),
    d7 = if (!is.na(col_d7)) suppressWarnings(max(as.numeric(.data[[col_d7]]), na.rm = TRUE)) else
    d10 = if (!is.na(col_d10)) suppressWarnings(max(as.numeric(.data[[col_d10]]), na.rm = TRUE)) else
    .groups = "drop"
  ) %>%
  mutate(
    d3 = if_else(is.infinite(d3), NA_real_, d3),
    d7 = if_else(is.infinite(d7), NA_real_, d7),
    d10 = if_else(is.infinite(d10), NA_real_, d10)
  )
```

### 3.6.10 Step-vækster + model-datasæt (D3): features, join og endelig klargøring

Denne kode laver billetsalgs-features baseret på step-vækster mellem D10→D7 og D7→D3, hvor negative ændringer klippes til 0. Herefter joines billetsalgs-features ind i dit feature-sæt (baseline\_d3) og der bygges et stabilt “fælles” datasæt (df\_d3\_all) til modellering. Til sidst sikres korrekte datatyper, fjernes rækker uden kampdato, og datasættet sorteres kronologisk.

```
# -----
# RETTELSE: step-vækster
# - vækst_d10_d7 = max(d7 - d10, 0)
# - vækst_d7_d3 = max(d3 - d7, 0)
# - Ingen vækst_d10_d3
```

```
# -----
tickets_feats_d3 <- tickets_1row %>%
  mutate(
    billetter_d3 = d3,
    vækst_d10_d7 = if_else(!is.na(d10) & !is.na(d7), pmax(d7 - d10, 0), NA_real_),
    vækst_d7_d3 = if_else(!is.na(d7), pmax(d3 - d7, 0), NA_real_)
  ) %>%
  select(kamp_id, billetter_d3, vækst_d10_d7, vækst_d7_d3)

analysis_df_d3 <- baseline_d3 %>%
  mutate(kamp_id = as.character(kamp_id)) %>%
  left_join(tickets_feats_d3, by = "kamp_id")

cat("analysis_df_d3 klar   Rækker:", nrow(analysis_df_d3),
    "Unikke kamp_id:", n_distinct(analysis_df_d3$kamp_id), "\n")
```

```
analysis_df_d3 klar   Rækker: 254 Unikke kamp_id: 254
```

```
cat("NA billetter_d3:", sum(is.na(analysis_df_d3$billetter_d3)), "\n\n")
```

```
NA billetter_d3: 0
```

```
# df_d3_all: fælles datasæt til modeller
df_d3_all <- analysis_df_d3 %>%
  transmute(
    kamp_id      = as.character(kamp_id),
    kamp_dato    = as.Date(kamp_dato),
    tilskuere    = as.numeric(tilskuere),

    sæson        = as.factor(sæson),
    runde        = as.integer(runde),
    kamp_time_h  = as.integer(kamp_time_h),

    billetter_d3 = as.numeric(billetter_d3),
    vækst_d10_d7 = as.numeric(vækst_d10_d7),
    vækst_d7_d3  = as.numeric(vækst_d7_d3),

    er_helligdag = as.integer(er_helligdag),
    er_håndboldkamp_SAH = as.integer(er_håndboldkamp_SAH),
    antal_håndboldkampe = as.integer(antal_håndboldkampe),

    temperatur_model = as.numeric(temperatur_model),
    vejrkode_model   = as.integer(vejrkode_model),
    vejrmangler_info = as.integer(vejrmangler_info),

    placering_før_kamp = as.integer(placering_før_kamp),
    befolkningstal     = as.numeric(befolkningstal),

    modstander = as.factor(modstander)
  ) %>%
```

```
filter(!is.na(kamp_dato)) %>%
arrange(kamp_dato)
View(df_d3_all)
```

### 3.7 Tidssplit: kronologisk 80/20 opdeling i træning og test

Her opdeles det samlede analysedatasæt i et træningssæt og et testsæt baseret på tid (kronologisk), så evalueringen efterligner en realistisk prognosesituation. Først fastsættes test-andelen (20%), hvorefter der beregnes et cut-point, og data splittes i de tidligste observationer (train) og de seneste (test). Til sidst udskrives en kort status med størrelserne samt startdatoen for testperioden, så du tydeligt kan dokumentere split'et i rapporten.

```
# =====
# Tidssplit (80/20) kronologisk - ÉN gang
# =====
test_prop <- 0.20
n_total <- nrow(df_d3_all)
n_test  <- max(1, floor(n_total * test_prop))
cut_idx <- n_total - n_test

train_d3 <- df_d3_all[1:cut_idx, , drop = FALSE]
test_d3  <- df_d3_all[(cut_idx + 1):n_total, , drop = FALSE]

cat("Tidssplit klar   Train:", nrow(train_d3), " Test:", nrow(test_d3), "\n")
```

```
Tidssplit klar   Train: 204  Test: 50
```

```
cat("Test-periode starter:", as.character(min(test_d3$kamp_dato)), "\n\n")
```

```
Test-periode starter: 2022-11-06
```

### 3.8 Predictor pool: dynamisk valg af forklaringsvariabler

Denne kode forbereder listen af forklaringsvariable, der skal indgå i modelleringen. Først tjekkes om variabelen modstander findes i træningsdatasættet, så scriptet ikke fejler, hvis kolonnen mangler. Derefter samles en “predictor pool”, hvor modstander kun inkluderes, når den faktisk er tilgængelig, og hvor befolkningstal bevidst udelades som standard (så det kan testes separat senere).

```
#bruges i nedenstående og fungere som et tjek
has_modstander <- "modstander" %in% names(train_d3)

# predictor pool (uden befolkning som default) sikring af modstander er i predictor_pool
predictor_pool <- c(
  "sæson","runde","kamp_time_h",
  "billetter_d3","vækst_d10_d7","vækst_d7_d3",
  if (has_modstander) "modstander" else NULL,
  "er_helligdag","er_håndboldkamp_SAH","antal_håndboldkampe",
  "temperatur_model","vejrkode_model","vejr_mangler_info",
  "placering_før_kamp"
)
```

Her sikrer, at din liste af forklaringsvariable faktisk kan bruges til modellering på træningsdata. Først fjernes variable, der ikke findes i train\_d3, og derefter fjernes variable, der er helt tomme (kun NA) i træningssættet. Til sidst bygges en lineær model-formel dynamisk, så du automatisk får tilskuere ~ x1 + x2 + ... baseret på den endelige predictor-liste.

```
# Fjern predictors som ikke findes i træningsdatasættet (robusthed)
predictor_pool <- predictor_pool[predictor_pool %in% names(train_d3)]
# Fjern predictors der kun består af NA i træningsdatasættet (ingen information)
predictor_pool <- predictor_pool[
  vapply(
    predictor_pool,
    function(v) !all(is.na(train_d3[[v]])),
    logical(1)
  )
]

# Stop hvis der ikke er nogen brugbare predictors tilbage (ellers kan modellen ikke estimeres)
if (length(predictor_pool) == 0) stop(" predictor_pool endte tom. Tjek df_d3_all kolonner/NA.")
```

Denne linje bygger en model-formel automatisk ud fra predictor\_pool, så du ikke skal skrive variablerne manuelt. Resultatet er en formula-variabel (full\_formula), som kan bruges direkte i lm() og step() til at fitte modeller med præcis de predictors, du har valgt (også hvis listen ændrer sig undervejs).

```
# Byg model-formel dynamisk til lm/stepwise:
# fx "tilskuere ~ runde + kamp_time_h + billetter_d3 + ..."
full_formula <- as.formula(paste("tilskuere ~", paste(predictor_pool, collapse = " + ")))
```

Faktor-niveauer i testdata tilpasses træningsdata, så ukendte kategorier samles i en fælles reference og modellen kan anvendes stabilt uden at fejle ved prediction.

```
# # robusthed: test modstander-levels til train levels (predict skal ikke knække)
# if ("modstander" %in%
#   names(train_step)) { tmp <- safe_factor_with_other(train_step$modstander, test_step$modstander,
#   train_step$modstander <- tmp$train
#   test_step$modstander <- tmp$test
#   rm(tmp)
# }

# # robusthed: test modstander-levels til train levels (predict skal ikke knække)
# if ("modstander" %in% names(train_step)) {
#   tmp <- safe_factor_with_other(train_step$modstander, test_step$modstander, "ANDRE")
#   train_step$modstander <- tmp$train
#   test_step$modstander <- tmp$test
#   rm(tmp)
# }

# 7B) complete cases til stepwise (samme logik som du gjorde)
train_step <- train_d3 %>% select(tilskuere, all_of(predictor_pool)) %>% drop_na()
test_step <- test_d3 %>% select(tilskuere, all_of(predictor_pool)) %>% drop_na()
```

```
# robusthed: test modstander-levels til train levels (predict skal ikke knække)
if ("modstander" %in% names(train_step)) {
  tmp <- safe_factor_with_other(train_step$modstander, test_step$modstander, "ANDRE")
  train_step$modstander <- tmp$train
  test_step$modstander <- tmp$test
  rm(tmp)
}
```

Efter stepwise-udvælgelse udskrives antallet af observationer i trænings- og testdatasættet. Samtidig dokumenteres hvor mange rækker der er fjernet som følge af manglende værdier, så datatab og konsistens i modelgrundlaget kan vurderes eksplicit.

```
cat("Stepwise datasæt   Train:", nrow(train_step), " Test:", nrow(test_step), "\n")
```

```
Stepwise datasæt   Train: 201  Test: 47
```

```
cat("Drop NA (train):", nrow(train_d3) - nrow(train_step), "rækker\n")
```

```
Drop NA (train): 3 rækker
```

```
cat("Drop NA (test) :", nrow(test_d3) - nrow(test_step), "rækker\n\n")
```

```
Drop NA (test) : 3 rækker
```

Der opstilles en nulmodel med kun konstantled som reference samt en fuld model med alle udvalgte forklarende variable, så den efterfølgende modeludvælgelse kan vurderes relativt til et simpelt baseline-niveau.

```
m_null <- lm(tilskuere ~ 1, data = train_step)
m_full <- lm(full_formula, data = train_step)
```

Der anvendes forward, backward og both stepwise selection inden for samme model space for at sammenligne, hvordan forskellige selektionsstrategier påvirker den endelige modelsammensætning.

```
m_step_fwd  <- step(m_null, scope = list(lower = formula(m_null), upper = formula(m_full)), direction
m_step_bwd  <- step(m_full, direction = "backward", trace = 0)
m_step_both <- step(m_null, scope = list(lower = formula(m_null), upper = formula(m_full)), direction
```

Model performance samles i en fælles struktur med RMSE, MAE og  $R^2$ , så den prædiktive nøjagtighed og forklaringsgrad kan sammenlignes direkte på tværs af stepwise forward, backward og both selection baseret på AIC.

```
pred_step_fwd  <- clip_0_cap(predict(m_step_fwd, newdata = test_step))
pred_step_bwd  <- clip_0_cap(predict(m_step_bwd, newdata = test_step))
pred_step_both <- clip_0_cap(predict(m_step_both, newdata = test_step))

perf_step <- tibble(
  model = c("Stepwise Forward (AIC)", "Stepwise Backward (AIC)", "Stepwise Both (AIC)"),
```

```

RMSE = c(rmse_vec(test_step$tilskuere, pred_step_fwd),
         rmse_vec(test_step$tilskuere, pred_step_bwd),
         rmse_vec(test_step$tilskuere, pred_step_both)),
MAE   = c(mae_vec(test_step$tilskuere, pred_step_fwd),
         mae_vec(test_step$tilskuere, pred_step_bwd),
         mae_vec(test_step$tilskuere, pred_step_both)),
R2     = c(r2_vec(test_step$tilskuere, pred_step_fwd),
         r2_vec(test_step$tilskuere, pred_step_bwd),
         r2_vec(test_step$tilskuere, pred_step_both))
)

cat("Stepwise færdig \n")

```

Stepwise færdig

```
print(perf_step)
```

```

# A tibble: 3 x 4
  model          RMSE  MAE  R2
  <chr>      <dbl> <dbl> <dbl>
1 Stepwise Forward (AIC)  214.  177. 0.965
2 Stepwise Backward (AIC) 214.  177. 0.965
3 Stepwise Both (AIC)    214.  177. 0.965

```

```
cat("\nValgte features (Backward):\n"); print(names(coef(m_step_bwd))); cat("\n")
```

Valgte features (Backward):

```

[1] "(Intercept)"      "billetter_d3"      "vækst_d10_d7"
[4] "vækst_d7_d3"      "temperatur_model"  "placering_før_kamp"

```

Resultaterne viser, at stepwise forward, backward og both (AIC) leverer identisk modelperformance, med RMSE på 214, MAE på 177 og  $R^2$  på 0,97. Dette indikerer, at de tre selektionsstrategier konvergerer mod samme endelige model og variabelsæt. Valg af stepwise-retning har derfor ingen praktisk betydning for modellens prædiktionssevne i dette setup.

Stepwise backward (AIC) udvælger en enkel model, hvor billetsalg tæt på kampdatoen udgør den primære forklaringsfaktor, suppleret af kortsigtet salgsdynamik, temperatur og placering før kamp. Dette indikerer, at kampnære efterspørgselsdata er afgørende for modellens prædiktionssevne.

Den estimerede null-model indeholder udelukkende et konstantled og fungerer som reference for modeludvidelse, mens den fulde model inkluderer samtlige potentielle forklaringsvariable. Disse to modeller danner yderpunkterne for den efterfølgende stepwise-selektion baseret på AIC.

```

m_null <- lm(tilskuere ~ 1, data = train_step)
m_full <- lm(full_formula, data = train_step)

```

Stepwise forward estimeres ved gradvist at udvide null-modellen mod den fulde model, mens stepwise backward reducerer den fulde model ved successiv fjernelse af variable. Stepwise both kombinerer de to strategier ved både at tilføje og fjerne variable undervejs, hvor alle tre metoder optimerer modelvalget ud fra AIC-kriteriet.

```
m_step_fwd <- step(m_null, scope = list(lower = formula(m_null), upper = formula(m_full)), direction = "forward", trace = 0)
m_step_bwd <- step(m_full, direction = "backward", trace = 0)
m_step_both <- step(m_null, scope = list(lower = formula(m_null), upper = formula(m_full)), direction = "both", trace = 0)

pred_step_fwd <- clip_0_cap(predict(m_step_fwd, newdata = test_step))
pred_step_bwd <- clip_0_cap(predict(m_step_bwd, newdata = test_step))
pred_step_both <- clip_0_cap(predict(m_step_both, newdata = test_step))
```

Modellernes forudsigelser beregnes på testdatasættet og efterbehandles ved at afgrænse værdierne til et realistisk interval. Dette sikrer, at negative eller urealistisk høje tilskuertal ikke indgår i den efterfølgende performanceevaluering

```
perf_step <- tibble(
  model = c("Stepwise Forward (AIC)", "Stepwise Backward (AIC)", "Stepwise Both (AIC)"),
  RMSE = c(rmse_vec(test_step$tilskuere, pred_step_fwd),
           rmse_vec(test_step$tilskuere, pred_step_bwd),
           rmse_vec(test_step$tilskuere, pred_step_both)),
  MAE = c(mae_vec(test_step$tilskuere, pred_step_fwd),
           mae_vec(test_step$tilskuere, pred_step_bwd),
           mae_vec(test_step$tilskuere, pred_step_both)),
  R2 = c(r2_vec(test_step$tilskuere, pred_step_fwd),
          r2_vec(test_step$tilskuere, pred_step_bwd),
          r2_vec(test_step$tilskuere, pred_step_both))
)
perf_step
```

```
# A tibble: 3 x 4
  model          RMSE  MAE  R2
  <chr>        <dbl> <dbl> <dbl>
1 Stepwise Forward (AIC)  214.  177. 0.965
2 Stepwise Backward (AIC) 214.  177. 0.965
3 Stepwise Both (AIC)    214.  177. 0.965
```

Modelperformance evalueres på testdatasættet ved hjælp af RMSE, MAE og  $R^2$  for hver stepwise-variant. Resultaterne viser identisk performance på tværs af forward, backward og both, hvilket indikerer, at metoderne konvergerer mod samme endelige model og prædiktionssevne.

```
cat("Stepwise færdig \n")
```

Stepwise færdig

```
print(perf_step)
```

```
# A tibble: 3 x 4
  model          RMSE  MAE    R2
  <chr>          <dbl> <dbl> <dbl>
1 Stepwise Forward (AIC)  214.  177. 0.965
2 Stepwise Backward (AIC) 214.  177. 0.965
3 Stepwise Both (AIC)    214.  177. 0.965
```

```
cat("\nValgte features (Backward):\n"); print(names(coef(m_step_bwd))); cat("\n")
```

Valgte features (Backward):

```
[1] "(Intercept)"      "billetter_d3"      "vækst_d10_d7"
[4] "vækst_d7_d3"      "temperatur_model"  "placering_før_kamp"
```

### 3.9 Best Subset

Variabelpuljen afgrænses ved at udelukke højdimensionelle faktorer for at reducere kompleksitet og beregningsbyrde. Hvis dette medfører en tom pulje, genindsættes det fulde variabelsæt for at sikre estimerbarhed.

```
subset_exclude_factors <- TRUE
predictor_pool_subset <- predictor_pool
if (subset_exclude_factors) predictor_pool_subset <- setdiff(predictor_pool_subset, c("sæson", "modstan
if (length(predictor_pool_subset) == 0) predictor_pool_subset <- predictor_pool
```

Der konstrueres en modelmatrix ud fra det reducerede variabelsæt, hvorefter trænings- og testdata opdeles konsistent. Variable med nul-varians fjernes, og testmatricen tilpasses træningsmatricens struktur for at undgå dimensionskonflikter.

```
mm_formula <- as.formula(paste("~", paste(predictor_pool_subset, collapse = " + ")))
mm_list <- make_mm_train_test(train_step, test_step, mm_formula)
X_train <- drop_zero_variance_cols(mm_list$X_train)
X_test <- mm_list$X_test[, colnames(X_train), drop = FALSE]
```

Den afhængige variabel udtrækkes separat for trænings- og testdatasættet for at sikre korrekt estimering og efterfølgende performanceevaluering.

```
y_train <- train_step$tilskuere
y_test <- test_step$tilskuere
```

Det maksimale antal variable i best subset-selektionen begrænses for at sikre beregningsmæssig kontrol og reducere risikoen for overfitting. Denne afgrænsning balancerer modelkompleksitet og generalisering.

```
nvmax <- min(10L, ncol(X_train))
cat("Best subset setup    ncol(X_train):", ncol(X_train), " nvmax:", nvmax, "\n\n")
```

```
Best subset setup    ncol(X_train): 10  nvmax: 10
```

Hvis træningsmatricen ikke indeholder forklaringsvariable, anvendes en intercept-only referencemodel baseret på gennemsnittet af træningsdata. Ellers estimeres en best subset-model via sekventiel erstatning, hvor det optimale antal variable vælges ud fra BIC. Afhængigt af det valgte variabelsæt estimeres en lineær model, og performance evalueres på testdatasættet ved hjælp af RMSE, MAE og  $R^2$ .

```
if (ncol(X_train) == 0) {
  pred_subset <- clip_0_cap(rep(mean(y_train, na.rm = TRUE), length(y_test)))
  perf_subset <- tibble(
    model = "Best Subset (Intercept-only)",
    RMSE = rmse_vec(y_test, pred_subset),
    MAE = mae_vec(y_test, pred_subset),
    R2 = r2_vec(y_test, pred_subset),
    k_selected = 0L
  )
  regfit <- NULL; sel_cols <- character(0); fit_subset <- NULL
} else {
  regfit <- regsubsets(x = X_train, y = y_train, nvmax = nvmax, method = "seqrep")
  regsum <- summary(regfit)
  best_k_bic <- which.min(regsum$bic)
  which_mat <- regsum$which
  sel_cols <- names(which_mat[best_k_bic, ])[which_mat[best_k_bic, ]]
  sel_cols <- setdiff(sel_cols, "(Intercept)")

  cat("Best subset valgt (BIC)    k =", best_k_bic, "\nValgte dummy-features:\n")
  print(sel_cols); cat("\n")

  if (length(sel_cols) == 0) {
    pred_subset <- clip_0_cap(rep(mean(y_train, na.rm = TRUE), length(y_test)))
    perf_subset <- tibble(
      model = "Best Subset (BIC → Intercept-only)",
      RMSE = rmse_vec(y_test, pred_subset),
      MAE = mae_vec(y_test, pred_subset),
      R2 = r2_vec(y_test, pred_subset),
      k_selected = 0L
    )
    fit_subset <- NULL
  } else {
    Xtr_sel <- X_train[, sel_cols, drop = FALSE]
    Xte_sel <- X_test[, sel_cols, drop = FALSE]
    Xtr_sel_i <- cbind(`(Intercept)` = 1, Xtr_sel)
    Xte_sel_i <- cbind(`(Intercept)` = 1, Xte_sel)

    fit_subset <- lm.fit(x = Xtr_sel_i, y = y_train)
    pred_subset <- clip_0_cap(as.numeric(Xte_sel_i %*% fit_subset$coefficients))

    perf_subset <- tibble(
      model = "Best Subset (BIC, regsubsets, seqrep)",
      RMSE = rmse_vec(y_test, pred_subset),
      MAE = mae_vec(y_test, pred_subset),
      R2 = r2_vec(y_test, pred_subset),
      k_selected = length(sel_cols)
    )
  }
}
```

```
}
}
```

```
Best subset valgt (BIC)    k = 3
Valgte dummy-features:
[1] "billetter_d3" "vækst_d10_d7" "vækst_d7_d3"
```

```
cat("Best subset færdig \n")
```

```
Best subset færdig
```

```
print(perf_subset); cat("\n")
```

```
# A tibble: 1 x 5
  model          RMSE    MAE    R2 k_selected
  <chr>        <dbl> <dbl> <dbl>      <int>
1 Best Subset (BIC, regsubsets, seqrep)  213.  176. 0.966          3
```

Best subset-selektionen baseret på BIC identificerer en parsimonisk model med tre forklaringsvariable: billetsalg tre dage før kamp samt to kortsigtede vækstindikatorer. Modellen opnår høj prædiktionssevne med RMSE på 213, MAE på 176 og  $R^2$  på 0,966, hvilket indikerer, at kampnær efterspørgselsdynamik forklarer hovedparten af variationen i tilskuertallet. Resultatet viser, at en meget enkel model kan matche performance fra mere komplekse specifikationer.

### 3.9.1 Modelperformance for D3-modeller

Dette afsnit sammenfatter testperformance for de estimerede D3-modeller med henblik på at sammenligne prædiktionssevne på tværs af forskellige modeludvælgelsesstrategier. Modellerne evalueres konsekvent på samme testdatasæt og sammenlignes ved hjælp af RMSE, MAE og  $R^2$ .

```
perf_all <- bind_rows(
  perf_step,
  perf_subset %>% select(model, RMSE, MAE, R2)
) %>% arrange(RMSE)

cat("=====\n")
```

```
=====
```

```
cat("D3 - SAMLET TEST PERFORMANCE (SORTERET PÅ RMSE)\n")
```

```
D3 - SAMLET TEST PERFORMANCE (SORTERET PÅ RMSE)
```

```
cat("=====\n")
```

```
=====
```

```
print(perf_all); cat("\n")
```

```
# A tibble: 4 x 4
  model          RMSE  MAE  R2
  <chr>      <dbl> <dbl> <dbl>
1 Best Subset (BIC, regsubsets, seqrep) 213.  176. 0.966
2 Stepwise Forward (AIC)                214.  177. 0.965
3 Stepwise Both (AIC)                   214.  177. 0.965
4 Stepwise Backward (AIC)                214.  177. 0.965
```

Resultaterne viser, at best subset-modellen baseret på BIC opnår den laveste RMSE og MAE samt den højeste forklaringsgrad, om end forskellene til stepwise-modellerne er marginale. Alle stepwise-varianter (forward, backward og both) leverer identisk performance, hvilket indikerer konvergens mod samme effektive modelstruktur. Samlet peger resultaterne på, at en meget enkel model kan opnå samme prædiktionssevne som mere komplekse.

Resultater fra modeludvælgelsen for D3 er struktureret og gemt i et samlet objekt, der indeholder datagrundlag, anvendte modelspecifikationer, estimerede modeller samt tilhørende performancemål. Denne struktur sikrer fuld reproducerbarhed og gør det muligt systematisk at sammenligne modeller og genanvende resultater i den videre analyse.

```
d3_selection_results <- list(
  data = list(
    analysis_df_d3 = analysis_df_d3,
    df_d3_all = df_d3_all,
    train_step = train_step,
    test_step = test_step
  ),
  formulas = list(full_formula = full_formula, mm_formula = mm_formula),
  models = list(
    step_forward = m_step_fwd,
    step_backward = m_step_bwd,
    step_both = m_step_both,
    best_subset = list(
      regsubsets = regfit,
      selected_cols = sel_cols,
      lmfit = fit_subset,
      subset_exclude_factors = subset_exclude_factors
    )
  ),
  performance = list(table = perf_all, step = perf_step, subset = perf_subset)
)

cat("D3 selection resultater gemt i: d3_selection_results \n\n")
```

D3 selection resultater gemt i: d3\_selection\_results

### 3.10 Regulariserede modeller (Ridge og Lasso)

I dette afsnit estimeres regulariserede regressionsmodeller (Ridge og Lasso) med henblik på at vurdere, om shrinkage og variabelselektion kan forbedre prædiktionssevnen i D3-opsætningen. Modellerne estimeres på samme train-test-split

og samme complete cases som de øvrige analyser for at sikre sammenlignelighed. Hyperparametre tunes udelukkende på træningsdata ved hjælp af tidsblok-baseret krydsvalidering, hvorefter performance evalueres én gang på test-datasættet.

Trænings- og testdatasættene kopieres til et separat datasæt til brug for de regulariserede modeller for at bevare konsistens med det oprindelige datasplit.

```
# For at undgå massiv dummy-eksplosion konverterer vi sæson -> sæson_start (numerisk)
train_glm <- train_step
test_glm  <- test_step
```

Den kategoriske sæsonvariabel transformeres til en numerisk sæsonstartindikator, hvorefter den oprindelige variabel fjernes. Dette reducerer dimensionalitet og sikrer en konsistent repræsentation i trænings- og testdatasættene.

```
if ("sæson" %in% names(train_glm)) {
  train_glm <- train_glm %>% mutate(sæson_start = season_start_year(sæson))
  test_glm  <- test_glm  %>% mutate(sæson_start = season_start_year(sæson))
  train_glm$sæson <- NULL
  test_glm$sæson  <- NULL
}
```

Predictor-poolen til glmnet konstrueres ved at erstatte den oprindelige sæsonvariabel med den numeriske sæsonstart og ved kun at inkludere variable, som faktisk er tilgængelige og informative i datasættet. Variable uden observationer udelukkes eksPLICIT for at sikre estimerbarhed af de regulariserede modeller.

```
predictor_pool_glm <- predictor_pool
if ("sæson" %in% predictor_pool_glm) predictor_pool_glm <- setdiff(predictor_pool_glm, "sæson")
if ("sæson_start" %in% names(train_glm)) predictor_pool_glm <- unique(c("sæson_start", predictor_pool_

predictor_pool_glm <- predictor_pool_glm[predictor_pool_glm %in% names(train_glm)]
predictor_pool_glm <- predictor_pool_glm[vapply(predictor_pool_glm, function(v) !all(is.na(train_glm[,
if (length(predictor_pool_glm) == 0) stop(" predictor_pool_glm endte tom. Kan ikke køre glmnet.")

tibble(predictor_pool_glm)
```

```
# A tibble: 14 x 1
  predictor_pool_glm
  <chr>
1 sæson_start
2 runde
3 kamp_time_h
4 billetter_d3
5 vækst_d10_d7
6 vækst_d7_d3
7 modstander
8 er_helligdag
9 er_håndboldkamp_SAH
10 antal_håndboldkampe
11 temperatur_model
12 vejrcode_model
13 vejr_mangler_info
14 placering_før_kamp
```

Der afgrænses til complete cases i både trænings- og testdata, da glmnet kræver fuldstændige observationer uden manglende værdier i hverken forklaringsvariable eller respons. Dette sikrer korrekt estimering og konsistent evaluering af de regulariserede modeller.

```
# complete cases igen (glmnet skal ikke have NA i X eller y)
keep_train <- complete.cases(train_glm %>% select(tilskuere, all_of(predictor_pool_glm)))

keep_test  <- complete.cases(test_glm  %>% select(tilskuere, all_of(predictor_pool_glm)))

train_glm_cc <- train_glm[keep_train, , drop = FALSE]
test_glm_cc  <- test_glm[keep_test,  , drop = FALSE]
```

Et intercept er modellens skæring med y-aksen, dvs. den forventede værdi af y, når alle forklaringsvariable er 0. I denne opsætning udelades et eksPLICIT intercept i modelmatricen, fordi glmnet selv estimerer skæringen med y-aksen internt.

Modelmatricen konstrueres uden eksPLICIT intercept (skæring med y-aksen), da glmnet estimerer dette internt. Modelmatricen bygges konsistent for trænings- og testdata for at sikre korrekt estimering og sammenlignelighed.

```
# model.matrix: ~ 0 + ... (glmnet håndterer intercept selv)
mm_rhs_terms <- paste(predictor_pool_glm, collapse = " + ")

mm_formula_glm <- as.formula(paste("~ 0 +", mm_rhs_terms))

mm_glm <- make_mm_train_test(
  train_df = train_glm_cc %>% select(all_of(predictor_pool_glm)),
  test_df  = test_glm_cc  %>% select(all_of(predictor_pool_glm)),
  mm_formula = mm_formula_glm
)
```

Modelmatricer og responsvariable klargøres i numerisk form i overensstemmelse med glmnets inputkrav, hvorefter datasættets dimensioner verificeres. Dette sikrer korrekt opsætning af trænings- og testdata før modelestimering.

```
X_train_glm <- as.matrix(mm_glm$X_train)
X_test_glm  <- as.matrix(mm_glm$X_test)

y_train_glm <- as.numeric(train_glm_cc$tilskuere)
y_test_glm  <- as.numeric(test_glm_cc$tilskuere)

cat("glmnet setup   ncol(X_train):", ncol(X_train_glm),
    "Train n:", length(y_train_glm), "Test n:", length(y_test_glm), "\n\n")
```

```
glmnet setup   ncol(X_train): 34 Train n: 201 Test n: 47
```

Krydsvalideringsfolds (CV folds) konstrueres som tidsblokke for at bevare den tidsmæssige struktur i træningsdata og undgå informationslækage mellem træning og validering.

```
foldid <- make_time_block_foldid(n = nrow(X_train_glm), k = 10)
```

Ridge- og Lasso-modeller estimeres ved hjælp af k-fold krydsvalidering med tidsblokke, hvor regulariseringsparameteren vælges på baggrund af træningsdata. En fast seed anvendes for at sikre reproducerbarhed af krydsvalideringen.

```
set.seed(42)
cv_ridge <- cv.glmnet(x = X_train_glm, y = y_train_glm, alpha = 0, standardize = TRUE, foldid = foldid)

set.seed(42)
cv_lasso <- cv.glmnet(x = X_train_glm, y = y_train_glm, alpha = 1, standardize = TRUE, foldid = foldid)
```

Forudsigelser beregnes på testdatasættet for både Ridge- og Lasso-modeller ved henholdsvis `__min` og `__1se`, hvorefter ekstreme værdier afgrænses til et realistisk interval. Modelperformance evalueres ved hjælp af RMSE, MAE,  $R^2$  og MSE, og resultaterne samles i en oversigtstabel sorteret efter RMSE for direkte sammenligning af prædiktionssevne.

```
pred_ridge_min <- clip_0_cap(as.numeric(predict(cv_ridge, newx = X_test_glm, s = "lambda.min")))
pred_ridge_1se <- clip_0_cap(as.numeric(predict(cv_ridge, newx = X_test_glm, s = "lambda.1se")))
pred_lasso_min <- clip_0_cap(as.numeric(predict(cv_lasso, newx = X_test_glm, s = "lambda.min")))
pred_lasso_1se <- clip_0_cap(as.numeric(predict(cv_lasso, newx = X_test_glm, s = "lambda.1se")))

perf_glmnet <- bind_rows(
  tibble(model = "Ridge (lambda.min)", RMSE = rmse_vec(y_test_glm, pred_ridge_min), MAE = mae_vec(y_test_glm, pred_ridge_min), R2 = r2_vec(y_test_glm, pred_ridge_min), MSE = mse_vec(y_test_glm, pred_ridge_min)),
  tibble(model = "Ridge (lambda.1se)", RMSE = rmse_vec(y_test_glm, pred_ridge_1se), MAE = mae_vec(y_test_glm, pred_ridge_1se), R2 = r2_vec(y_test_glm, pred_ridge_1se), MSE = mse_vec(y_test_glm, pred_ridge_1se)),
  tibble(model = "Lasso (lambda.min)", RMSE = rmse_vec(y_test_glm, pred_lasso_min), MAE = mae_vec(y_test_glm, pred_lasso_min), R2 = r2_vec(y_test_glm, pred_lasso_min), MSE = mse_vec(y_test_glm, pred_lasso_min)),
  tibble(model = "Lasso (lambda.1se)", RMSE = rmse_vec(y_test_glm, pred_lasso_1se), MAE = mae_vec(y_test_glm, pred_lasso_1se), R2 = r2_vec(y_test_glm, pred_lasso_1se), MSE = mse_vec(y_test_glm, pred_lasso_1se))
) %>% arrange(RMSE)

cat("Ridge/Lasso færdig \n")
```

Ridge/Lasso færdig

```
print(perf_glmnet); cat("\n")
```

```
# A tibble: 4 x 5
  model          RMSE    MAE    R2     MSE
  <chr>         <dbl> <dbl> <dbl>  <dbl>
1 Lasso (lambda.min)  219.  178. 0.964  47757.
2 Lasso (lambda.1se)  226.  183. 0.961  51162.
3 Ridge (lambda.min)  438.  337. 0.855 192256.
4 Ridge (lambda.1se)  461.  357. 0.839 212445.
```

Resultaterne viser, at Lasso-modellen med `__min` opnår den bedste prædiktionssevne blandt de regulariserede modeller, med RMSE på 219, MAE på 178 og  $R^2$  på 0,964. Dette indikerer, at en relativt begrænset mængde forklaringsvariable er tilstrækkelig til at forklare størstedelen af variationen i tilskuertallet, når irrelevante variable undertrykkes gennem regularisering.

Valget af `__1se` medfører, som forventet, en mere konservativ model med lavere kompleksitet, hvilket resulterer i en mindre forringelse af performance. Ridge-modellerne performer markant dårligere end Lasso, hvilket indikerer, at variabelselektion er afgørende i denne opsætning, og at shrinkage alene ikke er tilstrækkeligt til at håndtere multikollinearitet og støj i datagrundlaget.

Samlet understøtter resultaterne, at sparsomme modeller med eksplicit variabeludvælgelse er bedre egnet end fuldt regulariserede modeller i den kampnære D3-kontekst.

Efter evaluering af Ridge- og Lasso-modellerne udtrækkes koefficienterne fra Lasso-modellen for både `__min` og `__1se`. Antallet af ikke-nul koefficienter anvendes som mål for modellens kompleksitet og graden af variabelselektion.

```
cat("Ridge/Lasso færdig \n")
```

Ridge/Lasso færdig

```
print(perf_glmnet); cat("\n")
```

```
# A tibble: 4 x 5
  model          RMSE  MAE    R2    MSE
  <chr>        <dbl> <dbl> <dbl>  <dbl>
1 Lasso (lambda.min)  219.  178. 0.964 47757.
2 Lasso (lambda.1se)  226.  183. 0.961 51162.
3 Ridge (lambda.min)  438.  337. 0.855 192256.
4 Ridge (lambda.1se)  461.  357. 0.839 212445.
```

```
lasso_coef_min <- as.matrix(coef(cv_lasso, s = "lambda.min"))
lasso_coef_1se <- as.matrix(coef(cv_lasso, s = "lambda.1se"))
```

```
sel_lasso_min <- setdiff(rownames(lasso_coef_min)[lasso_coef_min[,1] != 0], "(Intercept)")
sel_lasso_1se <- setdiff(rownames(lasso_coef_1se)[lasso_coef_1se[,1] != 0], "(Intercept)")
```

```
cat("Lasso non-zero (ekskl. intercept): lambda.min =", length(sel_lasso_min), " lambda.1se =", length(sel_lasso_1se))
```

Lasso non-zero (ekskl. intercept): lambda.min = 25 lambda.1se = 19

Lasso-modellerne outperformer klart Ridge-modellerne på tværs af alle performancemål. Lasso (`_min`) opnår den bedste prædiktionssevne med  $RMSE = 219$ ,  $MAE = 178$ ,  $R^2 = 0,964$  og  $MSE = 47\,757$ , hvilket indikerer høj forklaringsgrad og lav gennemsnitlig fejl. Lasso (`_1se`) reducerer modelkompleksiteten med en begrænset forringelse i performance ( $RMSE = 226$ ,  $MAE = 183$ ,  $R^2 = 0,961$ ,  $MSE = 51\,162$ ). Ridge-modellerne performer markant dårligere, med  $RMSE$  over 438,  $MAE$  over 337 og  $R^2$  under 0,86, hvilket viser, at shrinkage uden eksplicit variabelselektion ikke er tilstrækkelig i denne kontekst.

Resultaterne fra de regulariserede D3-modeller samles i et struktureret objekt, der indeholder datagrundlag, estimerede modeller, performancemål samt udvalgte forklaringsvariable. Denne struktur sikrer reproducerbarhed og muliggør systematisk videreanalyse og sammenligning på tværs af modeltyper.

```
d3_glmnet_results <- list(
  data = list(
    train_cc = train_glm_cc,
    test_cc = test_glm_cc,
    X_train = X_train_glm,
    X_test = X_test_glm,
    y_train = y_train_glm,
    y_test = y_test_glm,
    predictor_pool_glm = predictor_pool_glm,
    foldid_time_blocks = foldid
  ),
  models = list(cv_ridge = cv_ridge, cv_lasso = cv_lasso),
  performance = list(table = perf_glmnet),
  selected_features = list(lasso_lambda_min = sel_lasso_min, lasso_lambda_1se = sel_lasso_1se)
```

```
)
cat("D3 glmnet resultater gemt i: d3_glmnet_results \n\n")
```

D3 glmnet resultater gemt i: d3\_glmnet\_results

Lasso med `__min` udvælger et bredere sæt forklaringsvariable, herunder sæson, kampkontekst, kampnære billetsalgsindikatorer, vejr samt et omfattende sæt modstander-dummies. Dette indikerer, at modellen udnytter både efterspørgselsdynamik tæt på kampdatoen og kontekstuelle faktorer for at maksimere prædiktionssevn.

Ved `__lse` reduceres variabelsættet markant, idet færre modstandere og ingen runde- eller vejrkodevariable indgår, mens de mest centrale forklaringsvariable – billetsalg tre dage før kamp, kortsigtede vækstrater, sæsonstart, kampstartstid, temperatur og placering før kamp – fastholdes. Dette understøtter, at disse variable udgør den robuste kerne i modelleringen, mens øvrige faktorer primært bidrager marginalt til prædiktionssevn.

### 3.11 Fair sammenligning af D3-modeller med og uden befolkningstal

I dette afsnit gennemføres en fair sammenligning af D3-modeller med og uden befolkningstal, hvor både datasætsstørrelse og train-test-split holdes konstant. Formålet er at isolere den marginale effekt af befolkningstal på modellernes prædiktionssevne uden påvirkning fra forskelle i datagrundlag eller modelopsætning. Analysen udføres som et supplement og påvirker ikke projektets hovedmodeller.

Funktionen estimerer en stepwise backward lineær model (AIC) på et givet train-test-split med et lokalt defineret variabelsæt. Predictor-puljen afgrænses til tilgængelige og informative variable, hvorefter der filtreres til complete cases og kategoriske variable håndteres robust ved samling af sjældne niveauer. Modellen evalueres på testdatasættet ved hjælp af RMSE, MAE og  $R^2$ , og både model, performance og anvendt datagrundlag returneres samlet.

```
run_models_on_split <- function(train_df, test_df, predictor_pool_local, tag = "") {
  predictor_pool_local <- predictor_pool_local[predictor_pool_local %in% names(train_df)]
  predictor_pool_local <- predictor_pool_local[vapply(predictor_pool_local, function(v) !all(is.na(train_df[,v])),
    if (length(predictor_pool_local) == 0) stop(" predictor_pool_local endte tom for: ", tag)

  train_step2 <- train_df %>% select(tilskuere, all_of(predictor_pool_local)) %>% drop_na()
  test_step2  <- test_df  %>% select(tilskuere, all_of(predictor_pool_local)) %>% drop_na()

  if ("modstander" %in% names(train_step2)) {
    tmp <- safe_factor_with_other(train_step2$modstander, test_step2$modstander, "ANDRE")
    train_step2$modstander <- tmp$train
    test_step2$modstander  <- tmp$test
    rm(tmp)
  }

  full_formula2 <- as.formula(paste("tilskuere ~", paste(predictor_pool_local, collapse = " + ")))
  m_null2 <- lm(tilskuere ~ 1, data = train_step2)
  m_full2 <- lm(full_formula2, data = train_step2)

  m_bwd2 <- step(m_full2, direction = "backward", trace = 0)
  pred_bwd2 <- clip_0_cap(predict(m_bwd2, newdata = test_step2))

  perf2 <- tibble(
    model = paste0("Stepwise Backward (AIC)", tag),
```

```

  RMSE = rmse_vec(test_step2$tilskuere, pred_bwd2),
  MAE = mae_vec(test_step2$tilskuere, pred_bwd2),
  R2 = r2_vec(test_step2$tilskuere, pred_bwd2),
  n_train_cc = nrow(train_step2),
  n_test_cc = nrow(test_step2)
)

list(model = m_bwd2, performance = perf2, data = list(train = train_step2, test = test_step2))
}

```

Analysen afgrænses til et fast deldatasæt bestående af kampe, hvor befolkningstal er observeret, for at sikre en fair sammenligning. Datasættet opdeles kronologisk i trænings- og testdata (80/20), og centrale kategoriske variable håndteres robust gennem transformation af sæson og samling af sjældne modstandere. Herefter estimeres identiske modeller med og uden befolkningstal på samme datasplit, således at den marginale effekt af befolkningstal på prædiktionsniveauet kan vurderes isoleret.

```

if ("befolkningstal" %in% names(df_d3_all)) {

  df_base_131 <- df_d3_all %>% filter(!is.na(befolkningstal)) %>% arrange(kamp_dato)
  n_total_131 <- nrow(df_base_131)

  if (n_total_131 >= 30) {
    n_test_131 <- max(1, floor(n_total_131 * 0.20))
    cut_idx_131 <- n_total_131 - n_test_131
    train0 <- df_base_131[1:cut_idx_131, , drop = FALSE]
    test0 <- df_base_131[(cut_idx_131 + 1):n_total_131, , drop = FALSE]

    # robusthed: sæson -> sæson_start, modstander -> ANDRE
    train0 <- train0 %>% mutate(sæson_start = season_start_year(sæson)) %>% select(-sæson)
    test0 <- test0 %>% mutate(sæson_start = season_start_year(sæson)) %>% select(-sæson)
    if ("modstander" %in% names(train0)) {
      tmp <- safe_factor_with_other(train0$modstander, test0$modstander, "ANDRE")
      train0$modstander <- tmp$train
      test0$modstander <- tmp$test
      rm(tmp)
    }

    common_pool <- c(
      "sæson_start", "runde", "kamp_time_h",
      "billetter_d3", "vækst_d10_d7", "vækst_d7_d3",
      if ("modstander" %in% names(train0)) "modstander" else NULL,
      "er_helligdag", "er_håndboldkamp_SAH", "antal_håndboldkampe",
      "temperatur_model", "vejrkode_model", "vejr_mangler_info",
      "placering_før_kamp"
    )
    common_pool <- common_pool[common_pool %in% names(train0)]

    predictor_with_pop <- unique(c(common_pool, "befolkningstal"))
    predictor_no_pop <- setdiff(common_pool, "befolkningstal")

    cat("FAIR POP TEST: subset N =", n_total_131, " Train =", nrow(train0), " Test =", nrow(test0), "\n")
  }
}

```

```

res_with_pop <- run_models_on_split(train0, test0, predictor_with_pop, tag = " + POP (samme N)")
res_no_pop    <- run_models_on_split(train0, test0, predictor_no_pop, tag = " (samme N, uden POP)

fair_pop_comparison <- bind_rows(
  res_no_pop$performance %>% mutate(version = "Uden befolkning (samme N)",
  res_with_pop$performance %>% mutate(version = "Med befolkning (samme N)")
) %>% select(version, model, RMSE, MAE, R2, n_train_cc, n_test_cc)

print(fair_pop_comparison); cat("\n")

d3_fair_pop_test <- list(
  subset = list(df_base = df_base_131, train0 = train0, test0 = test0),
  results = list(with_pop = res_with_pop, without_pop = res_no_pop),
  performance = list(fair_pop_comparison = fair_pop_comparison)
)

cat("D3 fair pop-test gemt i: d3_fair_pop_test \n\n")
} else {
  cat("FAIR POP TEST sprunget over: for få rækker med befolkningstal (N < 30)\n\n")
}
} else {
  cat("FAIR POP TEST sprunget over: df_d3_all mangler befolkningstal\n\n")
}

```

FAIR POP TEST: subset N = 131 Train = 105 Test = 26

# A tibble: 2 x 7

	version	model	RMSE	MAE	R2	n_train_cc	n_test_cc
	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<int>	<int>
1	Uden befolkning (samme N)	Stepwise Bac~	193.	155.	0.964	102	23
2	Med befolkning (samme N)	Stepwise Bac~	193.	155.	0.964	102	23

D3 fair pop-test gemt i: d3\_fair\_pop\_test

### 3.11.1 Vurdering af FAIR POP-testen

Den fair sammenligning, hvor både datasætsstørrelse ( $N = 131$ ), train-test-split og modelopsætning holdes konstant, viser ingen forskel i modelperformance ved inkludering af befolkningstal. Modellerne med og uden befolkningstal opnår identiske resultater med  $RMSE = 193$ ,  $MAE = 155$  og  $R^2 = 0,964$ . Derudover udvælger stepwise backward (AIC) i begge tilfælde den samme endelige modelstruktur, hvor befolkningstal ikke indgår som forklaringsvariabel.

Dette indikerer, at befolkningstal ikke bidrager med selvstændig forklaringskraft, når der allerede indgår kampnære efterspørgselsvariable såsom billetsalg tre dage før kamp og kortsigtede vækstrater.

### 3.11.2 Betaler det sig at miste data for at beholde befolkningstal?

Nej. Resultaterne viser klart, at det ikke er fordelagtigt at reducere datagrundlaget for at inkludere befolkningstal. Variablen forbedrer hverken prædiktionssevne eller forklaringsgrad og fravælges endda implicit af modeludvælgelsen. Set i forhold til modellens formål – præcis kortsigtet forudsigelse af tilskuertal – er det metodisk mere hensigtsmæssigt at bevare flest mulige observationer og fokusere på kampnære efterspørgselsdata.

Konklusionen er derfor, at befolkningstal ikke bør indgå i D3-modellen, hvis det kræver tab af observationer, da omkostningen i datatab ikke modsvares af nogen målbar gevinst i performance.

### 3.12 Grafisk sammenligning og modelrepræsentation af de bedste D3-modeller

I dette afsnit visualiseres de bedst performende D3-modeller med henblik på at understøtte den analytiske fortolkning i rapport og mundtlig eksamen. For hver model vises et *Predicted vs. Actual*-plot, hvor både prædiktionssevne og systematiske afvigelser kan vurderes visuelt. Derudover indlejres modelligningen og RMSE direkte i figurene for at koble den statistiske model til dens praktiske præstation.

Nødvendige pakker indlæses med undertrykkelse af opstartsbeskeder for at sikre et ryddeligt output i rapporten. Derudover defineres en hjælpefunktion, som kontrollerer, om påkrævede objekter eksisterer i miljøet, før de anvendes i den efterfølgende visualisering.

```
suppressPackageStartupMessages({
  library(dplyr)
  library(tibble)
  library(ggplot2)
})

.must_exist <- function(x) exists(x, inherits = TRUE)
```

Der defineres en hjælpefunktion til ensartet formatering af numeriske værdier med afrunding og dansk talnotation. Dette sikrer konsistent og læsevenlig præsentation af resultater i de efterfølgende figurer.

```
.format_num <- function(x, digits = 3) {
  format(round(as.numeric(x), digits), big.mark = ".", decimal.mark = ",", trim = TRUE)
}
```

Funktionen genererer en kompakt tekstuel repræsentation af en lineær models ligning ved at fremhæve de forklaringsvariable med størst absolutte koefficienter. Dette muliggør en læsbar præsentation af modellens centrale effekter direkte i de grafiske visualiseringer.

```
# OLS/LM: kort ligning (top |coef|)
lm_equation_text <- function(fit, digits = 3, max_terms = 8) {
  b <- coef(fit)
  b[is.na(b)] <- 0
  nm <- names(b)

  intercept <- if ("(Intercept)" %in% nm) b["(Intercept)"] else 0
  rest <- setdiff(nm, "(Intercept)")

  if (length(rest) == 0) {
    return(paste0("ŷ = ", .format_num(intercept, digits)))
  }

  ord <- order(abs(b[rest]), decreasing = TRUE)
  rest <- rest[ord]
  if (length(rest) > max_terms) rest <- rest[1:max_terms]

  rhs <- vapply(rest, function(v) paste0(v, ".", .format_num(b[v], digits)), character(1))
```

```
paste0("ŷ = ", .format_num(intercept, digits), " + ", paste(rhs, collapse = " + "))
}
```

Funktionen konstruerer en forenklet ligningsrepræsentation af en glmnet-model ved at kombinere interceptet med de ikke-nul koefficienter med størst absolut effekt. Dette muliggør en overskuelig præsentation af de mest betydende variable fra regulariserede modeller i de grafiske fremstillinger.

```
# glmnet: "ligning" = intercept + top non-zero koefficienter
glmnet_equation_text <- function(cvobj, s = "lambda.min", digits = 3, max_terms = 10) {
  cm <- as.matrix(coef(cvobj, s = s))
  if (nrow(cm) == 0) return("Ingen koefficienter")

  coefs <- as.numeric(cm[, 1])
  names(coefs) <- rownames(cm)

  intercept <- if ("(Intercept)" %in% names(coefs)) coefs["(Intercept)"] else 0
  rest <- setdiff(names(coefs), "(Intercept)")

  nz <- rest[coefs[rest] != 0]
  if (length(nz) == 0) {
    return(paste0("ŷ = ", .format_num(intercept, digits), " (alle øvrige = 0)"))
  }

  ord <- order(abs(coefs[nz]), decreasing = TRUE)
  nz <- nz[ord]
  if (length(nz) > max_terms) nz <- nz[1:max_terms]

  rhs <- vapply(nz, function(v) paste0(v, ".", .format_num(coefs[v], digits)), character(1))
  paste0("ŷ = ", .format_num(intercept, digits), " + ", paste(rhs, collapse = " + "))
}
```

Funktionen genererer et *Predicted vs. Actual*-plot, hvor modellens forudsigelser sammenholdes med de observerede værdier. En referencelinje med hældning 1 indikerer perfekt prædiktions, mens RMSE og modelligningen indlejres i figuren for at koble visuel performance med kvantitativ modelkvalitet.

```
# Pred vs Actual plot (med ligning + RMSE)
plot_pva_with_eq <- function(df, title, eq_text, rmse_val) {
  ggplot(df, aes(x = y, y = yhat)) +
    geom_point(alpha = 0.7) +
    geom_abline(slope = 1, intercept = 0) +
    labs(
      title = title,
      x = "Faktisk tilskuertal",
      y = "Forudsagt tilskuertal"
    ) +
    annotate(
      "text",
      x = Inf, y = -Inf,
      hjust = 1.02, vjust = -0.2,
      label = paste0("RMSE = ", .format_num(rmse_val, 1)),
      size = 3.8
    )
}
```

```

) +
  annotate(
    "text",
    x = -Inf, y = Inf,
    hjust = -0.02, vjust = 1.15,
    label = eq_text,
    size = 3.2
  ) +
  theme_minimal()
}

```

Den bedst performende OLS-model baseret på stepwise backward (AIC) evalueres på testdatasættet, og forudsigelser sammenholdes med observerede værdier i et *Predicted vs. Actual*-plot. Modellens ligning og RMSE indlejres i figuren for at give en samlet visuel og kvantitativ vurdering af prædiktionssevnen.

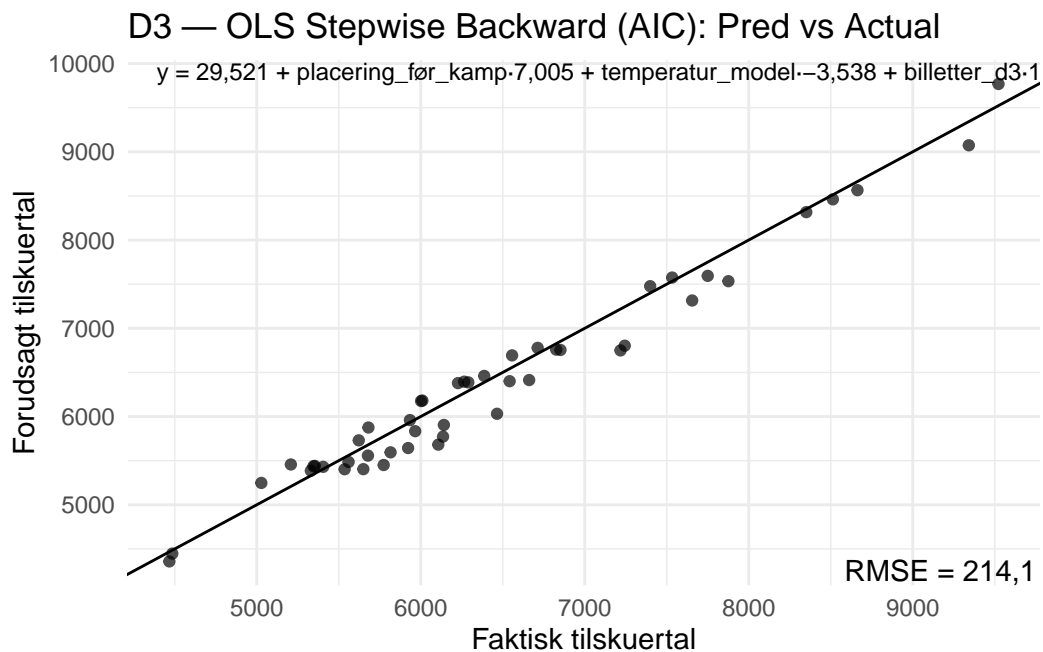
```

# -----
# 10B) OLS (Stepwise backward) - altid hvis findes
# -----
if (!.must_exist("m_step_bwd") || !.must_exist("test_step")) {
  stop(" Kan ikke lave modelplots: mangler m_step_bwd og/eller test_step.")
}

pred_ols <- clip_0_cap(predict(m_step_bwd, newdata = test_step))
df_ols <- tibble(y = as.numeric(test_step$tilskuere), yhat = as.numeric(pred_ols))
rmse_ols <- rmse_vec(df_ols$y, df_ols$yhat)
eq_ols <- lm_equation_text(m_step_bwd, digits = 3, max_terms = 8)

p_ols <- plot_pva_with_eq(
  df = df_ols,
  title = "D3 - OLS Stepwise Backward (AIC): Pred vs Actual",
  eq_text = eq_ols,
  rmse_val = rmse_ols
)
print(p_ols)

```



/

Grafen “D3 — OLS Stepwise Backward (AIC): Pred vs Actual” viser sammenhængen mellem faktiske og forudsagte tilskuertal for den bedst performende D3-model. Ved at sammenholde observationerne med referencelinjen for perfekt prædiktions giver grafen et visuelt grundlag for at vurdere modellens prædiktionssevne og eventuelle systematiske afvigelser.

Best subset-modellen baseret på BIC visualiseres, hvis et gyldigt variabelsæt foreligger, ved at genestimere modellen på træningsdata for at opnå en konsistent og fortolkbar ligning. Modellens forudsigelser evalueres på testdatasættet og præsenteres i et *Predicted vs. Actual*-plot med indlejret ligning og RMSE.

```
# -----
# 10C) Best subset (BIC) - hvis sel_cols findes
#      (refit lm for at få "rigtig" ligning og predict)
# -----
has_subset <- .must_exist("sel_cols") && is.character(sel_cols) && length(sel_cols) > 0 &&
  .must_exist("train_step") && .must_exist("test_step")

if (has_subset) {
  f_sub <- as.formula(paste0("tilskuere ~ ", paste(sel_cols, collapse = " + ")))
  fit_sub_refit <- lm(f_sub, data = train_step)

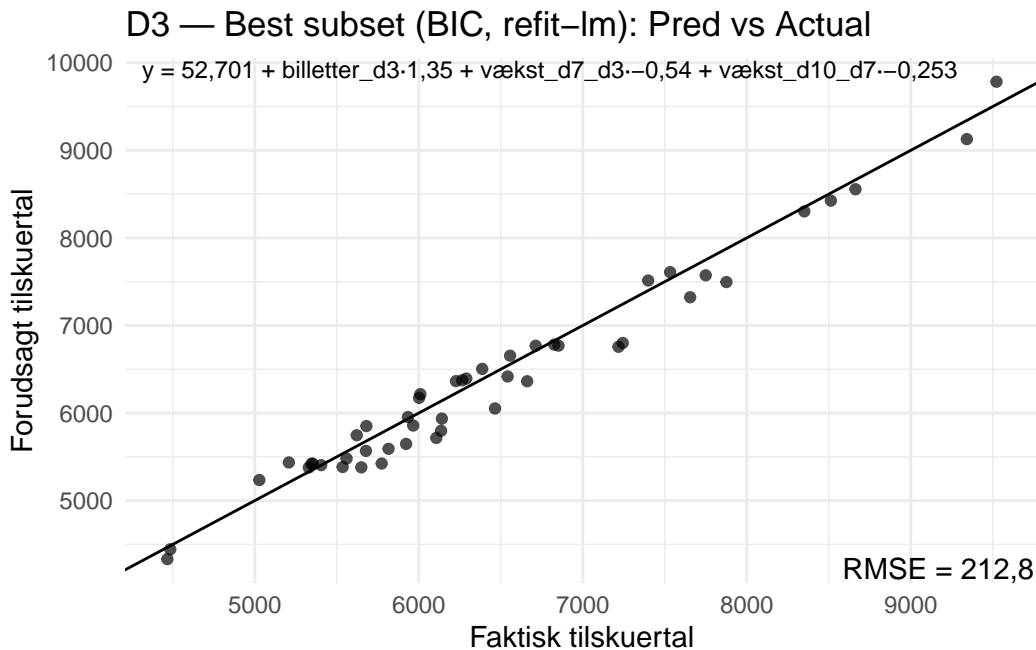
  pred_sub <- clip_0_cap(predict(fit_sub_refit, newdata = test_step))
  df_sub <- tibble(y = as.numeric(test_step$tilskuere), yhat = as.numeric(pred_sub))
  rmse_sub <- rmse_vec(df_sub$y, df_sub$yhat)
  eq_sub <- lm_equation_text(fit_sub_refit, digits = 3, max_terms = 8)

  p_sub <- plot_pva_with_eq(
    df = df_sub,
    title = "D3 - Best subset (BIC, refit-lm): Pred vs Actual",
    eq_text = eq_sub,
    rmse_val = rmse_sub
  )
  print(p_sub)
```

```

} else {
  cat("\n Best subset-plot sprang over (sel_cols/train_step/test_step mangler eller sel_cols er tom).")
}

```



Grafen “D3 — Best subset (BIC, refit-lm): Pred vs Actual” illustrerer forholdet mellem faktiske og forudsagte tilskuertal for D3-modellen estimeret via best subset-udvælgelse baseret på BIC. Grafen viser, hvordan en relativt kompakt model med få centrale forklaringsvariable formår at ramme observationerne tæt langs referencelinjen for perfekt prædiktions. Den lave RMSE indikerer, at modellen opnår en præcision, der er fuldt på højde med – og marginalt bedre end – den tilsvarende stepwise-model, samtidig med at modelkompleksiteten holdes nede.

Lasso- og Ridge-modellerne visualiseres for den optimale regulariseringsparameter `_min` ved hjælp af *Predicted vs. Actual*-plots, hvor RMSE og en forenklet ligningsrepræsentation indlejres i figurene. Derudover vises krydsvalideringskurver for begge modeller for at illustrere sammenhængen mellem regularisering og valideringsfejl. Dette giver et samlet visuelt grundlag for at vurdere både prædiktionssevne og modelkompleksitet.

```

# -----
# 10D) glmnet: Lasso/Ridge (lambda.min) - hvis objekter findes
# -----
has_glmnet <- .must_exist("cv_lasso") && .must_exist("cv_ridge") && .must_exist("X_test_glm") && .must_exist("y_test_glm")

if (has_glmnet) {
  pred_lasso <- clip_0_cap(as.numeric(predict(cv_lasso, newx = X_test_glm, s = "lambda.min")))
  pred_ridge <- clip_0_cap(as.numeric(predict(cv_ridge, newx = X_test_glm, s = "lambda.min")))

  df_lasso <- tibble(y = as.numeric(y_test_glm), yhat = as.numeric(pred_lasso))
  df_ridge <- tibble(y = as.numeric(y_test_glm), yhat = as.numeric(pred_ridge))

  rmse_lasso <- rmse_vec(df_lasso$y, df_lasso$yhat)
  rmse_ridge <- rmse_vec(df_ridge$y, df_ridge$yhat)

  eq_lasso <- glmnet_equation_text(cv_lasso, s = "lambda.min", digits = 3, max_terms = 10)
  eq_ridge <- glmnet_equation_text(cv_ridge, s = "lambda.min", digits = 3, max_terms = 10)
}

```

```

p_lasso <- plot_pva_with_eq(
  df = df_lasso,
  title = "D3 - Lasso (lambda.min): Pred vs Actual",
  eq_text = eq_lasso,
  rmse_val = rmse_lasso
)
p_ridge <- plot_pva_with_eq(
  df = df_ridge,
  title = "D3 - Ridge (lambda.min): Pred vs Actual",
  eq_text = eq_ridge,
  rmse_val = rmse_ridge
)

print(p_lasso)
print(p_ridge)

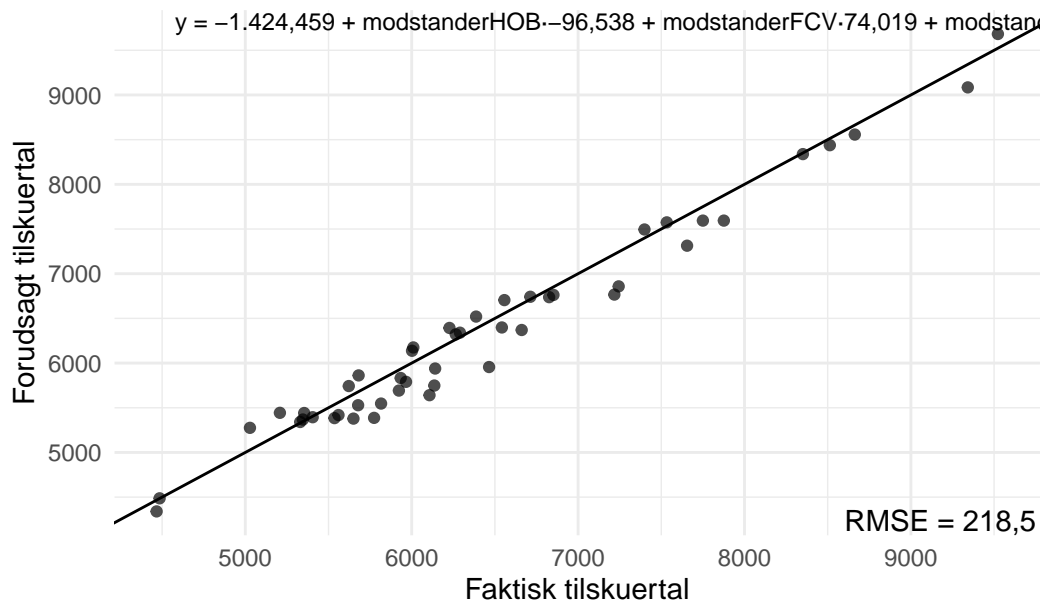
# CV-kurver som ggplot - pænt i rapport
plot_cv_glmnet <- function(cvobj, title = "") {
  df <- tibble(lambda = cvobj$lambda, cvm = cvobj$cvm, cvsd = cvobj$cvsd)
  lam_min <- cvobj$lambda.min
  lam_1se <- cvobj$lambda.1se

  ggplot(df, aes(x = log(lambda), y = cvm)) +
    geom_line() +
    geom_line(aes(y = cvm + cvsd), linetype = 2) +
    geom_line(aes(y = cvm - cvsd), linetype = 2) +
    geom_vline(xintercept = log(lam_min), linetype = 3) +
    geom_vline(xintercept = log(lam_1se), linetype = 3) +
    labs(title = title, x = "log()", y = "CV MSE") +
    theme_minimal()
}

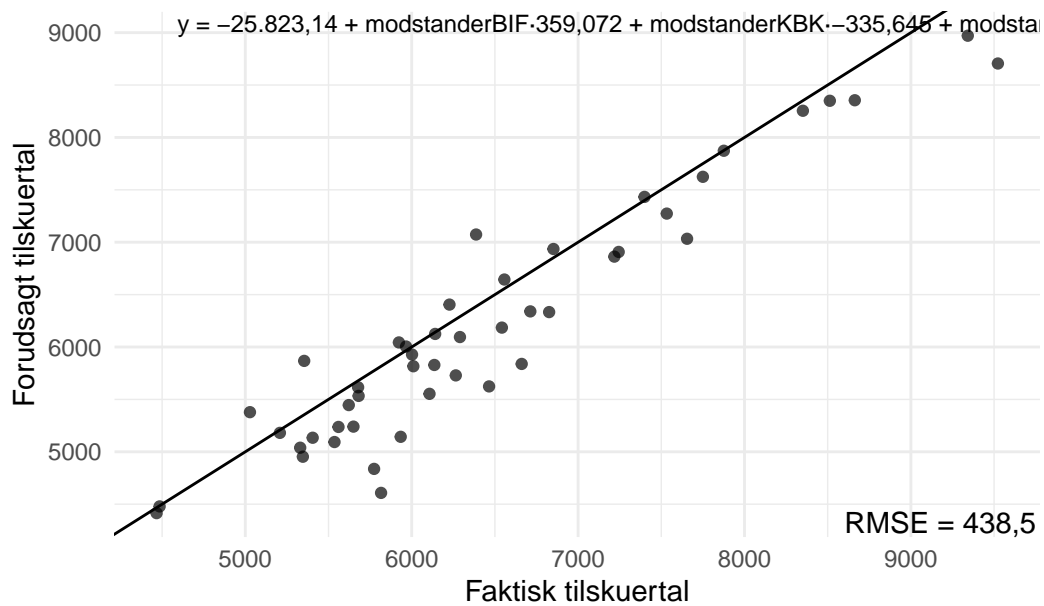
print(plot_cv_glmnet(cv_lasso, "D3 - Lasso CV-kurve (ggplot)"))
print(plot_cv_glmnet(cv_ridge, "D3 - Ridge CV-kurve (ggplot)"))
} else {
  cat("\n glmnet-plots sprang over (mangler cv_lasso/cv_ridge/X_test_glm/y_test_glm).\n")
}

```

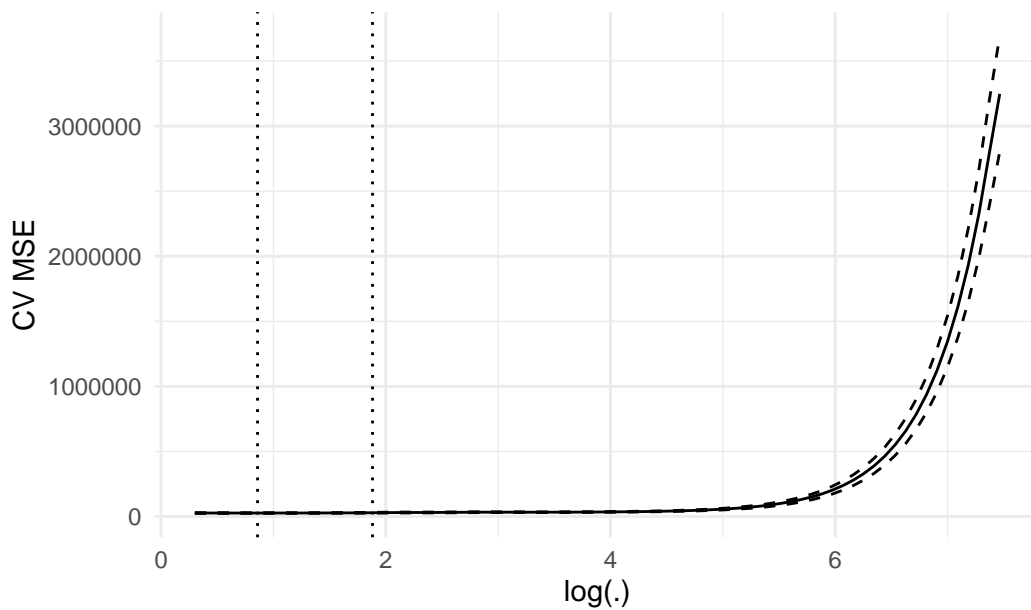
### D3 — Lasso (lambda.min): Pred vs Actual



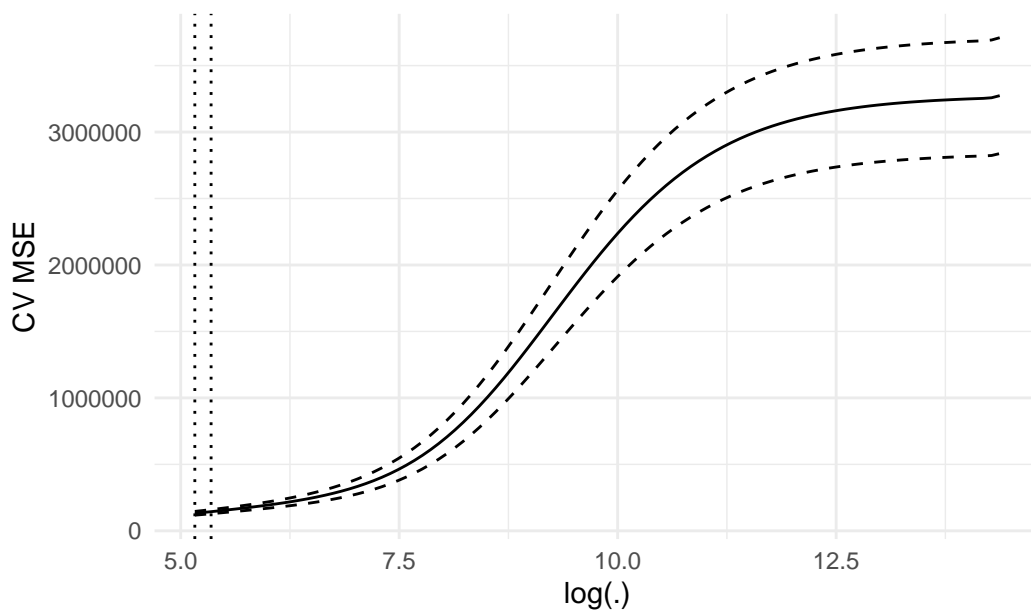
### D3 — Ridge (lambda.min): Pred vs Actual



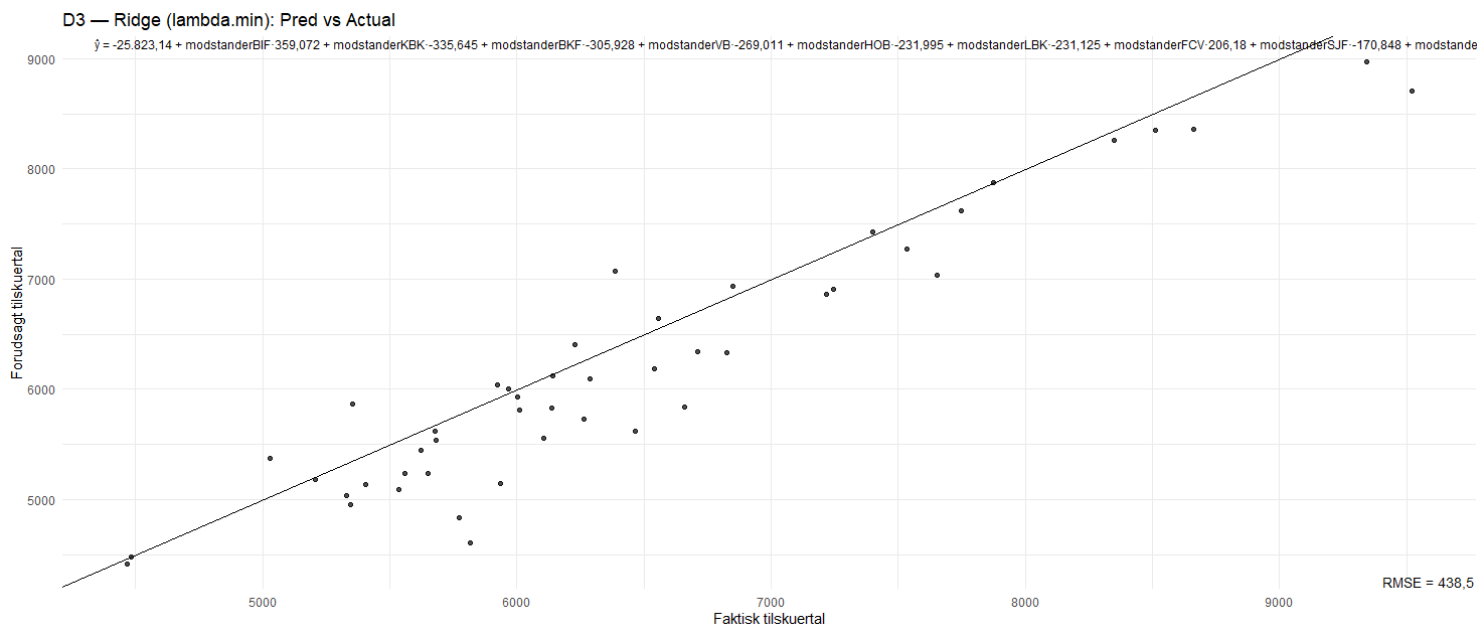
D3 — Lasso CV-kurve (ggplot)



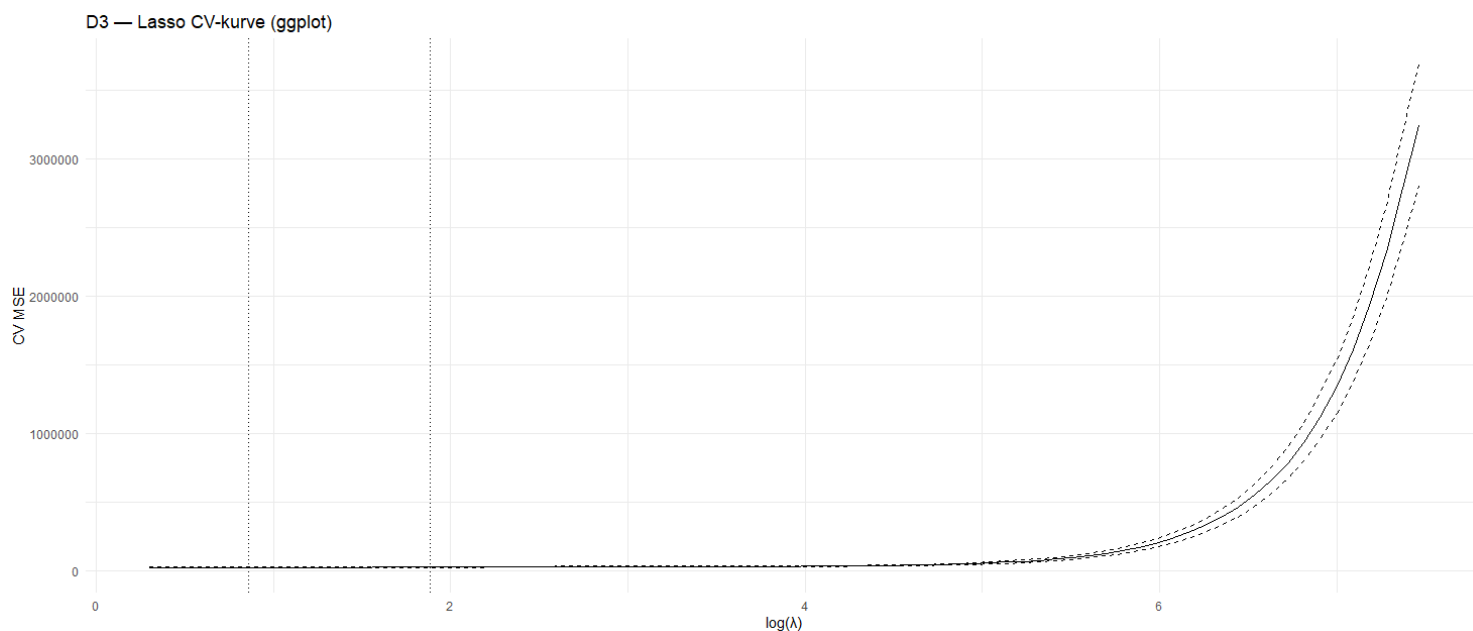
D3 — Ridge CV-kurve (ggplot)



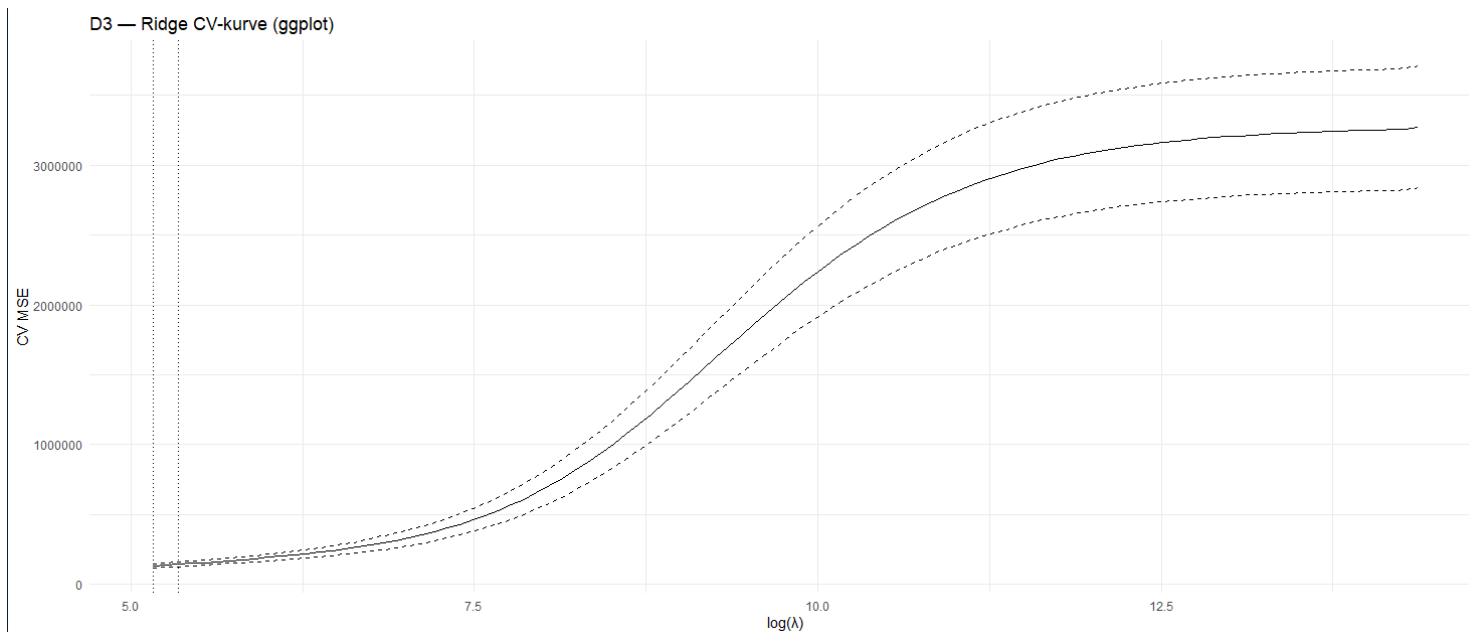
Grafen “D3 — Lasso (lambda.min): Pred vs Actual” viser sammenhængen mellem faktiske og forudsagte tilskuertal for D3-modellen estimeret med Lasso-regularisering ved  $\lambda_{\min}$ . Punkterne ligger generelt tæt omkring 45-graderslinjen, hvilket indikerer en høj grad af overensstemmelse mellem modelprediktioner og observerede værdier. Den opnåede RMSE på omkring 218 viser, at modellen præsterer på et niveau, der er sammenligneligt med de bedste OLS-baserede modeller, men med den væsentlige forskel, at Lasso automatisk har foretaget variabelselektion. Det betyder, at modellen reducerer kompleksiteten ved at sætte mindre relevante koefficienter til nul, samtidig med at den bevarer en høj prædiktionssevne.



Grafen “D3 —” illustrerer sammenhængen mellem faktiske og forudsagte tilskuertal for D3-modellen estimeret med Ridge-regularisering ved `_min`. I modsætning til de øvrige modeller ses en tydelig større spredning omkring 45-graderslinjen, hvilket indikerer en lavere prædiktionssevne. Den relativt høje RMSE på cirka 439 afspejler, at modellen har vanskeligt ved præcist at ramme både lave og høje tilskuertal. Resultatet viser, at Ridge-modellen – trods regularisering – ikke formår at tilpasse sig datamønstrene på samme niveau som OLS- og Lasso-modellerne i denne D3-kontekst.



Når vi ser på grafen “D3 — Lasso CV-kurve (ggplot)”, illustrerer den, hvordan modellens prædiktionsfejl varierer som funktion af regulariseringsparameteren  $\lambda$  i Lasso-modellen. X-aksen viser  $\log(\lambda)$ , mens y-aksen viser den krydsvaliderede middelmåbte fejl (CV MSE), som er beregnet via tidsblok-baseret krydsvalidering. Kurvens laveste punkt angiver det `_min`-niveau, hvor modellen opnår den bedste gennemsnitlige performance på tværs af foldene (`_min`). De lodrette stiplede linjer markerer henholdsvis `_min` og `_1se`, hvor sidstnævnte repræsenterer en mere konservativ model, der accepterer en marginalt højere fejl til gengæld for øget regularisering og lavere modelkompleksitet. Grafen viser samtidig, at fejlen stiger markant ved høje  $\lambda$ -værdier, hvilket afspejler underfitting, når for mange koefficienter presses mod nul.



Når vi betragter grafen “D3 — Ridge CV-kurve (ggplot)”, viser den sammenhængen mellem regulariseringsparameteren og modellens krydsvaliderede fejl for Ridge-modellen. X-aksen angiver  $\log(\lambda)$ , mens y-aksen viser den krydsvaliderede middelvadrerede fejl (CV MSE), beregnet via tidsblok-opdelt krydsvalidering. Kurven illustrerer, at modellen har lavest fejl ved relativt lave  $\lambda$ -værdier, hvor regulariseringen er svag. De lodrette stiplede linjer markerer  $\lambda_{\min}$  og  $\lambda_{1se}$ , som angiver henholdsvis den  $\lambda$ -værdi med lavest gennemsnitlig CV-fejl og en mere konservativ løsning inden for én standardafvigelse. I takt med at  $\lambda$  øges, stiger fejlen markant, hvilket indikerer underfitting som følge af kraftig shrinkage af koefficienterne. I modsætning til Lasso sætter Ridge ikke koefficienter præcist til nul, og grafen afspejler derfor, at øget regularisering hurtigt reducerer modellens fleksibilitet uden at tilføre stabil prædiktionssevne i denne D3-kontekst.

### 3.13 Operationalisering af D3-modeller via Shiny-applikation

For at demonstrere, hvordan de estimerede D3-modeller kunnen anvendes i praksis, er der udviklet en interaktiv Shiny-applikation. Applikationen indgår ikke som en integreret del af selve QMD-filen, men præsenteres her som dokumentation for, at modellerne er operationaliseret og kan anvendes i en beslutningsstøttekontekst.

Formålet med applikationen er at afprøve modellernes anvendelighed i et realistisk scenarie, hvor en bruger – eksempelvis en medarbejder i Viborg FF – kan indtaste kamp- og kontekstspecifikke oplysninger og modtage en konkret prognose for det forventede tilskuertal. Dermed fungerer applikationen som et bindeled mellem den statistiske modellering og den forretningsmæssige anvendelse.

Applikationen bygger direkte på de samme modeller, datastrukturer og prædiktionsstrin, som er anvendt i analyse- og evalueringsafsnittene. Koden gennemgås derfor på samme måde som resten af projektets pipeline, med fokus på transparens, reproducerbarhed og konsistens mellem analyse og produkt.

Koden indlæser de nødvendige pakker og sikrer, at den estimerede D3-model samt træningsdata er tilgængelige, så applikationen anvender samme grundlag som analysen.

```
# shiny

suppressPackageStartupMessages({
  library(shiny)
  library(dplyr)
  library(ggplot2)
})
```

```
if (!exists("m_step_bwd", inherits = TRUE)) stop("Mangler m_step_bwd i environment.")
if (!exists("train_step", inherits = TRUE)) stop("Mangler train_step i environment.")
```

Denne hjælpefunktion anvendes til at formatere numeriske værdier ensartet, så modeloutput præsenteres læsbart med korrekt afrunding og dansk talnotation.

```
# shiny

.format_num <- function(x, digits = 3) {
  format(round(as.numeric(x), digits), big.mark = ".", decimal.mark = ",", trim = TRUE)
}
```

Funktionen genererer en kort, læsbar repræsentation af en lineær models ligning ved at vise interceptet og de mest betydende koefficienter baseret på absolut størrelse.

```
# shiny

# Kort ligning for lm: viser intercept + top |coef|
lm_equation_text <- function(fit, digits = 3, max_terms = 8) {
  b <- coef(fit)
  b[is.na(b)] <- 0
  nm <- names(b)

  intercept <- if ("(Intercept)" %in% nm) b["(Intercept)"] else 0
  rest <- setdiff(nm, "(Intercept)")

  if (length(rest) == 0) return(paste0("ŷ = ", .format_num(intercept, digits)))

  ord <- order(abs(b[rest]), decreasing = TRUE)
  rest <- rest[ord]
  if (length(rest) > max_terms) rest <- rest[1:max_terms]

  rhs <- vapply(rest, function(v) paste0(v, ".", .format_num(b[v], digits)), character(1))
  paste0("ŷ = ", .format_num(intercept, digits), " + ", paste(rhs, collapse = " + "))
}
```

Denne hjælpefunktion sikrer, at brugerinput castes til samme datatype og struktur som i træningsdata, så prædiktioner kan foretages korrekt og robust.

```
# shiny

# Helper: lav korrekt datatype ud fra train_step
.cast_like_train <- function(var, value) {
  ref <- train_step[[var]]

  if (is.factor(ref)) {
    lvls <- levels(ref)
    v <- as.character(value)
    if (is.na(v) || !(v %in% lvls)) {
      if ("ANDRE" %in% lvls) v <- "ANDRE" else v <- lvls[1]
    }
  }
}
```

```

    }
    return(factor(v, levels = lvls))
  }

  if (inherits(ref, "Date")) return(as.Date(value))
  if (is.integer(ref)) return(as.integer(value))
  if (is.numeric(ref)) return(as.numeric(value))

  as.character(value)
}

```

Funktionen opretter dynamiske inputfelter i brugergrænsefladen baseret på variablenes datatyper i træningsdata, så brugeren kun kan indtaste gyldige værdier for den valgte model.

```

# shiny

# UI controls dynamisk ud fra valgte model_vars + train_step
.make_input_control <- function(var) {
  x <- train_step[[var]]

  if (is.factor(x)) {
    selectInput(
      inputId = var,
      label = var,
      choices = levels(x),
      selected = levels(x)[1]
    )
  } else if (is.numeric(x) || is.integer(x)) {
    rng <- range(x, na.rm = TRUE)
    val <- stats::median(x, na.rm = TRUE)
    numericInput(
      inputId = var,
      label = var,
      value = val,
      min = rng[1],
      max = rng[2]
    )
  } else if (inherits(x, "Date")) {
    dateInput(
      inputId = var,
      label = var,
      value = Sys.Date()
    )
  } else {
    textInput(
      inputId = var,
      label = var,
      value = ""
    )
  }
}
}

```

Funktionen klassificerer den estimerede stadionfyldning i diskrete niveauer baseret på procentvis kapacitetsudnyttelse, så resultatet kan formidles enkelt og intuitivt.

```
# shiny

# Fyldnings-buckets
.fill_bucket <- function(pct) {
  if (!is.finite(pct)) return("UKENDT")
  if (pct >= 85) return("HØJ")
  if (pct >= 60) return("MIDDEL")
  "LAV"
}
```

Denne kode kontrollerer, om en Best Subset-model kan genskabes på baggrund af de valgte variable og træningsdata. Hvis det er muligt, refittes modellen og vælges som standard; ellers anvendes den estimerede stepwise OLS-model.

```
# shiny

# -----
# Byg Best Subset refit-model hvis muligt
# -----
best_subset_available <- exists("sel_cols", inherits = TRUE) &&
  is.character(sel_cols) && length(sel_cols) > 0 &&
  all(sel_cols %in% names(train_step))

fit_best_subset <- NULL
if (best_subset_available) {
  f_sub <- as.formula(paste0("tilskuere ~ ", paste(sel_cols, collapse = " + ")))
  fit_best_subset <- lm(f_sub, data = train_step)
}

# Default model: bedste hvis available, ellers stepwise
default_model_key <- if (!is.null(fit_best_subset)) "best_subset" else "stepwise"
```

Denne kode kontrollerer, om en Best Subset-model kan genskabes på baggrund af de valgte variable og træningsdata. Hvis det er muligt, refittes modellen og vælges som standard; ellers anvendes den estimerede stepwise OLS-model.

```
# shiny

# -----
# UI
# -----
ui <- fluidPage(
  titlePanel("D3 - Prediction (Modelvalg)"),

  sidebarLayout(
    sidebarPanel(
      tags$h4("Model"),
```

```

selectInput(
  "model_choice",
  "Vælg model (Bedste model valgt by default",
  choices = c(
    "Best subset (BIC) - refit-lm" = "best_subset",
    "OLS stepwise backward (AIC)" = "stepwise"
  ),
  selected = default_model_key
),
tags$hr(),

tags$h4("Inputs (kun de variable modellen bruger)"),
uiOutput("dyn_inputs"),

tags$hr(),
numericInput("capacity", "Stadion-kapacitet", value = 10000, min = 1),
actionButton("go", "Predict", class = "btn-primary")
),

mainPanel(
  tags$h4("Output"),
  tags$div(style="font-size:20px; margin-bottom:8px;",
    textOutput("pred_txt")),
  tags$div(style="font-size:20px; margin-bottom:8px;",
    textOutput("pct_txt")),
  tags$div(style="font-size:20px; margin-bottom:8px;",
    textOutput("delta_txt")),
  tags$div(style="font-size:14px; margin-bottom:14px; color:#333;",
    textOutput("eq_txt")),
  plotOutput("cap_plot", height = "520px")
)
)
)

```

Denne server-del udgør den operationelle kerne i Shiny-applikationen og binder model, input og output sammen i en interaktiv arbejdsgang.

Serveren håndterer først valg af aktiv model (Best Subset eller OLS stepwise) og sikrer et robust fallback, hvis den foretrukne model ikke er tilgængelig. På baggrund af den valgte model identificeres de relevante forklarende variable, hvorefter inputfelterne genereres dynamisk, så brugeren kun præsenteres for variable, der faktisk indgår i modellen.

Når brugeren aktiverer en prediction, konstrueres et korrekt typet input-datasæt, som anvendes til at beregne det forudsagte tilskuertal. Resultatet efterbehandles med simple, forretningsnære mål såsom kapacitetsudnyttelse og forventet ekstra billetsalg fra D3 til kampdag. Output præsenteres både tekstuelt og grafisk, herunder en kapacitetsgraf, der tydeliggør forholdet mellem den estimerede efterspørgsel og stadionets maksimale kapacitet.

Samlet set demonstrerer server-logikken, hvordan de estimerede modeller kan operationaliseres i et beslutningsstøtteværktøj, der omsætter statistiske modeller til konkrete og anvendelige indsigter for en organisation.

```
# shiny
```

```
# -----
```

```

# Server
# -----
server <- function(input, output, session) {

  # Aktiv model (reactive)
  active_model <- reactive({
    key <- input$model_choice

    if (key == "best_subset") {
      if (is.null(fit_best_subset)) {
        return(list(
          key = "stepwise",
          fit = m_step_bwd,
          name = "OLS stepwise backward (AIC) (fallback)",
          note = "Best subset er ikke tilgængelig (sel_cols mangler eller passer ikke til train_step)."
        ))
      }
      return(list(
        key = "best_subset",
        fit = fit_best_subset,
        name = "Best subset (BIC) - refit-lm",
        note = ""
      ))
    }

    list(
      key = "stepwise",
      fit = m_step_bwd,
      name = "OLS stepwise backward (AIC)",
      note = ""
    )
  })

  # Hvilke variable bruger den valgte model?
  model_vars <- reactive({
    fit <- active_model()$fit
    vars <- all.vars(stats::terms(fit)) |> setdiff("tilskuere")
    vars
  })

  # Dynamiske inputs
  output$dyn_inputs <- renderUI({
    tagList(lapply(model_vars(), .make_input_control))
  })

  # Prediction objekt
  pred_obj <- eventReactive(input$go, {

    fit <- active_model()$fit
    vars <- model_vars()
  })
}

```

```

# Byg én-rækkes newdata med korrekt type pr variabel
nd <- as.data.frame(setNames(replicate(length(vars), NA, simplify = FALSE), vars))

for (v in vars) {
  nd[[v]] <- .cast_like_train(v, input[[v]])
}

# Predict
yhat <- suppressWarnings(as.numeric(predict(fit, newdata = nd)))
if (length(yhat) == 0) yhat <- NA_real_

# Clip til ikke-negative (ingen hård cap på 1.000.000 i pipeline, men vi holder det her som sikker)
yhat_clip <- max(min(yhat, 1000000), 0)

cap <- as.numeric(input$capacity)
if (!is.finite(cap) || cap <= 0) cap <- 1

pct <- (yhat_clip / cap) * 100
pct_clip <- max(min(pct, 100), 0)

# Ekstra billetsalg fra D3 -> kampdag:  $\hat{y}$  - billetter_d3 (kun hvis billetter_d3 i modellen/inputs)
bil_d3 <- if ("billetter_d3" %in% vars) as.numeric(input$billetter_d3) else NA_real_
delta <- if (is.finite(bil_d3)) (yhat_clip - bil_d3) else NA_real_

eq_text <- lm_equation_text(fit, digits = 3, max_terms = 10)

list(
  pred = yhat_clip,
  pct = pct_clip,
  cap = cap,
  bil_d3 = bil_d3,
  delta = delta,
  eq = eq_text,
  model_name = active_model()$name,
  model_note = active_model()$note
)
})

output$pred_txt <- renderText({
  x <- pred_obj()
  if (is.null(x)) return("Ingen prediction endnu.")
  paste0("Model: ", x$model_name, "\nForudsagt tilskuertal: ",
    format(round(x$pred), big.mark = ".", decimal.mark = ","))
})

output$pct_txt <- renderText({
  x <- pred_obj()
  if (is.null(x)) return("")
  paste0("Stadion fyldt: ", format(round(x$pct, 1), big.mark = ".", decimal.mark = ","), " %")
})

```

```

output$delta_txt <- renderText({
  x <- pred_obj()
  if (is.null(x)) return("")
  if (!is.finite(x$bil_d3)) return("Ekstra billetter (D3 → kampdag): kan ikke beregnes (billetter_d3
paste0(
  "Ekstra billetter (D3 → kampdag): ",
  format(round(x$delta), big.mark = ".", decimal.mark = ","),
  " (ŷ - billetter_d3)"
)
})

output$eq_txt <- renderText({
  x <- pred_obj()
  if (is.null(x)) return("")
  note <- if (!is.null(x$model_note) && nzchar(x$model_note)) paste0("\nNote: ", x$model_note) else
  paste0("Ligning (kort): ", x$eq, note)
})

output$cap_plot <- renderPlot({
  x <- pred_obj()
  if (is.null(x)) return(NULL)

  bucket <- .fill_bucket(x$pct)

  df <- data.frame(
    label = "Stadionkapacitet",
    y = x$pred,
    bucket = bucket
  )

  ggplot(df, aes(x = label, y = y, fill = bucket)) +
    geom_col(width = 0.6) +
    geom_hline(yintercept = x$cap, linewidth = 0.9, linetype = 2) +
    coord_cartesian(ylim = c(0, x$cap)) +
    scale_fill_manual(
      values = c("LAV" = "#d55e00", "MIDDEL" = "#e69f00", "HØJ" = "#009e73", "UKENDT" = "grey60"),
      breaks = c("LAV", "MIDDEL", "HØJ", "UKENDT"),
      name = "Fyldningsniveau"
    ) +
    labs(
      x = NULL,
      y = "Tilskuere (forudsagt)",
      title = "Forudsagt tilskuertal vs stadionkapacitet",
      subtitle = paste0(
        "Kapacitet (stiplet linje): ",
        format(round(x$cap), big.mark=".", decimal.mark=",")
      )
    ) +
    theme_minimal(base_size = 16) +
    theme(
      legend.position = "top",

```

```

    plot.title = element_text(face = "bold"),
    panel.grid.minor = element_blank()
  )
})
}

shinyApp(ui, server)

```

## 3.14 To scenarier for tilskuertal i Superligaen baseret på D3-modellen

### 3.14.1 Scenarie 1 – Høj interesse og næsten fyldt stadion

Palle arbejder med billetsalg og planlægning hos Viborg FF og står tre dage før en Superliga-hjemmekamp over for en modstander, der historisk trækker mange tilskuere. Allerede på D3-tidspunktet er der solgt 7.200 billetter. I perioden fra 10 til 7 dage før kamp er billetsalget steget med 1.400 billetter, og fra 7 dage til 3 dage før kamp er salget yderligere accelereret med 1.900 billetter.

#### D3 — Prediction (Modelvalg)

Model
Vælg model (Bedste model valgt by default)
Best subset (BIC) — refit-lm

Inputs (kun de variable modellen bruger)
billetter\_d3
7200
vækst\_d10\_d7
1400
vækst\_d7\_d3
1900
Stadion-kapacitet
10000
Predict

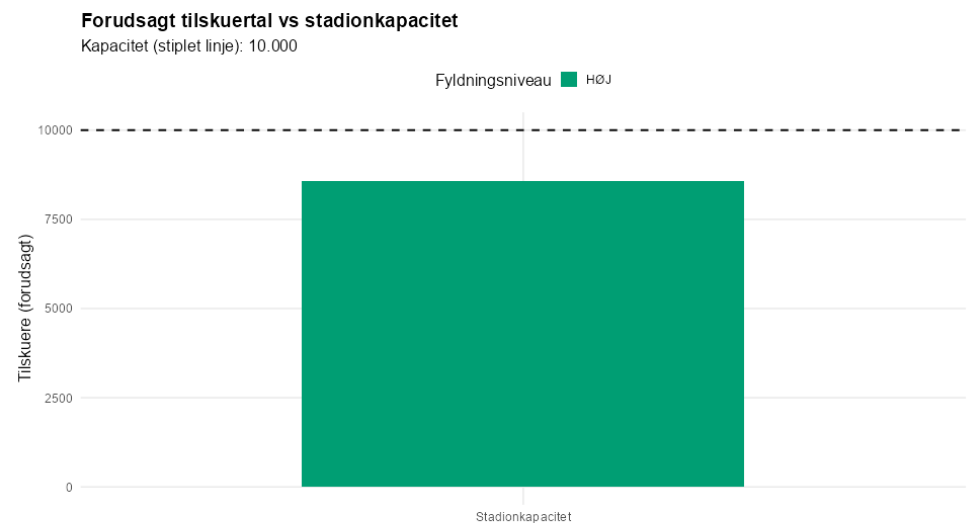
Output

Model: Best subset (BIC) — refit-lm Forudsagt tilskuertal: 8.573

Stadion fyldt: 85,7 %

Ekstra billetter (D3 → kampdag): 1.373 ( $\hat{y}$  - billetter\_d3)

Ligning (kort):  $\hat{y} = 52,701 + \text{billetter\_d3} \cdot 1,35 + \text{vækst\_d7\_d3} \cdot -0,54 + \text{vækst\_d10\_d7} \cdot -0,253$



Disse tal indtaster Palle i Shiny-applikationen, hvor stadionkapaciteten er sat til 10.000 pladser. Den valgte model (Best subset, BIC) estimerer på baggrund af disse inputs et forventet tilskuertal på 8.573, svarende til en fyldningsgrad på 85,7 %. Modellen indikerer dermed, at der fortsat kan forventes et betydeligt billetsalg frem mod kampdagen, med cirka 1.373 ekstra solgte billetter fra D3 til kickoff.

For Palle betyder dette, at kampen med høj sandsynlighed bliver tæt på udsolgt. Resultatet kan bruges direkte i den operative planlægning, eksempelvis i forhold til bemanning, catering, sikkerhed og markedsføring, hvor fokus nu kan flyttes fra bred synlighed til målrettet sidste-øjebliks salg.

### 3.14.2 Scenarie 2 – Lav efterspørgsel og behov for salgsindsats

Daniel arbejder i den kommercielle afdeling og analyserer en anden Superliga-hjemmekamp, hvor interessen ser mere afdæmpet ud. Tre dage før kamp er der solgt 3.501 billetter. Billetsalget har haft en moderat udvikling, med en stigning på 733 billetter fra 10 til 7 dage før kamp og yderligere 911 billetter fra 7 til 3 dage før kamp.

#### D3 — Prediction (Modelvalg)

**Model**  
Vælg model (Bedste model valgt by default)  
Best subset (BIC) — refit-lm

Inputs (kun de variable modellen bruger)

**billetter\_d3**  
3501

**vækst\_d10\_d7**  
733

**vækst\_d7\_d3**  
911

**Stadion-kapacitet**  
10000

Predict

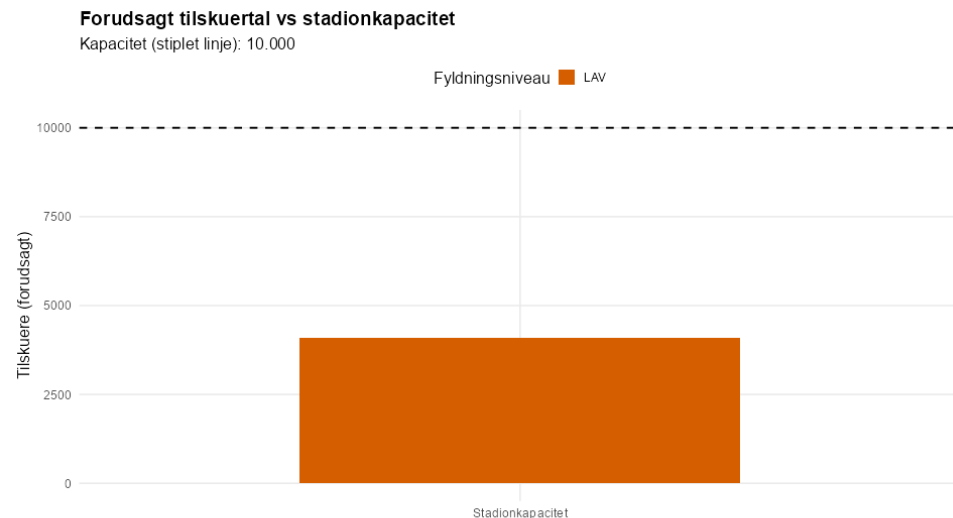
Output

Model: Best subset (BIC) — refit-lm Forudsagt tilskuertal: 4.101

Stadion fyldt: 41 %

Ekstra billetter (D3 → kampdag): 600 ( $\hat{y}$  – billetter\_d3)

Ligning (kort):  $\hat{y} = 52,701 + \text{billetter\_d3} \cdot 1,35 + \text{vækst\_d7\_d3} \cdot -0,54 + \text{vækst\_d10\_d7} \cdot -0,253$



Daniel indtaster disse værdier i Shiny-appen sammen med stadionkapaciteten på 10.000. Modellen estimerer her et forventet tilskuertal på 4.101, hvilket svarer til en fyldningsgrad på cirka 41 %. Der forventes kun omkring 600 ekstra solgte billetter frem mod kampdagen, hvis udviklingen fortsætter uændret.

Dette scenarie giver Daniel et klart beslutningsgrundlag: uden en ekstra salgs- eller marketingindsats vil kampen sandsynligvis få et lavt fremmøde. Modellen kan dermed bruges som et tidligt advarselssignal, der gør det muligt at iværksætte kampagner, tilbud eller samarbejder med kort varsel for at løfte interessen.

## 4 ML model D7

Her præsenteres udviklingen af **MODEL\_D7**, som har til formål at forudsige tilskuertallet til Viborg FF's hjemmekampe **7 dage før kampstart**. Modellen indgår i et samlet prognoseframework med fokus på robuste, forklarlige og operationelle løsninger frem for maksimal kompleksitet.

Analysen tager udgangspunkt i et baseline-datasæt med én observation pr. kamp, som gradvist udvides med relevante forklaringsvariable, herunder billetsalg, kampkontekst, kalendereffekter, vejr og temperatur samt udvalgte strukturelle forhold. Nye variable testes systematisk mod en fast baseline og vurderes på baggrund af statistiske kriterier, modelkvalitet og database.

Den endelige D7-model evalueres ved et tidssplit og sammenlignes på tværs af OLS-, Ridge- og Lasso-modeller. Resultaterne anvendes både analytisk og i et efterfølgende produktlag, hvor modellen operationaliseres gennem visualiseringer og en interaktiv applikation.

## 4.1 Pakker og globale indstillinger

Denne kodeblok indlæser de nødvendige pakker til analysen. Samtidig justeres globale indstillinger for mere læsbart output.

```
if (!requireNamespace("pacman", quietly = TRUE)) {  
  install.packages("pacman")  
}  
  
suppressPackageStartupMessages({  
  pacman::p_load(  
    DBI,  
    odbc,  
    dplyr,  
    tidyr,  
    lubridate,  
    stringr,  
    tibble,  
    ggplot2,  
    glmnet,  
    rsample  
  )  
})  
  
options(scipen = 999)  
cat("Pakker er klar \n\n")
```

Pakker er klar

## 4.2 Hjælpfunktioner

Denne funktion sikrer, at alle nødvendige miljøvariabler til databaseforbindelsen er korrekt sat. Hvis kravene ikke er opfyldt, stoppes analysen for at undgå skjulte forbindelsesfejl senere i pipeline'en.

### 4.2.1 .Renviron

```
ensure_renviro <- 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")  
}
```

### 4.2.2 Robust databaseforbindelse

Denne funktion opretter en stabil forbindelse til Azure SQL med gentagne forsøg og stigende timeouts. Formålet er at reducere midlertidige forbindelsesfejl og sikre, at analysen kan køre reproducerbart.

```
connect_azure_retry <- function(forsøg_max = 6, timeouts = c(60, 180, 200, 260, 360, 600), delay_sec = 3) {
  for (forsøg in seq_len(forsøg_max)) {
    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"),
        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) Sys.sleep(delay_sec)
  }
  stop(" Kunne ikke forbinde til Azure SQL efter ", forsøg_max, " forsøg.")
}
```

### 4.2.3 Sikker indlæsning af databaser

Denne funktion læser tabeller fra databasen med indbyggede genforsøg ved fejl. Den sikrer, at midlertidige læsefejl ikke stopper analysen unødigt.

```
safe_dbReadTable <- function(con, schema, table, forsøg_max = 4, delay_sec = 3) {
  for (i in seq_len(forsøg_max)) {
    out <- try(DBI::dbReadTable(con, DBI::Id(schema = schema, table = table)), silent = TRUE)
    if (!inherits(out, "try-error")) return(out)
    msg <- as.character(out)
    cat(" dbReadTable fejlede (forsøg", i, "af", forsøg_max, "):", schema, ".", table, "\n")
    cat(substr(msg, 1, 220), "... \n\n")
    if (i < forsøg_max) Sys.sleep(delay_sec)
  }
  stop(" Kunne ikke læse tabel: ", schema, ".", table)
}
```

#### 4.2.4 Normalisering af tidsformater

Denne funktion sikrer ensartet tidsformat ved at udvide klokkeslæt uden sekunder. Det gør efterfølgende tidsbaserede beregninger mere robuste.

```
normalize_time_chr <- function(x) {  
  x <- str_trim(as.character(x))  
  if_else(str_detect(x, "^\\d{1,2}:\\d{2}$"), paste0(x, ":00"), x)  
}
```

#### 4.2.5 Udtræk af timekomponent

Denne funktion konverterer tidsstrengene til heltals-timer. Det muliggør simple og konsistente tidsafhængige features i analysen.

```
time_chr_to_hour <- function(x) {  
  x <- normalize_time_chr(x)  
  ok <- str_detect(x, "^\\d{1,2}:\\d{2}:\\d{2}$")  
  out <- rep(NA_integer_, length(x))  
  out[ok] <- as.integer(str_extract(x[ok], "^\\d{1,2}"))  
  out  
}
```

#### 4.2.6 Parsing af observationsdatoer

Denne funktion sikrer korrekt konvertering af datoer til Date-format. Det giver konsistent datohåndtering på tværs af datakilder.

```
parse_obs_date <- function(x) {  
  if (inherits(x, "Date")) return(x)  
  as.Date(x, format = "%d-%m-%Y")  
}
```

#### 4.2.7 Udledning af sæsonens startår

Denne funktion udtrækker sæsonens startår fra sæsonbetegnelser. Det muliggør numerisk modellering af sæsonvariationer.

```
season_start_year <- function(x) {  
  x <- as.character(x)  
  suppressWarnings(as.integer(sub("^([0-9]{4}).*$", "\\1", x)))  
}
```

#### 4.2.8 Evalueringsmål

Disse funktioner beregner RMSE, MAE og  $R^2$  til vurdering af modellernes præcision. De anvendes konsekvent til sammenligning af modelperformance.

```
rmse_vec <- function(y, yhat) sqrt(mean((y - yhat) ^ 2, na.rm = TRUE))
mae_vec  <- function(y, yhat) mean(abs(y - yhat), na.rm = TRUE)
r2_vec <- function(y, yhat) {
  sse <- sum((y - yhat) ^ 2, na.rm = TRUE)
  sst <- sum((y - mean(y, na.rm = TRUE)) ^ 2, na.rm = TRUE)
  1 - (sse / sst)
}
```

#### 4.2.9 Hjælpefunktioner til datarensning

Disse funktioner sikrer henholdsvis, at forudsigelser ikke bliver negative, og at der kun bevares én observation pr. nøgle. Ved dubletter prioriteres rækker med færrest manglende værdier.

```
clip_nonneg <- function(x) pmax(x, 0)

dedup_by_na <- function(df, id_col) {
  id_col <- rlang::ensym(id_col)
  df2 <- df
  df2$.na_count <- rowSums(is.na(df2))
  df2 |>
    arrange(!!id_col, .na_count) |>
    group_by(!!id_col) |>
    slice(1) |>
    ungroup() |>
    select(-.na_count)
}
```

#### 4.2.10 Samlet feature-testfunktion

Denne funktion sammenligner systematisk en baseline-model med en udvidet model, hvor én ekstra variabel tilføjes. Effekten vurderes via modelresumé, ANOVA og AIC for at sikre konsistente og sammenlignelige beslutninger om feature-inklusion.

```
run_feature_test <- function(df, base_vars, add_var = NULL, label = "TEST", force_int = character()) {
  vars <- base_vars
  if (!is.null(add_var)) vars <- c(vars, add_var)

  d <- df |>
    select(all_of(vars)) |>
    drop_na() |>
    mutate(
      sæson = as.factor(sæson),
      across(any_of(force_int), as.integer)
    )

  f_base <- as.formula(paste0("tilskuere ~ ", paste(base_vars[base_vars != "tilskuere"], collapse = " + "))
  m_base <- lm(f_base, data = d)

  if (is.null(add_var)) {
    cat("=====\n")
  }
```

```

cat(label, "- SUMMARY\n")
cat("=====\n")
print(summary(m_base))
cat("\nAIC:\n")
print(AIC(m_base))
return(invisible(list(data = d, m_base = m_base, m_full = m_base)))
}

f_full <- as.formula(paste0("tilskuere ~ ", paste(vars[vars != "tilskuere"], collapse = " + ")))
m_full <- lm(f_full, data = d)

cat("=====\n")
cat(label, "- SUMMARY\n")
cat("=====\n")
print(summary(m_full))

cat("\n=====\n")
cat("ANOVA: BASE vs +", add_var, "\n")
cat("=====\n")
print(anova(m_base, m_full))

cat("\n=====\n")
cat("AIC-SAMMENLIGNING\n")
cat("=====\n")
print(AIC(m_base, m_full))

invisible(list(data = d, m_base = m_base, m_full = m_full))
}

```

#### 4.2.11 Miljø- og forbindelsestjek

Denne kode sikrer, at nødvendige miljøvariabler er sat korrekt og opretter herefter forbindelse til databasen. Dermed stoppes analysen tidligt, hvis forudsætningerne ikke er opfyldt.

```

# =====
# 2) Safeguard: .Renviron + connection
# =====
ensure_renviron()

```

```
.Renviron OK
```

```
con <- connect_azure_retry()
```

Forsøg 1 - ConnectionTimeout = 60 sekunder  
 Forbundet til Azure SQL på forsøg 1

## 4.2.12 Indlæsning af baseline-datasæt

Denne kode indlæser det centrale baseline-datasæt med én observation pr. kamp. Samtidig foretages simple type- og strukturchecks for at sikre datakvalitet.

```
baseline_dir <- "C:/Users/janpe/OneDrive/Skrivebord/PBA Dataanalyse/01_Første semester/1 Semester proj  
baseline_file <- file.path(baseline_dir, "baseline_azure.rds")  
stopifnot(file.exists(baseline_file))  
  
baseline <- readRDS(baseline_file) |>  
  mutate(kamp_id = as.character(kamp_id), kamp_dato = as.Date(kamp_dato))  
  
stopifnot(all(c("kamp_id", "kamp_dato", "tilskuere") %in% names(baseline)))  
cat("Baseline loaded   Rækker:", nrow(baseline), "Unikke kamp_id:", n_distinct(baseline$kamp_id), "\n")
```

Baseline loaded Rækker: 259 Unikke kamp\_id: 254

## 4.2.13 Klargøring af kampstartstid og baseline-kvalitet

Denne kode sikrer, at kampstartstid er tilgængelig, ensartet formateret og konverteret til numerisk time. Samtidig udføres et kvalitetstjek og deduplikering, så baseline ender med én entydig række pr. kamp.

```
# Hvis kamp_tid mangler: hent fra future_baseline_feature_azure.rds (samme som du gør)  
if (!("kamp_tid" %in% names(baseline))) {  
  cat(" baseline_azure.rds har ikke 'kamp_tid'. Henter fra future_baseline_feature_azure.rds...\n")  
  feature_baseline_file <- file.path(baseline_dir, "future_baseline_feature_azure.rds")  
  stopifnot(file.exists(feature_baseline_file))  
  
  fb <- readRDS(feature_baseline_file) |>  
    mutate(kamp_id = as.character(kamp_id), kamp_dato = as.Date(kamp_dato)) |>  
    arrange(kamp_id) |>  
    group_by(kamp_id) |>  
    slice(1) |>  
    ungroup() |>  
    select(kamp_id, kamp_tid)  
  
  baseline <- baseline |> left_join(fb, by = "kamp_id")  
  cat(" kamp_tid joinet ind (1 række pr kamp_id)\n\n")  
  rm(fb)  
}  
  
baseline <- baseline |>  
  mutate(  
    kamp_tid_chr = normalize_time_chr(kamp_tid),  
    kamp_time_h  = time_chr_to_hour(kamp_tid_chr)  
  )  
  
cat("Kamp-rækker uden parsebar kamp_time_h:", sum(is.na(baseline$kamp_time_h)), "\n\n")
```

Kamp-rækker uden parsebar kamp\_time\_h: 0

```
cat("\n--- BASELINE QA: DUPLIKAT-TJEK ---\n")
```

--- BASELINE QA: DUPLIKAT-TJEK ---

```
dup_ids <- baseline |> count(kamp_id, sort = TRUE) |> filter(n > 1)
cat("Antal kamp_id med dubletter:", nrow(dup_ids), "\n")
```

Antal kamp\_id med dubletter: 5

```
if (nrow(dup_ids) > 0) {
  baseline <- dedup_by_na(baseline, kamp_id)
  stopifnot(nrow(baseline) == n_distinct(baseline$kamp_id))
  cat(" Baseline deduplikeret (1 række pr kamp_id)\n\n")
} else {
  cat(" Ingen dubletter fundet\n\n")
}
```

Baseline deduplikeret (1 række pr kamp\_id)

```
baseline_key <- baseline |> transmute(kamp_id, kamp_dato, kamp_time_h)
```

## 4.2.14 Features & variabler

### 4.2.14.1 Helligdage

Denne kode tilføjer en binær indikator for, om kampen afvikles på en helligdag. Formålet er at indfange eventuelle kalender-effekter på tilskuertallet.

```
hellig_raw <- safe_dbReadTable(con, "PBA01_Raw", "dim_helligdage_dkk_raw")
stopifnot(all(c("dato", "helligdag_navn") %in% names(hellig_raw)))

hellig_min <- hellig_raw |>
  mutate(hellig_dato = as.Date(dato)) |>
  filter(!is.na(hellig_dato)) |>
  group_by(hellig_dato) |>
  summarise(er_helligdag = 1L, .groups = "drop")

baseline <- baseline |>
  left_join(hellig_min, by = c("kamp_dato" = "hellig_dato")) |>
  mutate(er_helligdag = if_else(is.na(er_helligdag), 0L, er_helligdag))

cat("Helligdage jointet \n\n")
```

Helligdage jointet

#### 4.2.14.2 Samtidige håndboldkampe (SAH)

Denne kode tilføjer information om samtidige håndboldkampe i SAH på kampdagen. Formålet er at modellere potentiel konkurrence om tilskuernes opmærksomhed.

```
sah_raw <- safe_dbReadTable(con, "PBA02_Clean", "fact_håndboldkampe_SAH_clean")
stopifnot(all(c("kamp_dato", "kamp_tid", "Event") %in% names(sah_raw)))

sah_min <- sah_raw |>
  mutate(sah_dato = as.Date(kamp_dato)) |>
  filter(!is.na(sah_dato)) |>
  group_by(sah_dato) |>
  summarise(
    er_håndboldkamp_SAH = 1L,
    antal_håndboldkampe = n(),
    .groups = "drop"
  )

baseline <- baseline |>
  left_join(sah_min, by = c("kamp_dato" = "sah_dato")) |>
  mutate(
    er_håndboldkamp_SAH = if_else(is.na(er_håndboldkamp_SAH), 0L, er_håndboldkamp_SAH),
    antal_håndboldkampe = if_else(is.na(antal_håndboldkampe), 0L, antal_håndboldkampe)
  )

cat("Håndbold SAH joinet \n\n")
```

Håndbold SAH joinet

#### 4.2.14.3 Befolkning

Denne kode tilføjer kvartalsvise befolkningstal til hver kamp ved at matche med nærmeste tidligere observation. Formålet er at indfange langsigtede strukturelle ændringer i befolkningsgrundlaget.

```
bef_raw <- safe_dbReadTable(con, "PBA03_JoinReady", "fact_Viborg_befolkning_join_ready")
stopifnot(all(c("køn", "civilstand", "dato", "befolkningstal") %in% names(bef_raw)))

bef_all <- bef_raw |>
  mutate(
    køn = str_to_lower(trimws(as.character(køn))),
    civilstand = str_to_lower(trimws(as.character(civilstand))),
    dato = as.Date(dato),
    befolkningstal = as.numeric(befolkningstal)
  ) |>
  filter(
    !is.na(dato),
    !is.na(befolkningstal),
    køn %in% c("i alt", "alt"),
    civilstand %in% c("i alt", "alt")
  ) |>
  distinct(dato, .keep_all = TRUE) |>
```

```

  arrange(dato) |>
  transmute(bef_dato = dato, befolkningstal = befolkningstal)

baseline_bef <- baseline |>
  transmute(kamp_id, kamp_dato) |>
  left_join(bef_all, join_by(closest(kamp_dato >= bef_dato)))

baseline <- baseline |>
  left_join(baseline_bef |> select(kamp_id, befolkningstal), by = "kamp_id")

cat("Befolkning joinet \n\n")

```

Befolkning joinet

#### 4.2.14.4 Vejrdata (nærmeste observation)

Denne kode matcher hver kamp med den tidsmæssigt nærmeste vejr-observation på kampdagen ud fra faste måletidspunkter. Formålet er at tilknytte robuste vejrfeatures og samtidig håndtere manglende eller ufuldstændige observationer.

```
cat("\n--- FEATURE: VEJR (nærmeste blandt 08/11/14/17) ---\n")
```

```
--- FEATURE: VEJR (nærmeste blandt 08/11/14/17) ---
```

```

vejr_sql <- safe_dbReadTable(con, "PBA03_JoinReady", "fact_vejr_join_ready")
stopifnot(all(c("obs_dato", "tid", "vejrkode", "vejrbeskrivelse") %in% names(vejr_sql)))

build_weather_match <- function(baseline_key, vejr_sql, grid_hours = c(8L, 11L, 14L, 17L)) {
  vejr_pre <- vejr_sql |>
    mutate(
      vejr_dato = parse_obs_date(obs_dato),
      vejr_tid_chr = normalize_time_chr(tid),
      vejr_time_h = time_chr_to_hour(vejr_tid_chr),
      vejrkode = suppressWarnings(as.integer(vejrkode)),
      vejrbeskrivelse = as.character(vejrbeskrivelse)
    ) |>
    filter(!is.na(vejr_dato), !is.na(vejr_time_h))

  vejr_grid <- vejr_pre |>
    filter(vejr_time_h %in% grid_hours) |>
    group_by(vejr_dato, vejr_time_h) |>
    slice(1) |>
    ungroup()

  vejr_fallback_1 <- vejr_pre |>
    arrange(vejr_dato, vejr_time_h) |>
    group_by(vejr_dato) |>
    slice(1) |>
    ungroup()

```

```

kandidater <- baseline_key |>
  filter(!is.na(kamp_dato), !is.na(kamp_time_h)) |>
  inner_join(vejr_grid |> select(vejr_dato, vejr_time_h, vejrcode, vejrbeskrivelse),
    by = c("kamp_dato" = "vejr_dato")) |>
  mutate(dist_hours_to_kickoff = abs(kamp_time_h - vejr_time_h)) |>
  group_by(kamp_id) |>
  arrange(dist_hours_to_kickoff, .by_group = TRUE) |>
  slice(1) |>
  ungroup() |>
  transmute(
    kamp_id,
    vejr_tid_h = vejr_time_h,
    vejrcode,
    vejrbeskrivelse,
    dist_hours_to_kickoff,
    vejr_match_type = "grid_nearest"
  )

mangler <- baseline_key |> anti_join(kandidater |> select(kamp_id), by = "kamp_id")

fallback <- mangler |>
  left_join(vejr_fallback_1 |> select(vejr_dato, vejr_time_h, vejrcode, vejrbeskrivelse),
    by = c("kamp_dato" = "vejr_dato")) |>
  transmute(
    kamp_id,
    vejr_tid_h = vejr_time_h,
    vejrcode,
    vejrbeskrivelse,
    dist_hours_to_kickoff = NA_integer_,
    vejr_match_type = "fallback_dato"
  )

out <- bind_rows(kandidater, fallback)
stopifnot(sum(duplicated(out$kamp_id)) == 0)
out
}

```

#### 4.2.14.5 Sammenkobling af vejrdato

Her kobles de matchede vejrdato på baseline-datasættet.

Der udskrives samtidig simple diagnosticer for manglende eller ikke-observerbare vejrcoder.

```

vejr_match <- build_weather_match(baseline_key, vejr_sql)

baseline <- baseline |>
  left_join(vejr_match, by = "kamp_id")

cat("Vejr joinet \n")

```

Vejr joinet

```
cat("Kampe uden vejrkode (NA):", sum(is.na(baseline$vejrkode)), "\n")
```

Kampe uden vejrkode (NA): 38

```
cat("Kampe med vejrkode=0:", sum(baseline$vejrkode == 0, na.rm = TRUE), "\n\n")
```

Kampe med vejrkode=0: 70

#### 4.2.14.6 Robust håndtering af manglende vejrdato

Denne kode adskiller manglende og ikke-observerbare vejrdato og omsætter dem til konsistente model-features. Formålet er at bevare alle kampe i datasættet uden at indføre skjult bias fra manglende vejrinformation.

```
baseline <- baseline |>
  mutate(
    vejrmangler_obs = if_else(is.na(vejrkode), 1L, 0L),
    vejrikke_observerbar = if_else(!is.na(vejrkode) & vejrkode == 0, 1L, 0L),
    vejrmangler_info = if_else(vejrmangler_obs == 1L | vejrikke_observerbar == 1L, 1L, 0L),

    vejrkode_model = if_else(vejrmangler_info == 1L, -1L, as.integer(vejrkode)),
    vejrbeskrivelse_model = case_when(
      vejrmangler_obs == 1L ~ "UKENDT",
      vejrikke_observerbar == 1L ~ "IKKE_OBSERVERBAR",
      TRUE ~ as.character(vejrbeskrivelse)
    ),
    vejrtid_h_model = if_else(vejrmangler_info == 1L, -1L, as.integer(vejrtid_h)),
    dist_hours_to_kickoff_model = if_else(vejrmangler_info == 1L, -1L, as.integer(dist_hours_to_kickoff)),
    vejrmatch_type_model = if_else(is.na(vejrmatch_type), "NO_WEATHER", as.character(vejrmatch_type))
  )

cat("--- VEJR HÅNTERING ---\n")
```

--- VEJR HÅNTERING ---

```
cat("vejrmangler_obs (NA):", sum(baseline$vejrmangler_obs == 1L, na.rm = TRUE), "\n")
```

vejrmangler\_obs (NA): 38

```
cat("vejrikke_observerbar (kode=0):", sum(baseline$vejrikke_observerbar == 1L, na.rm = TRUE), "\n")
```

vejrikke\_observerbar (kode=0): 70

```
cat("vejrmangler_info (NA eller 0):", sum(baseline$vejrmangler_info == 1L, na.rm = TRUE), "\n\n")
```

vejrmangler\_info (NA eller 0): 108

#### 4.2.14.7 Temperatur før kamp

Denne kode matcher hver kamp med den senest observerede temperatur før kickoff via en bagudgående tidslogik. Formålet er at knytte en realistisk og konsistent temperaturfeature til kampafviklingen.

```
cat("\n--- FEATURE: TEMPERATUR (bagud-trappe 18→15→12→09) ---\n")
```

```
--- FEATURE: TEMPERATUR (bagud-trappe 18→15→12→09) ---
```

```
temp_sql <- safe_dbReadTable(con, "PBA03_JoinReady", "fact_temperatur_join_ready")
stopifnot(all(c("obs_dato", "tid", "temperatur") %in% names(temp_sql)))
```

```
build_temp_match <- function(baseline_key, temp_sql, hours = c(9L, 12L, 15L, 18L)) {
  temp_grid <- temp_sql |>
  mutate(
    temp_dato = parse_obs_date(obs_dato),
    temp_tid_chr = normalize_time_chr(tid),
    temp_time_h = time_chr_to_hour(temp_tid_chr),
    temperatur = suppressWarnings(as.numeric(temperatur))
  ) |>
  filter(!is.na(temp_dato), !is.na(temp_time_h), temp_time_h %in% hours) |>
  group_by(temp_dato, temp_time_h) |>
  slice(1) |>
  ungroup()
```

```
kandidater <- baseline_key |>
  inner_join(temp_grid |> select(temp_dato, temp_time_h, temperatur),
    by = c("kamp_dato" = "temp_dato")) |>
  filter(!is.na(kamp_time_h), temp_time_h <= kamp_time_h) |>
  mutate(temp_dist_hours_to_kickoff = kamp_time_h - temp_time_h)
```

```
valgt <- kandidater |>
  group_by(kamp_id) |>
  arrange(desc(temp_time_h), .by_group = TRUE) |>
  slice(1) |>
  ungroup() |>
  transmute(
    kamp_id,
    temp_tid_h = temp_time_h,
    temperatur,
    temp_dist_hours_to_kickoff = as.integer(temp_dist_hours_to_kickoff),
    temp_match_type = "step_back_same_day"
  )
```

```
mangler <- baseline_key |>
  anti_join(valgt |> select(kamp_id), by = "kamp_id") |>
  transmute(
    kamp_id,
    temp_tid_h = NA_integer_,
    temperatur = NA_real_,
```

```

    temp_dist_hours_to_kickoff = NA_integer_,
    temp_match_type = case_when(
      is.na(kamp_time_h) ~ "NO_KICKOFF_TIME",
      TRUE ~ "NO_TEMP_BEFORE_KICKOFF"
    )
  )
)

out <- bind_rows(valgt, mangler)
stopifnot(sum(duplicated(out$kamp_id)) == 0)
out
}

```

#### 4.2.14.8 Sammenkobling og håndtering af temperaturdata

Her kobles de matchede temperaturdata på baseline og omsættes til modelklare features. Der udskrives samtidig diagnostik for manglende temperaturobservationer og match-typer.

```

temp_match <- build_temp_match(baseline_key, temp_sql)

baseline <- baseline |>
  left_join(temp_match, by = "kamp_id") |>
  mutate(
    temp_mangler_obs = if_else(is.na(temperatur), 1L, 0L),
    temperatur_model = as.numeric(temperatur),
    temp_tid_h_model = as.integer(temp_tid_h),
    temp_dist_hours_to_kickoff_model = as.integer(temp_dist_hours_to_kickoff),
    temp_match_type_model = if_else(is.na(temp_match_type), "NO_TEMP", as.character(temp_match_type))
  )

cat("Temperatur joinet \n")

```

Temperatur joinet

```
cat("step_back_same_day:", sum(baseline$temp_match_type == "step_back_same_day", na.rm = TRUE), "\n")
```

step\_back\_same\_day: 253

```
cat("NO_KICKOFF_TIME:", sum(baseline$temp_match_type == "NO_KICKOFF_TIME", na.rm = TRUE), "\n")
```

NO\_KICKOFF\_TIME: 0

```
cat("NO_TEMP_BEFORE_KICKOFF:", sum(baseline$temp_match_type == "NO_TEMP_BEFORE_KICKOFF", na.rm = TRUE)
```

NO\_TEMP\_BEFORE\_KICKOFF: 1

```
cat("NA temperatur i alt:", sum(is.na(baseline$temperatur)), "\n\n")
```

NA temperatur i alt: 1

#### 4.2.14.9 Billetsalg og vækstfeatures

Denne sektion konstruerer centrale billetsalgsfeatures til D7-modellen, herunder solgte billetter 7 dage før kamp samt vækst i billetsalg fra dag 10 til dag 7. Formålet er at indfange den kortsigtede efterspørgsel tæt på kampdatoen i et konsistent og modelklart format.

```
vff_billetsalg <- safe_dbReadTable(con, "PBA03_JoinReady", "fact_VFF_Billetsalg_join_ready") |>
  mutate(
    kamp_id      = as.character(kamp_id),
    d10_tilskuere = as.numeric(d10_tilskuere),
    d7_tilskuere  = as.numeric(d7_tilskuere)
  )
stopifnot(all(c("kamp_id", "d10_tilskuere", "d7_tilskuere") %in% names(vff_billetsalg)))
cat("Billetsalg hentet   Rækker:", nrow(vff_billetsalg), "\n\n")
```

Billetsalg hentet Rækker: 259

#### 4.2.14.10 Konstruktion af D7-billetsalgsfeatures

Denne kode aggregerer billetsalgsdata til én observation pr. kamp og konstruerer centrale D7-features. De afledte variable kobles herefter på baseline-datasættet og danner grundlag for den videre analyse.

```
vff_billetsalg_1row <- vff_billetsalg |>
  group_by(kamp_id) |>
  summarise(
    d10_tilskuere = max(d10_tilskuere, na.rm = TRUE),
    d7_tilskuere  = max(d7_tilskuere,   na.rm = TRUE),
    .groups = "drop"
  ) |>
  mutate(
    d10_tilskuere = if_else(is.infinite(d10_tilskuere), NA_real_, d10_tilskuere),
    d7_tilskuere  = if_else(is.infinite(d7_tilskuere),   NA_real_, d7_tilskuere)
  )

tickets_feats_d7 <- vff_billetsalg_1row |>
  mutate(
    billetter_d7 = d7_tilskuere,
    vækst_d10_d7 = pmax(d7_tilskuere - d10_tilskuere, 0)
  ) |>
  select(kamp_id, billetter_d7, d10_tilskuere, d7_tilskuere, vækst_d10_d7)

d7_dataset <- baseline |> left_join(tickets_feats_d7, by = "kamp_id")
cat("Join gennemført   Rækker:", nrow(d7_dataset),
    "Unikke kamp_id:", n_distinct(d7_dataset$kamp_id), "\n\n")
```

Join gennemført Rækker: 254 Unikke kamp\_id: 254

## 4.2.15 Diagnostik og datakvalitetstjek

Denne kode opsummerer datakvaliteten i D7-datasættet efter alle joins.

Formålet er at give et hurtigt overblik over datatab, manglende observationer og kvaliteten af feature-matchene.

```
diag <- d7_dataset |>
  summarise(
    rækker_total = n(),
    unikke_kamp_id = n_distinct(kamp_id),
    uden_d7 = sum(is.na(billetter_d7)),
    uden_bef = sum(is.na(befolkningstal)),
    uden_vejrkode_raw = sum(is.na(vejrkode)),
    vejrkode0_raw = sum(vejrkode == 0, na.rm = TRUE),
    uden_temp_raw = sum(is.na(temperatur)),
    vejrmangler_obs_total = sum(vejrmangler_obs == 1L, na.rm = TRUE),
    tempmangler_obs_total = sum(tempmangler_obs == 1L, na.rm = TRUE),
    vejrgrid_match = sum(vejrmatch_type == "grid_nearest", na.rm = TRUE),
    vejrfallback = sum(vejrmatch_type == "fallback_dato", na.rm = TRUE),
    tempstepback = sum(tempmatch_type == "step_back_same_day", na.rm = TRUE),
    tempnomatch = sum(tempmatch_type == "NO_TEMP_BEFORE_KICKOFF", na.rm = TRUE)
  )
print(diag)
```

# A tibble: 1 x 13

```
  rækker_total unikke_kamp_id uden_d7 uden_bef uden_vejrkode_raw vejrkode0_raw
    <int>         <int>    <int>    <int>         <int>         <int>
1       254         254      0      123             38             70
# i 7 more variables: uden_temp_raw <int>, vejrmangler_obs_total <int>,
#   tempmangler_obs_total <int>, vejrgrid_match <int>, vejrfallback <int>,
#   tempstepback <int>, tempnomatch <int>
```

## 4.2.16 Endeligt D7-analysedatasæt

Denne kode filtrerer datasættet til kampe med komplette billetsalgsfeatures og færdiggør kodningen af modelvariable.

Resultatet er et konsistent og modelklart D7-datasæt, som anvendes i de efterfølgende modeltests.

```
analysis_df_d7 <- d7_dataset |>
  filter(!is.na(billetter_d7), !is.na(vækst_d10_d7)) |>
  mutate(
    vejrkode_model = if_else(is.na(vejrkode_model), -1L, vejrkode_model),
    vejrbeskrivelse_model = if_else(is.na(vejrbeskrivelse_model), "UKENDT", vejrbeskrivelse_model),
    vejrtid_h_model = if_else(is.na(vejrtid_h_model), -1L, vejrtid_h_model),
    dist_hours_to_kickoff_model = if_else(is.na(dist_hours_to_kickoff_model), -1L, dist_hours_to_kickoff_model),
    vejrmatch_type_model = if_else(is.na(vejrmatch_type_model), "NO_WEATHER", vejrmatch_type_model),
    temperatur_model = as.numeric(temperatur_model),
    temp_tid_h_model = as.integer(temp_tid_h_model),
    temp_dist_hours_to_kickoff_model = as.integer(temp_dist_hours_to_kickoff_model),
    tempmatch_type_model = if_else(is.na(tempmatch_type_model), "NO_TEMP", as.character(tempmatch_type_model))
  )

cat("Analyse-datasæt (D7) klar   Rækker:", nrow(analysis_df_d7), "\n\n")
```

Analyse-datasæt (D7) klar      Rækker: 254

#### 4.2.17 Opsætning af baseline til modeltests

Her defineres de faste baseline-variable, som anvendes i alle efterfølgende modeltests.  
Det sikrer, at effekten af nye features vurderes konsistent mod samme reference-model.

```
base_vars <- c("tilskuere","sæson","runde","kamp_time_h","billetter_d7","vækst_d10_d7")
```

#### 4.2.18 Nulmodel (baseline)

Her estimeres nulmodellen uden ekstra forklaringsvariable.  
Den fungerer som reference for vurdering af alle efterfølgende feature-udvidelser.

```
# NULMODEL
m0 <- run_feature_test(
  df = analysis_df_d7,
  base_vars = base_vars,
  add_var = NULL,
  label = "MODEL_D7_00 - NULMODEL"
)
```

```
=====
MODEL_D7_00 - NULMODEL - SUMMARY
=====
```

```
Call:
lm(formula = f_base, data = d)
```

```
Residuals:
    Min       1Q   Median       3Q      Max
-878.70 -204.75   -5.74  174.40  778.18
```

```
Coefficients:
              Estimate Std. Error t value      Pr(>|t|)
(Intercept)  -145.72507   188.21321  -0.774    0.4396
sæson2001/2002    69.15803   101.64120   0.680    0.4969
sæson2002/2003   132.51099   105.17731   1.260    0.2090
sæson2003/2004    90.73525   106.13276   0.855    0.3935
sæson2004/2005   119.03801   108.23047   1.100    0.2725
sæson2005/2006   125.28475   108.37695   1.156    0.2489
sæson2006/2007   119.31489   104.24005   1.145    0.2535
sæson2007/2008    86.59516   107.24416   0.807    0.4202
sæson2013/2014    58.42544   110.84109   0.527    0.5986
sæson2015/2016   227.50792   106.05630   2.145    0.0330 *
sæson2016/2017    40.36120   102.66112   0.393    0.6946
sæson2021/2022    61.76442   108.38053   0.570    0.5693
sæson2022/2023   136.54916   115.19580   1.185    0.2371
sæson2023/2024   274.02395   113.68295   2.410    0.0167 *
sæson2024/2025   101.94022   115.02136   0.886    0.3764
```

```
sæson2025/2026 149.30082 132.21637 1.129 0.2600
runde 1.10635 2.02787 0.546 0.5859
kamp_time_h 6.84471 11.18132 0.612 0.5410
billetter_d7 1.75300 0.03064 57.216 <0.0000000000000002 ***
vækst_d10_d7 -0.67809 0.06708 -10.109 <0.0000000000000002 ***
---
```

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 295.3 on 234 degrees of freedom

Multiple R-squared: 0.9767, Adjusted R-squared: 0.9748

F-statistic: 516.2 on 19 and 234 DF, p-value: < 0.00000000000000022

AIC:

[1] 3631.505

Nulmodellen viser, at billetter\_d7 er den klart vigtigste forklaringsvariabel og har en stærk, signifikant positiv effekt på tilskuertallet, mens vækst\_d10\_d7 er signifikant med negativ koefficient og indikerer aftagende margineffekt tæt på kampdatoen. Modellens AIC på 3631,5 anvendes som reference for vurdering af, om efterfølgende feature-udvidelser forbedrer modelkvaliteten.

#### 4.2.18.1 Feature-test: Helligdag

Denne test vurderer, om kampe på helligdage har en selvstændig effekt på tilskuertallet. Effekten sammenlignes direkte med nulmodellen.

```
# + Helligdag
run_feature_test(
  df = analysis_df_d7,
  base_vars = base_vars,
  add_var = "er_helligdag",
  label = "MODEL_D7_01 - + HELLIGDAG",
  force_int = c("er_helligdag")
)
```

```
=====
MODEL_D7_01 - + HELLIGDAG - SUMMARY
=====
```

Call:

```
lm(formula = f_full, data = d)
```

Residuals:

	Min	1Q	Median	3Q	Max
	-879.17	-209.59	-7.59	175.32	774.41

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	-143.13614	188.62572	-0.759	0.4487
sæson2001/2002	69.19688	101.81571	0.680	0.4974
sæson2002/2003	133.22694	105.37009	1.264	0.2074

sæson2003/2004	89.05990	106.38136	0.837	0.4034
sæson2004/2005	119.92299	108.43444	1.106	0.2699
sæson2005/2006	126.10253	108.57849	1.161	0.2467
sæson2006/2007	122.12171	104.60866	1.167	0.2442
sæson2007/2008	89.86613	107.67857	0.835	0.4048
sæson2013/2014	57.46793	111.05213	0.517	0.6053
sæson2015/2016	228.48617	106.26101	2.150	0.0326 *
sæson2016/2017	42.80115	102.98291	0.416	0.6781
sæson2021/2022	63.13494	108.61010	0.581	0.5616
sæson2022/2023	140.16940	115.67898	1.212	0.2269
sæson2023/2024	275.67640	113.93842	2.420	0.0163 *
sæson2024/2025	103.45749	115.26907	0.898	0.3704
sæson2025/2026	150.19398	132.45848	1.134	0.2580
runde	1.37447	2.11853	0.649	0.5171
kamp_time_h	6.56693	11.21783	0.585	0.5588
billetter_d7	1.75243	0.03072	57.050	<0.0000000000000002 ***
vækst_d10_d7	-0.67778	0.06720	-10.086	<0.0000000000000002 ***
er_helligdag	-35.09130	78.72143	-0.446	0.6562

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 295.8 on 233 degrees of freedom  
Multiple R-squared: 0.9767, Adjusted R-squared: 0.9747  
F-statistic: 488.7 on 20 and 233 DF, p-value: < 0.00000000000000022

=====  
ANOVA: BASE vs + er\_helligdag  
=====

Analysis of Variance Table

Model 1: tilskuere ~ sæson + runde + kamp\_time\_h + billetter\_d7 + vækst\_d10\_d7  
Model 2: tilskuere ~ sæson + runde + kamp\_time\_h + billetter\_d7 + vækst\_d10\_d7 +  
er\_helligdag

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	234	20406574				
2	233	20389185	1	17388	0.1987	0.6562

=====  
AIC-SAMMENLIGNING  
=====

	df	AIC
m_base	21	3631.505
m_full	22	3633.289

Helligdagsvariablen har ingen signifikant effekt på tilskuertallet ( $p = 0,66$ ), og ANOVA-testen viser, at modellen ikke forbedres ved inkludering af er\_helligdag. AIC forværres i forhold til nulmodellen, hvilket indikerer, at variablen ikke bidrager med yderligere forklaringskraft og derfor forkastes i den videre modellering.

#### 4.2.18.2 Feature-test: Samtidig håndboldkamp (SAH)

Denne test undersøger, om samtidige håndboldkampe påvirker tilskuertallet til fodboldkampene. Resultatet vurderes relativt til nulmodellen.

```
# + Håndbold SAH
run_feature_test(
  df = analysis_df_d7,
  base_vars = base_vars,
  add_var = "er_håndboldkamp_SAH",
  label = "MODEL_D7_02 - + HÅNDBOLD SAH",
  force_int = c("er_håndboldkamp_SAH")
)
```

```
=====
MODEL_D7_02 - + HÅNDBOLD SAH - SUMMARY
=====
```

Call:  
lm(formula = f\_full, data = d)

Residuals:

	Min	1Q	Median	3Q	Max
	-876.95	-205.36	-6.79	177.04	779.57

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	-149.71093	187.52010	-0.798	0.4255
sæson2001/2002	67.95585	101.26122	0.671	0.5028
sæson2002/2003	133.52730	104.78322	1.274	0.2038
sæson2003/2004	94.00725	105.75157	0.889	0.3749
sæson2004/2005	123.39301	107.85485	1.144	0.2538
sæson2005/2006	128.18539	107.98310	1.187	0.2364
sæson2006/2007	122.09150	103.86110	1.176	0.2410
sæson2007/2008	87.22644	106.84119	0.816	0.4151
sæson2013/2014	62.81449	110.45538	0.569	0.5701
sæson2015/2016	229.59862	105.66459	2.173	0.0308 *
sæson2016/2017	42.02129	102.27959	0.411	0.6816
sæson2021/2022	63.64838	107.97854	0.589	0.5561
sæson2022/2023	143.43022	114.83664	1.249	0.2129
sæson2023/2024	279.28897	113.29923	2.465	0.0144 *
sæson2024/2025	76.44529	115.60729	0.661	0.5091
sæson2025/2026	154.23983	131.75215	1.171	0.2429
runde	1.27898	2.02290	0.632	0.5278
kamp_time_h	7.23039	11.14164	0.649	0.5170
billetter_d7	1.74741	0.03071	56.905	<0.0000000000000002 ***
vækst_d10_d7	-0.66796	0.06710	-9.954	<0.0000000000000002 ***
er_håndboldkamp_SAH	509.48622	306.04108	1.665	0.0973 .

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 294.2 on 233 degrees of freedom  
Multiple R-squared: 0.977, Adjusted R-squared: 0.975  
F-statistic: 494.2 on 20 and 233 DF, p-value: < 0.00000000000000022

```
=====
ANOVA: BASE vs + er_håndboldkamp_SAH
=====
Analysis of Variance Table

Model 1: tilskuere ~ sæson + runde + kamp_time_h + billetter_d7 + vækst_d10_d7
Model 2: tilskuere ~ sæson + runde + kamp_time_h + billetter_d7 + vækst_d10_d7 +
  er_håndboldkamp_SAH
   Res.Df    RSS Df Sum of Sq    F Pr(>F)
1      234 20406574
2      233 20166699  1    239875 2.7714 0.0973 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
=====
AIC-SAMMENLIGNING
=====
      df      AIC
m_base 21 3631.505
m_full 22 3630.502
```

Variablen `er_håndboldkamp_SAH` har en positiv koefficient, men er kun svagt signifikant på 10 %-niveauet ( $p = 0,097$ ) og er ikke robust ved 5 %-niveauet. ANOVA-testen viser ingen klar forbedring i forhold til nulmodellen, og AIC forbedres ikke tilstrækkeligt til at retfærdiggøre variabelen. På den baggrund vurderes effekten som usikker, og variabelen fravælges i den endelige model.

#### 4.2.18.3 Feature-test: Temperatur (fair sammenligning)

Her testes temperaturens betydning på et fælles datasæt uden manglende observationer. Det sikrer, at effekten vurderes uafhængigt af datatab.

```
# + Temperatur (fair test: common dataset, som du gjorde)
# + Temperatur (fair test: common dataset, som du gjorde)
df_temp_common <- analysis_df_d7 |>
  select(all_of(c(base_vars, "temperatur_model"))) |>
  drop_na()

run_feature_test(
  df = df_temp_common,
  base_vars = base_vars,
  add_var = "temperatur_model",
  label = "MODEL_D7_03 - + TEMPERATUR (common)"
)
```

```
=====
MODEL_D7_03 - + TEMPERATUR (common) - SUMMARY
=====
```

Call:

```
lm(formula = f_full, data = d)
```

Residuals:

	Min	1Q	Median	3Q	Max
	-880.57	-204.58	-9.56	172.10	769.61

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	-148.06086	197.88654	-0.748	0.4551
sæson2001/2002	68.38140	101.59846	0.673	0.5016
sæson2002/2003	135.38314	105.28985	1.286	0.1998
sæson2003/2004	93.04758	106.10699	0.877	0.3814
sæson2004/2005	121.40444	108.19067	1.122	0.2630
sæson2005/2006	127.52859	108.58728	1.174	0.2414
sæson2006/2007	121.61618	104.24065	1.167	0.2445
sæson2007/2008	87.54040	107.20891	0.817	0.4150
sæson2013/2014	60.90534	110.80818	0.550	0.5831
sæson2015/2016	229.22597	106.01121	2.162	0.0316 *
sæson2016/2017	40.54791	102.61995	0.395	0.6931
sæson2021/2022	62.31581	108.39614	0.575	0.5659
sæson2022/2023	141.02103	115.21289	1.224	0.2222
sæson2023/2024	276.55704	113.75918	2.431	0.0158 *
sæson2024/2025	104.94014	115.14268	0.911	0.3630
sæson2025/2026	100.86416	136.06422	0.741	0.4593
runde	0.85432	2.16088	0.395	0.6929
kamp_time_h	7.81810	11.24003	0.696	0.4874
billetter_d7	1.75243	0.03098	56.559	<0.0000000000000002 ***
vækst_d10_d7	-0.68150	0.06718	-10.145	<0.0000000000000002 ***
temperatur_model	-0.49595	3.34796	-0.148	0.8824

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 295.2 on 232 degrees of freedom

Multiple R-squared: 0.9767, Adjusted R-squared: 0.9747

F-statistic: 486 on 20 and 232 DF, p-value: < 0.00000000000000022

=====

ANOVA: BASE vs + temperatur\_model

=====

Analysis of Variance Table

Model 1: tilskuere ~ sæson + runde + kamp\_time\_h + billetter\_d7 + vækst\_d10\_d7

Model 2: tilskuere ~ sæson + runde + kamp\_time\_h + billetter\_d7 + vækst\_d10\_d7 +  
temperatur\_model

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	233	20213771				
2	232	20211859	1	1911.8	0.0219	0.8824

=====

AIC-SAMMENLIGNING

=====

	df	AIC
m_base	21	3615.970
m_full	22	3617.946

Temperaturvariablen har ingen signifikant effekt på tilskuertallet ( $p = 0,88$ ), og ANOVA-testen viser, at modellen ikke forbedres ved at inkludere temperatur\_model på et fælles datasæt. AIC forbedres ikke i forhold til baselinemodellen, hvilket indikerer, at temperatur ikke bidrager med yderligere forklaringskraft i D7-setup og derfor fravælges i den videre modellering.

```
# + vejr_mangler_info
run_feature_test(
  df = analysis_df_d7,
  base_vars = base_vars,
  add_var = "vejr_mangler_info",
  label = "MODEL_D7_04 - + VEJR_MANGLER_INFO",
  force_int = c("vejr_mangler_info")
)
```

```
=====
MODEL_D7_04 - + VEJR_MANGLER_INFO - SUMMARY
=====
```

```
Call:
lm(formula = f_full, data = d)
```

```
Residuals:
    Min       1Q   Median       3Q      Max
-872.12 -204.35  -12.59   169.48   784.68
```

```
Coefficients:
              Estimate Std. Error t value      Pr(>|t|)
(Intercept)   -176.05427    192.36420   -0.915     0.3610
sæson2001/2002    73.97928    101.91565    0.726     0.4686
sæson2002/2003   132.26206    105.26663    1.256     0.2102
sæson2003/2004    83.58825    106.61895    0.784     0.4338
sæson2004/2005   115.50027    108.41730    1.065     0.2878
sæson2005/2006   133.01559    108.92275    1.221     0.2232
sæson2006/2007   118.71414    104.33095    1.138     0.2563
sæson2007/2008    87.09496    107.33667    0.811     0.4180
sæson2013/2014    63.90755    111.15830    0.575     0.5659
sæson2015/2016   239.03926    107.17583    2.230     0.0267 *
sæson2016/2017    52.40581    103.90777    0.504     0.6145
sæson2021/2022    66.31190    108.62946    0.610     0.5422
sæson2022/2023   147.24748    116.11035    1.268     0.2060
sæson2023/2024   283.29538    114.40143    2.476     0.0140 *
sæson2024/2025   107.82481    115.36676    0.935     0.3509
sæson2025/2026   156.02604    132.61012    1.177     0.2406
runde           1.09438      2.02964    0.539     0.5903
kamp_time_h      7.63524     11.23680    0.679     0.4975
billetter_d7      1.75181      0.03070   57.058 <0.0000000000000002 ***
vækst_d10_d7     -0.67378      0.06736  -10.002 <0.0000000000000002 ***
```

```

vejr_mangler_info    31.63024    40.65836    0.778                0.4374
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

```

Residual standard error: 295.6 on 233 degrees of freedom
Multiple R-squared:  0.9768,    Adjusted R-squared:  0.9748
F-statistic: 489.6 on 20 and 233 DF,  p-value: < 0.00000000000000022

```

```

=====
ANOVA: BASE vs + vejr_mangler_info
=====
Analysis of Variance Table

```

```

Model 1: tilskuere ~ sæson + runde + kamp_time_h + billetter_d7 + vækst_d10_d7
Model 2: tilskuere ~ sæson + runde + kamp_time_h + billetter_d7 + vækst_d10_d7 +
      vejr_mangler_info
      Res.Df      RSS Df Sum of Sq      F Pr(>F)
1      234 20406574
2      233 20353706  1      52868 0.6052 0.4374

```

```

=====
AIC-SAMMENLIGNING
=====
      df      AIC
m_base 21 3631.505
m_full 22 3632.846

```

Variablen `vejr_mangler_info` har ingen signifikant effekt på tilskuertallet ( $p = 0,44$ ), og ANOVA-testen viser, at modellen ikke forbedres ved at inkludere indikatoren for manglende vejrinformation. AIC forbedres ikke i forhold til baselinemodellen, hvilket indikerer, at fravær af vejrdato ikke har systematisk betydning for tilskuertallet i D7-setup og derfor ikke bidrager med yderligere forklaringskraft.

#### 4.2.18.4 Feature-test: Placering før kamp

Denne sektion tilføjer og tester VFF's placering før kamp som forklaringsvariabel. Placeringerne renses og reduceres til én entydig observation pr. kamp, hvorefter effekten vurderes på et komplet datasæt for at sikre en fair sammenligning med baseline-modellen.

```

cat("\n--- Henter VFF rundeplaceringer (CLEAN) ---\n")

--- Henter VFF rundeplaceringer (CLEAN) ---

plac_raw <- safe_dbReadTable(con, "PBA02_Clean", "fact_vff_rundeplaceringer_clean") |>
  mutate(kamp_id = as.character(kamp_id))

need_cols <- c("kamp_id", "placering_før_kamp")
missing <- setdiff(need_cols, names(plac_raw))
if (length(missing) > 0) {

```

```

cat(" Mangler kolonner i placeringstabellen: ", paste(missing, collapse = ", "), "\n")
cat("Kolonner i tabellen er:\n")
print(names(plac_raw))
stop("Stopper så du kan se mismatch.")
}

plac_1row <- plac_raw |>
  select(kamp_id, placering_før_kamp) |>
  mutate(
    kamp_id = as.character(kamp_id),
    placering_før_kamp = suppressWarnings(as.integer(placering_før_kamp))
  ) |>
  arrange(kamp_id) |>
  group_by(kamp_id) |>
  slice(1) |>
  ungroup()

stopifnot(sum(duplicated(plac_1row$kamp_id)) == 0)

cat("Placeringstabel klar   Rækker:", nrow(plac_1row),
    "Unikke kamp_id:", n_distinct(plac_1row$kamp_id), "\n\n")

```

Placeringstabel klar Rækker: 545 Unikke kamp\_id: 545

```

# Byg datasæt til placeringstest (kun komplette rækker)
df_pos <- analysis_df_d7 |>
  mutate(kamp_id = as.character(kamp_id)) |>
  left_join(plac_1row, by = "kamp_id") |>
  transmute(
    tilskuere = as.numeric(tilskuere),
    sæson = as.factor(sæson),
    runde = as.integer(runde),
    kamp_time_h = as.integer(kamp_time_h),
    billetter_d7 = as.numeric(billetter_d7),
    vækst_d10_d7 = as.numeric(vækst_d10_d7),
    placering_før_kamp = as.integer(placering_før_kamp)
  ) |>
  filter(if_all(everything(), ~ !is.na(.)))

cat("df_pos klar   Rækker:", nrow(df_pos), "\n")

```

df\_pos klar Rækker: 249

```

cat("Placering NA efter join (før filter):",
    sum(is.na((analysis_df_d7 |>
      mutate(kamp_id = as.character(kamp_id)) |>
      left_join(plac_1row, by = "kamp_id"))$placering_før_kamp)),
    "\n\n")

```

Placering NA efter join (før filter): 5

```
# Kjør feature-test på placering
base_vars_pos <- c("tilskuere","sæson","runde","kamp_time_h","billetter_d7","vækst_d10_d7")

run_feature_test(
  df = df_pos,
  base_vars = base_vars_pos,
  add_var = "placering_før_kamp",
  label = "MODEL_D7_05 - + PLACERING_FØR_KAMP",
  force_int = c("runde","kamp_time_h","placering_før_kamp")
)
```

```
=====
MODEL_D7_05 - + PLACERING_FØR_KAMP - SUMMARY
=====
```

```
Call:
lm(formula = f_full, data = d)
```

```
Residuals:
    Min       1Q   Median       3Q      Max
-914.63 -196.82   -6.51  169.64  765.14
```

```
Coefficients:
                Estimate Std. Error t value      Pr(>|t|)
(Intercept)    -146.47500   197.69303   -0.741     0.4595
sæson2001/2002     63.84861   108.57253    0.588     0.5571
sæson2002/2003    122.28618   128.09690    0.955     0.3408
sæson2003/2004     87.83508   111.82302    0.785     0.4330
sæson2004/2005    118.44549   109.70989    1.080     0.2815
sæson2005/2006    127.76961   111.18773    1.149     0.2517
sæson2006/2007    110.99937   123.74689    0.897     0.3707
sæson2007/2008     74.46646   134.78401    0.552     0.5812
sæson2013/2014     70.41386   124.29747    0.566     0.5716
sæson2015/2016    219.26961   121.40144    1.806     0.0722 .
sæson2016/2017     43.47876   136.82698    0.318     0.7510
sæson2021/2022     55.17652   115.00007    0.480     0.6318
sæson2022/2023    156.37712   118.45953    1.320     0.1881
sæson2023/2024    267.69747   123.12157    2.174     0.0307 *
sæson2024/2025    121.01109   124.94699    0.968     0.3338
sæson2025/2026    143.50986   143.84452    0.998     0.3195
runde             0.17194     2.09710    0.082     0.9347
kamp_time_h       7.43874     11.38052    0.654     0.5140
billetter_d7      1.75047     0.03092   56.604 <0.0000000000000002 ***
vækst_d10_d7     -0.66893     0.06817   -9.813 <0.0000000000000002 ***
placering_før_kamp 1.54536    12.68683    0.122     0.9032
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 296.7 on 228 degrees of freedom
Multiple R-squared:  0.9766,    Adjusted R-squared:  0.9746
F-statistic: 476.4 on 20 and 228 DF,  p-value: < 0.00000000000000022
```

```
=====
```

```
ANOVA: BASE vs + placering_før_kamp
```

```
=====
```

```
Analysis of Variance Table
```

```
Model 1: tilskuere ~ sæson + runde + kamp_time_h + billetter_d7 + vækst_d10_d7
```

```
Model 2: tilskuere ~ sæson + runde + kamp_time_h + billetter_d7 + vækst_d10_d7 +
```

```
  placering_før_kamp
```

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	229	20070668				
2	228	20069362	1	1306	0.0148	0.9032

```
=====
```

```
AIC-SAMMENLIGNING
```

```
=====
```

	df	AIC
m_base	21	3561.663
m_full	22	3563.647

Placering\_før\_kamp har ingen signifikant effekt på tilskuertallet ( $p = 0,90$ ), og ANOVA-testen viser, at modellen ikke forbedres ved inkludering af variabelen. AIC ændres ikke i forhold til baselinemodellen, hvilket indikerer, at ligaplaceringen før kamp ikke tilfører selvstændig forklaringskraft, når der allerede kontrolleres for billetsalg tæt på kampdatoen. Resultatet peger på, at information om holdets sportslige position i høj grad allerede er indlejret i billetsalgsvariablerne, og at placering derfor ikke bidrager yderligere i D7-setup.

#### 4.2.18.5 Feature-test: Befolkning (fair test og omkostning)

Denne sektion tester befolkningstal som forklaringsvariabel ved at sammenligne modeller med og uden befolkning på et fælles datasæt.

Dermed vurderes både den statistiske effekt og omkostningen i form af reduceret datagrundlag, før variabelen eventuelt inkluderes.

```
# =====
# 12) TEST: Befolkning (fair test + omkostning) - FIX
# =====
set.seed(42)
test_prop <- 0.20

# Baseline datasæt (uden befolkning) - kun komplette rækker
df_d7_full <- analysis_df_d7 |>
  transmute(
    tilskuere = as.numeric(tilskuere),
    sæson     = as.factor(sæson),
    runde     = as.integer(runde),
    kamp_time_h = as.integer(kamp_time_h),
    billetter_d7 = as.numeric(billetter_d7),
    vækst_d10_d7 = as.numeric(vækst_d10_d7)
  ) |>
  filter(if_all(everything(), ~ !is.na(.)))
```

```
cat("df_d7_full (uden befolkning)    Rækker:", nrow(df_d7_full), "\n")
```

```
df_d7_full (uden befolkning)    Rækker: 254
```

```
# Fair test: fælles datasæt (samme rækker) når vi sammenligner med befolkning
df_d7_bef_common <- analysis_df_d7 |>
  transmute(
    tilskuere      = as.numeric(tilskuere),
    sæson          = as.factor(sæson),
    runde          = as.integer(runde),
    kamp_time_h    = as.integer(kamp_time_h),
    billetter_d7   = as.numeric(billetter_d7),
    vækst_d10_d7   = as.numeric(vækst_d10_d7),
    befolkningstal = as.numeric(befolkningstal)
  ) |>
  filter(if_all(everything(), ~ !is.na(.)))

cat("df_d7_bef_common (med befolkning)    Rækker:", nrow(df_d7_bef_common),
    " (tabt pga NA befolkning:", nrow(df_d7_full) - nrow(df_d7_bef_common), ")\n\n")
```

```
df_d7_bef_common (med befolkning)    Rækker: 131 (tabt pga NA befolkning: 123 )
```

```
# Modeltest: uden befolkning vs + befolkning (samme rækker)
base_vars_bef <- c("tilskuere","sæson","runde","kamp_time_h","billetter_d7","vækst_d10_d7")

run_feature_test(
  df = df_d7_bef_common,
  base_vars = base_vars_bef,
  add_var = NULL,
  label = "MODEL_D7_06A - BASE (common for befolkning)"
)
```

```
=====
MODEL_D7_06A - BASE (common for befolkning) - SUMMARY
=====
```

```
Call:
lm(formula = f_base, data = d)
```

```
Residuals:
    Min       1Q   Median       3Q      Max
-845.90 -242.31  -14.89   241.81   777.69
```

```
Coefficients:
              Estimate Std. Error t value    Pr(>|t|)
(Intercept)   -3.33303   299.23002  -0.011    0.9911
sæson2013/2014  23.60700   148.80533   0.159    0.8742
sæson2015/2016 167.67046   145.47288   1.153    0.2514
sæson2016/2017 -29.70316   141.72498  -0.210    0.8344
```

```

sæson2021/2022    9.77701   144.53148   0.068                0.9462
sæson2022/2023  119.54210   158.85726   0.753                0.4532
sæson2023/2024  252.59901   152.26306   1.659                0.0998 .
sæson2024/2025   85.36711   155.33227   0.550                0.5836
sæson2025/2026  133.79835   175.50093   0.762                0.4474
runde            2.67987    3.30155   0.812                0.4186
kamp_time_h      6.14680    14.64865   0.420                0.6755
billetter_d7     1.71398    0.04678   36.642 < 0.0000000000000002 ***
vækst_d10_d7    -0.66263    0.09663   -6.858                0.000000000342 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

```

Residual standard error: 325.3 on 118 degrees of freedom
Multiple R-squared:  0.9658,    Adjusted R-squared:  0.9623
F-statistic: 277.3 on 12 and 118 DF,  p-value: < 0.00000000000000022

```

```

AIC:
[1] 1901.701

```

```

run_feature_test(
  df = df_d7_bef_common,
  base_vars = base_vars_bef,
  add_var = "befolkningstal",
  label = "MODEL_D7_06B - + BEFOLKNING (common)"
)

```

```

=====
MODEL_D7_06B - + BEFOLKNING (common) - SUMMARY
=====

```

```

Call:
lm(formula = f_full, data = d)

```

```

Residuals:
    Min       1Q   Median       3Q      Max
-849.85 -233.06   -9.83  236.62  819.68

```

```

Coefficients:
            Estimate Std. Error t value      Pr(>|t|)
(Intercept)  31018.57437  27592.19451   1.124      0.263
sæson2013/2014   833.78668   735.73674   1.133      0.259
sæson2015/2016  1407.41141  1112.14706   1.265      0.208
sæson2016/2017  1402.88633  1281.97132   1.094      0.276
sæson2021/2022  1586.52547  1409.75616   1.125      0.263
sæson2022/2023  1954.19171  1639.41649   1.192      0.236
sæson2023/2024  2068.59889  1622.27750   1.275      0.205
sæson2024/2025  1954.10202  1669.26090   1.171      0.244
sæson2025/2026  2064.92969  1726.45232   1.196      0.234
runde          5.82505     4.32444   1.347      0.181
kamp_time_h     6.78081    14.64314   0.463      0.644

```

```

billetter_d7      1.72328      0.04745  36.317 < 0.0000000000000002 ***
vækst_d10_d7     -0.66785      0.09663  -6.911      0.000000000269 ***
befolkningstal   -0.33766      0.30031  -1.124      0.263

```

```
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 325 on 117 degrees of freedom
```

```
Multiple R-squared:  0.9661,    Adjusted R-squared:  0.9624
```

```
F-statistic: 256.6 on 13 and 117 DF,  p-value: < 0.00000000000000022
```

```
=====
```

```
ANOVA: BASE vs + befolkningstal
```

```
=====
```

```
Analysis of Variance Table
```

```
Model 1: tilskuere ~ sæson + runde + kamp_time_h + billetter_d7 + vækst_d10_d7
```

```
Model 2: tilskuere ~ sæson + runde + kamp_time_h + billetter_d7 + vækst_d10_d7 +
```

```
    befolkningstal
```

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	118	12489251				
2	117	12355746	1	133505	1.2642	0.2632

```
=====
```

```
AIC-SAMMENLIGNING
```

```
=====
```

	df	AIC
m_base	14	1901.701
m_full	15	1902.293

```

# (Valgfrit) split til senere out-of-sample test (samme common datasæt)
split_obj <- rsample::initial_split(df_d7_bef_common, prop = 1 - test_prop)
train_df  <- rsample::training(split_obj)
test_df   <- rsample::testing(split_obj)

cat("Train rækker:", nrow(train_df), " Test rækker:", nrow(test_df), "\n")

```

```
Train rækker: 104  Test rækker: 27
```

Befolkningstallet testes på et fælles datasæt for at sikre en fair sammenligning, men dette medfører et markant datatab, idet antallet af observationer reduceres fra 254 til 131 på grund af manglende befolkningsdata. På dette reducerede datasæt viser befolkningstal ingen signifikant effekt på tilskuertallet ( $p = 0,26$ ), og ANOVA-testen indikerer, at modellen ikke forbedres ved at inkludere variabelen. AIC forbedres ikke i forhold til baselinemodellen, og den marginale ændring i forklaringsgrad er ubetydelig. Samlet set vurderes omkostningen i form af halveret datagrundlag at overstige den potentielle gevinst, hvorfor befolkningstal fravælges i den endelige D7-model.

#### 4.2.18.6 Samlet oversigt over variabeltests

Denne tabel sammenfatter resultaterne af alle gennemførte feature-tests i D7-analysen. Den giver et samlet beslutningsgrundlag for, hvilke variable der fastholdes i den endelige model, baseret på statistisk evidens, AIC og datamæssige omkostninger.

```

# =====
# 13) Samlet variabeltest-oversigt (uændret)
# =====
variabeltest_oversigt_d7 <- tibble::tibble(
  variabel = c(
    "placering_før_kamp",
    "billetter_d7",
    "vækst_d10_d7",
    "sæson",
    "runde",
    "kamp_time_h",
    "er_helligdag",
    "er_håndboldkamp_SAH",
    "temperatur_model",
    "vejr_mangler_info",
    "befolkningstal"
  ),
  n_test = c(
    249,
    254, 254,
    254, 254, 254,
    254, 254,
    253, 254,
    131
  ),
  p_koefficient = c(
    0.9032,
    NA, NA,
    NA, NA, NA,
    0.656,
    0.097,
    0.8824,
    0.437,
    0.577
  ),
  p_anova = c(
    0.9032,
    NA, NA,
    NA, NA, NA,
    0.656,
    0.097,
    0.8824,
    0.437,
    0.577
  ),
  delta_AIC = c(
    +1.984,
    NA, NA,
    NA, NA, NA,
    +1.62,
    -1.02,

```

```

+1.88,
+1.10,
+1.64
),
beslutning = c(
  "Forkastet",
  "Fast baseline-feature",
  "Fast baseline-feature",
  "Fast baseline-feature",
  "Fast baseline-feature",
  "Fast baseline-feature",
  "Forkastet",
  "Forkastet",
  "Forkastet",
  "Forkastet",
  "Forkastet (cost > benefit)"
),
begrundelse = c(
  "Ingen signifikant effekt (p 0.90) og forværrer AIC",
  "Primær forklaringsvariabel 7 dage før kamp",
  "Stærk vækstindikator tæt på kamp",
  "Kontrollerer for strukturelle sæsonforskelle",
  "Kampkontekst",
  "Tidsmæssig efterspørgselseffekt",
  "Ingen signifikant effekt (p>0.6)",
  "Svag evidens (10%-niveau), ikke robust",
  "Ingen effekt og forværrer AIC",
  "Manglende vejrinfor er ikke systematisk",
  "Ingen signifikant effekt og halverer datasættet"
)
)
print(variabeltest_oversigt_d7)

```

# A tibble: 11 x 7

	variabel <chr>	n_test <dbl>	p_koefficient <dbl>	p_anova <dbl>	delta_AIC <dbl>	beslutning <chr>	begrundelse <chr>
1	placering_før_~	249	0.903	0.903	1.98	Forkastet	Ingen sign~
2	billetter_d7	254	NA	NA	NA	Fast base~	Primær for~
3	vækst_d10_d7	254	NA	NA	NA	Fast base~	Stærk væks~
4	sæson	254	NA	NA	NA	Fast base~	Kontroller~
5	runde	254	NA	NA	NA	Fast base~	Kampkontek~
6	kamp_time_h	254	NA	NA	NA	Fast base~	Tidsmæssig~
7	er_helligdag	254	0.656	0.656	1.62	Forkastet	Ingen sign~
8	er_håndboldkam~	254	0.097	0.097	-1.02	Forkastet	Svag evide~
9	temperatur_mod~	253	0.882	0.882	1.88	Forkastet	Ingen effe~
10	vejr_mangler_i~	254	0.437	0.437	1.1	Forkastet	Manglende ~
11	befolkningstal	131	0.577	0.577	1.64	Forkastet~	Ingen sign~

#### 4.2.18.7 Endelig D7-model: tidssplit og modelsammenligning

Denne sektion bygger den endelige D7-model ved hjælp af et tidssplit, hvor tidlige kampe anvendes til træning og de seneste til test. OLS-, Ridge- og Lasso-modeller sammenlignes på identiske features, og resultaterne opsummeres via performance-mål, standardplots og artefakter til videre brug i rapport og produktlag.

```
stopifnot(all(c(
  "kamp_id", "kamp_dato", "tilskuere", "sæson", "runde", "kamp_time_h", "billetter_d7", "vækst_d10_d7"
) %in% names(analysis_df_d7)))
```

#### 4.2.18.8 Hjælpere til ligningsformidling (OLS)

Disse funktioner formatterer modelkoefficienter til læsbare ligninger og sørger for pæn linjeombrydning. Formålet er at gøre OLS-modellens resultat let forståeligt i plots og rapportering.

```
wrap_text <- function(s, width = 85) {
  paste(strwrap(as.character(s), width = width), collapse = "\n")
}

build_lm_equation_text <- function(lm_model, y_name = "tilskuere", yhat_name = "ŷ") {
  cf <- stats::coef(lm_model)
  cf <- cf[!is.na(cf)]
  nm <- names(cf)

  intercept <- unname(cf[1])
  terms <- nm[-1]
  betas <- unname(cf[-1])

  # pæn formatering (3 decimaler)
  fmt <- function(x) formatC(x, format = "f", digits = 3)

  rhs <- paste0(fmt(intercept))
  if (length(terms) > 0) {
    for (i in seq_along(terms)) {
      b <- betas[i]
      sign_txt <- if (b >= 0) " + " else " - "
      rhs <- paste0(rhs, sign_txt, fmt(abs(b)), ".", terms[i])
    }
  }

  paste0(yhat_name, " = ", rhs, "      (OLS)")
}
```

#### 4.2.18.9 Hjælpere til ligningsformidling (Ridge/Lasso)

Disse funktioner omsætter glmnet-modeller til læsbare ligninger baseret på `lambda.min` og indsætter dem direkte i plots. Formålet er at sikre konsistent og sammenlignelig formidling af Ridge- og Lasso-resultater.

```
build_glmnet_equation_text <- function(cv_model, x_names, model_label = "GLMNET", yhat_name = "ŷ") {
  # coef() returnerer en dgMatrix med (Intercept) + koef for hver feature
  b <- as.matrix(stats::coef(cv_model, s = "lambda.min"))
  b <- as.numeric(b)
  names(b) <- rownames(as.matrix(stats::coef(cv_model, s = "lambda.min")))
}
```

```

# intercept
intercept <- b[1]
beta <- b[-1]
names(beta) <- names(b)[-1]

# sørg for at rækkefølge matcher X-kolonner (model.matrix)
beta <- beta[x_names]

fmt <- function(x) formatC(x, format = "f", digits = 3)

rhs <- paste0(fmt(intercept))
if (length(beta) > 0) {
  for (i in seq_along(beta)) {
    bi <- beta[i]
    sign_txt <- if (bi >= 0) " + " else " - "
    rhs <- paste0(rhs, sign_txt, fmt(abs(bi)), ".", names(beta)[i])
  }
}

paste0(yhat_name, " = ", rhs, "      (", model_label, ",      )")
}

add_eq_to_plot <- function(p, eq_text, size = 3) {
  p + ggplot2::annotate(
    "text",
    x = -Inf, y = Inf,
    label = wrap_text(eq_text, width = 85),
    hjust = -0.05, vjust = 1.1,
    size = size
  )
}

```

```

# Klargør modelling-datasæt (én række pr kamp) ---
df_d7_final <- analysis_df_d7 |>
  dplyr::transmute(
    kamp_id      = as.character(kamp_id),
    kamp_dato    = as.Date(kamp_dato),
    tilskuere    = as.numeric(tilskuere),
    sæson_chr    = as.character(sæson),
    sæson_start  = season_start_year(sæson_chr),
    runde        = as.integer(runde),
    kamp_time_h  = as.integer(kamp_time_h),
    billetter_d7 = as.numeric(billetter_d7),
    vækst_d10_d7 = as.numeric(vækst_d10_d7)
  ) |>
  dplyr::filter(dplyr::if_all(dplyr::everything(), ~ !is.na(.))) |>
  dplyr::arrange(kamp_dato)

cat("df_d7_final klar   Rækker:", nrow(df_d7_final),
    "Periode:", as.character(min(df_d7_final$kamp_dato)), "→", as.character(max(df_d7_final$kamp_dato))

```

df\_d7\_final klar      Rækker: 254 Periode: 2000-07-30 → 2025-12-07

```
cat("Antal unikke sæsoner:", dplyr::n_distinct(df_d7_final$sæson_chr), "\n\n")
```

Antal unikke sæsoner: 16

#### 4.2.18.10 Tidssplit til modelvalidering

Her opdeles datasættet tidsmæssigt, så de seneste kampe anvendes som testdata. Det sikrer en realistisk evaluering af modellernes prædiktive evne.

```
# Tidssplit (seneste 20% som test) ---
set.seed(42)
test_prop <- 0.20

n_total <- nrow(df_d7_final)
n_test  <- max(1, floor(n_total * test_prop))
cut_idx <- n_total - n_test

train_d7 <- df_d7_final[1:cut_idx, ]
test_d7  <- df_d7_final[(cut_idx + 1):n_total, ]

cat("Tidssplit klar \n")
```

Tidssplit klar

```
cat("Train:", nrow(train_d7), " Test:", nrow(test_d7), "\n")
```

Train: 204    Test: 50

```
cat("Test-periode starter:", as.character(min(test_d7$kamp_dato)), "\n\n")
```

Test-periode starter: 2022-11-06

#### 4.2.18.11 Endelig OLS-model og testperformance

Her estimeres den endelige OLS-model på træningsdata og evalueres på testdatasættet. Modelkvaliteten rapporteres ved RMSE, MAE og  $R^2$  for en samlet vurdering af præcisionen.

```
m_d7_final_lm <- lm(
  tilskuere ~ sæson_start + runde + kamp_time_h + billetter_d7 + vækst_d10_d7,
  data = train_d7
)

pred_lm <- as.numeric(predict(m_d7_final_lm, newdata = test_d7))

rmse_lm <- rmse_vec(test_d7$tilskuere, pred_lm)
mae_lm  <- mae_vec(test_d7$tilskuere, pred_lm)
r2_lm   <- r2_vec(test_d7$tilskuere, pred_lm)

cat("=====\n")
```

```
=====
```

```
cat("D7 FINAL OLS - TEST METRICS\n")
```

```
D7 FINAL OLS - TEST METRICS
```

```
cat("=====\\n")
```

```
=====
```

```
cat("RMSE:", rmse_lm, " MAE:", mae_lm, " R2:", r2_lm, "\\n\\n")
```

```
RMSE: 378.0339 MAE: 312.1547 R2: 0.8903231
```

Den endelige D7 OLS-model opnår en RMSE på cirka 378 og en MAE på cirka 312, hvilket indikerer en relativt stabil prædiktiv præcision på testdatasættet. Med en  $R^2$  på omkring 0,89 forklarer modellen størstedelen af variationen i tilskuertallet og vurderes som velegnet til operationel anvendelse syv dage før kampstart. Men vi kigger også på mere avancerede modeller.

#### 4.2.19 Designmatricer til Ridge og Lasso

Denne kode opbygger modelmatricer for trænings- og testdata, som kræves af glmnet. Det sikrer, at Ridge- og Lasso-modeller estimeres på samme features som OLS-modellen.

```
x_train <- model.matrix(
  tilskuere ~ sæson_start + runde + kamp_time_h + billetter_d7 + vækst_d10_d7,
  data = train_d7
)[, -1, drop = FALSE]
y_train <- train_d7$tilskuere

x_test <- model.matrix(
  tilskuere ~ sæson_start + runde + kamp_time_h + billetter_d7 + vækst_d10_d7,
  data = test_d7
)[, -1, drop = FALSE]
y_test <- test_d7$tilskuere

x_names <- colnames(x_train)
```

##### 4.2.19.1 Ridge-model og testperformance

Her estimeres en Ridge-model med krydsvalidering og evalueres på testdatasættet. Resultaterne rapporteres via RMSE, MAE og  $R^2$  sammen med det optimale .

```
# 14.5) Ridge (alpha = 0)

set.seed(42)
cv_ridge <- glmnet::cv.glmnet(
  x = x_train, y = y_train,
  alpha = 0,
  nfolds = 10,
  standardize = TRUE
)

lambda_ridge <- cv_ridge$lambda.min
pred_ridge <- as.numeric(predict(cv_ridge, newx = x_test, s = "lambda.min"))

rmse_ridge <- rmse_vec(y_test, pred_ridge)
mae_ridge <- mae_vec(y_test, pred_ridge)
r2_ridge <- r2_vec(y_test, pred_ridge)

cat("=====\n")
```

```
=====
```

```
cat("D7 RIDGE - TEST METRICS\n")
```

```
D7 RIDGE - TEST METRICS
```

```
cat("=====\n")
```

```
=====
```

```
cat("lambda.min:", lambda_ridge, "\n")
```

```
lambda.min: 171.6097
```

```
cat("RMSE:", rmse_ridge, " MAE:", mae_ridge, " R2:", r2_ridge, "\n\n")
```

```
RMSE: 540.3833 MAE: 423.3139 R2: 0.7758918
```

Ridge-modellen opnår en RMSE på cirka 540 og en MAE på cirka 423, hvilket er markant dårligere end OLS-modellen på testdatasættet. Med en  $R^2$  på omkring 0,78 forklarer modellen væsentligt mindre af variationen i tilskuertallet, og regulariseringen reducerer her modellens præcision uden at give en bedre generalisering. Ridge vurderes derfor ikke som konkurrencedygtig i D7-setup.

#### 4.2.19.2 Lasso-model og testperformance

Denne kode estimerer en Lasso-model med krydsvalidering og evaluerer den på testdatasættet. Resultaterne sammenlignes direkte med OLS- og Ridge-modellerne.

```
# 14.6) Lasso (alpha = 1)

set.seed(42)
cv_lasso <- glmnet::cv.glmnet(
  x = x_train, y = y_train,
  alpha = 1,
  nfolds = 10,
  standardize = TRUE
)

lambda_lasso <- cv_lasso$lambda.min
pred_lasso <- as.numeric(predict(cv_lasso, newx = x_test, s = "lambda.min"))

rmse_lasso <- rmse_vec(y_test, pred_lasso)
mae_lasso <- mae_vec(y_test, pred_lasso)
r2_lasso <- r2_vec(y_test, pred_lasso)

cat("=====\n")
```

```
=====  
cat("D7 LASSO - TEST METRICS\n")
```

```
D7 LASSO - TEST METRICS
```

```
cat("=====\n")
```

```
=====  
cat("lambda.min:", lambda_lasso, "\n")
```

```
lambda.min: 8.541053
```

```
cat("RMSE:", rmse_lasso, " MAE:", mae_lasso, " R2:", r2_lasso, "\n\n")
```

```
RMSE: 383.068 MAE: 309.679 R2: 0.8873825
```

```
# Samlet performance-tabel
```

```
perf_d7 <- tibble::tibble(
  model = c("OLS (lm)", "Ridge (cv.glmnet)", "Lasso (cv.glmnet)"),
  RMSE = c(rmse_lm, rmse_ridge, rmse_lasso),
  MAE = c(mae_lm, mae_ridge, mae_lasso),
  R2 = c(r2_lm, r2_ridge, r2_lasso)
)

cat("=====\n")
```

```
=====
```

```
cat("D7 - SAMLET TEST PERFORMANCE\n")
```

```
D7 - SAMLET TEST PERFORMANCE
```

```
cat("=====\\n")
```

```
=====
```

```
print(perf_d7)
```

```
# A tibble: 3 x 4
  model      RMSE  MAE  R2
<chr>    <dbl> <dbl> <dbl>
1 OLS (lm)    378.  312. 0.890
2 Ridge (cv.glmnet) 540.  423. 0.776
3 Lasso (cv.glmnet) 383.  310. 0.887
```

```
cat("\\n")
```

Sammenligningen viser, at OLS-modellen opnår den laveste RMSE og dermed den bedste samlede prædiktive præcision på testdatasættet. Lasso ligger meget tæt på OLS med næsten identiske MAE- og  $R^2$ -værdier, men uden en reel forbedring i performance. Ridge klarer sig markant dårligere på alle mål. På den baggrund vælges OLS som den endelige D7-model, mens Lasso primært fungerer som en robusthedskontrol, der bekræfter modellens stabilitet.

#### 4.2.20 Klargøring af plot-data og residualer

Denne kode samler faktiske værdier og forudsigelser fra alle tre modeller i ét datasæt. Datasættet anvendes til standardplots for sammenligning af prediction og residualer.

```
# Plot-data + standard plots (actual vs pred, residual plots)
```

```
plot_df <- tibble::tibble(
  kamp_dato = test_d7$kamp_dato,
  actual    = y_test,
  pred_lm   = pred_lm,
  pred_ridge = pred_ridge,
  pred_lasso = pred_lasso
) |>
dplyr::mutate(
  resid_lm = actual - pred_lm,
  resid_ridge = actual - pred_ridge,
  resid_lasso = actual - pred_lasso
)
```

#### 4.2.20.1 Ligninger for de endelige modeller

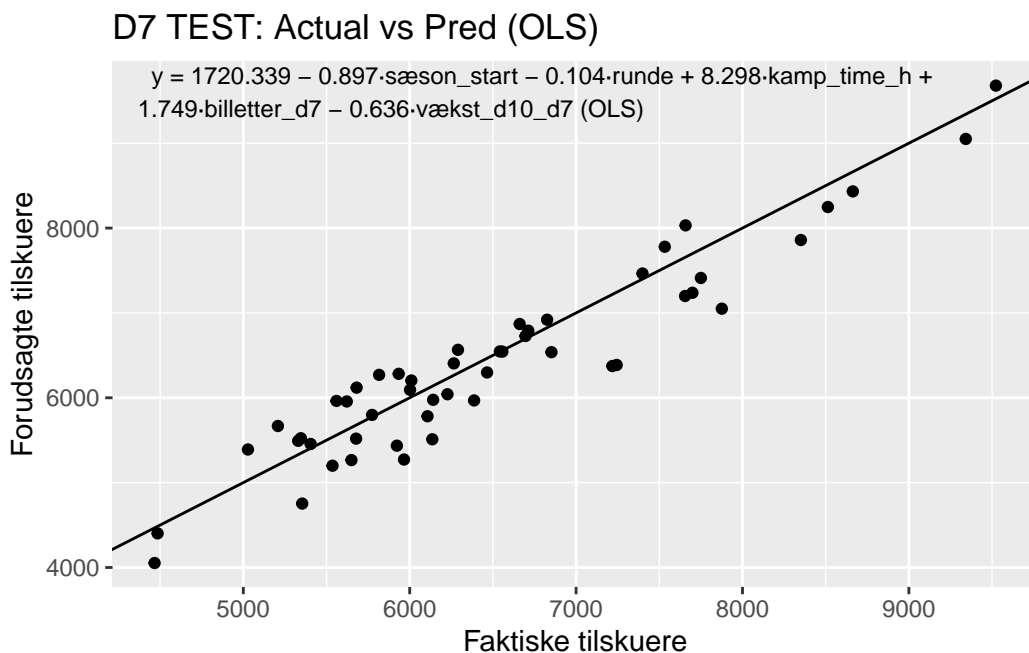
Her udledes læsbare ligninger for OLS-, Ridge- og Lasso-modellerne baseret på de estimerede koefficienter. Formålet er at gøre modellernes struktur og forskelle let forståelige i den efterfølgende formidling.

```
# --- LIGNINGSTEKSTER (alle tre modeller) ---
eq_ols  <- build_lm_equation_text(m_d7_final_lm, y_name = "tilskuere", yhat_name = "ŷ")
eq_ridge <- build_glmnet_equation_text(cv_ridge, x_names = x_names, model_label = "Ridge", yhat_name = "ŷ")
eq_lasso <- build_glmnet_equation_text(cv_lasso, x_names = x_names, model_label = "Lasso", yhat_name = "ŷ")
```

#### 4.2.20.2 Plot: Faktiske vs. forudsagte tilskuere (OLS)

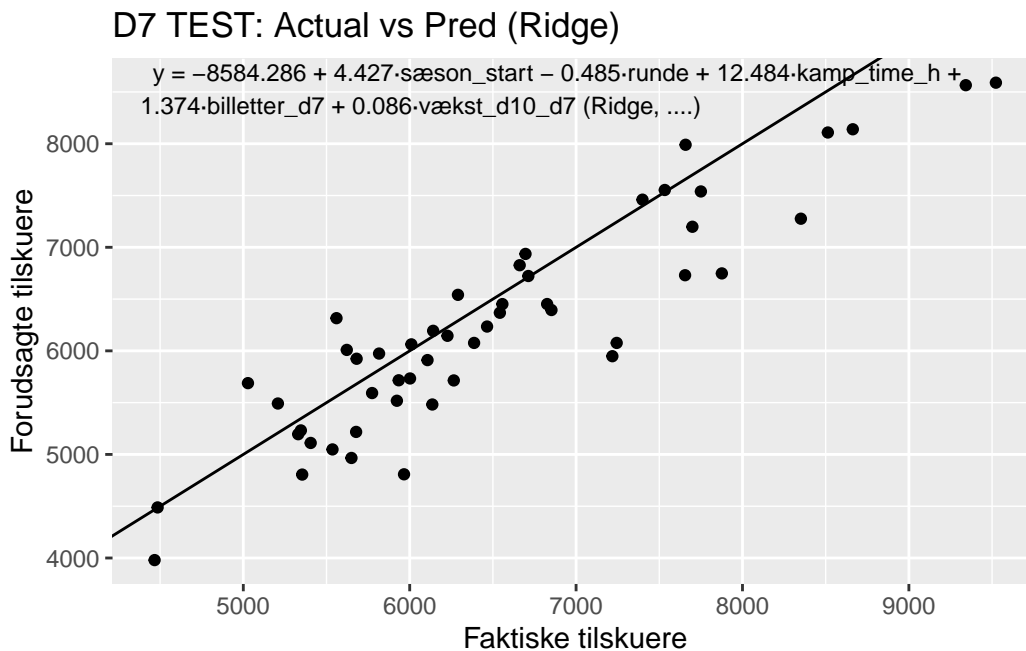
Dette plot sammenligner faktiske og forudsagte tilskuertal for OLS-modellen på testdatasættet. Ligningen indsættes direkte i figuren for at understøtte fortolkningen af modellen.

```
p1 <- ggplot2::ggplot(plot_df, ggplot2::aes(x = actual, y = pred_lm)) +
  ggplot2::geom_point() +
  ggplot2::geom_abline(slope = 1, intercept = 0) +
  ggplot2::labs(
    title = "D7 TEST: Actual vs Pred (OLS)",
    x = "Faktiske tilskuere",
    y = "Forudsagte tilskuere"
  )
p1 <- add_eq_to_plot(p1, eq_ols)
print(p1)
```



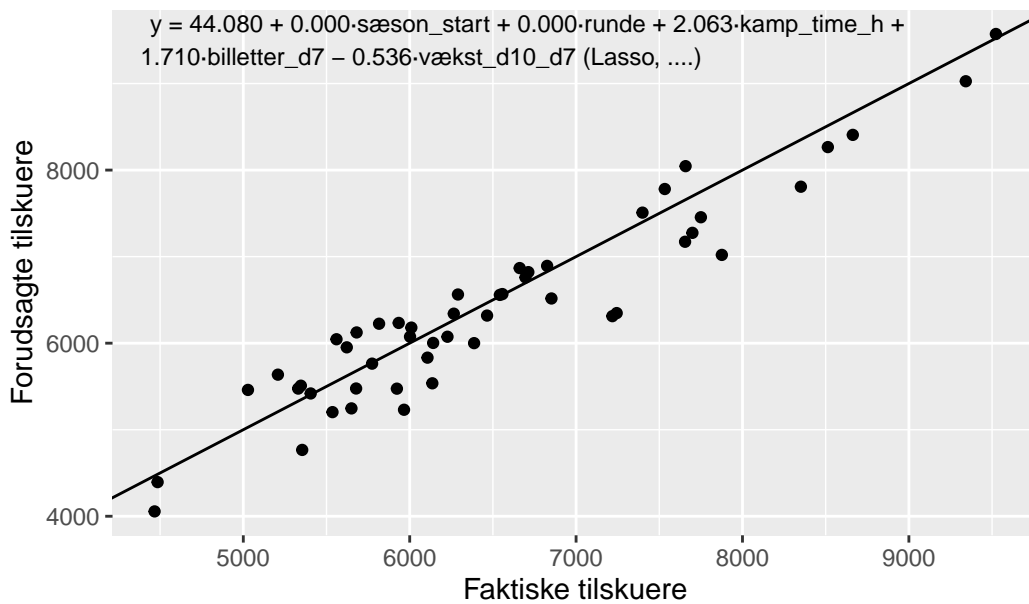
I grafen “D7 TEST: Actual vs Pred (OLS)” vises forholdet mellem faktiske og forudsagte tilskuertal i testdatasættet. Hvert punkt repræsenterer en kamp, hvor den vandrette akse er de faktiske tilskuere, og den lodrette akse er modellens forudsigelse. At punkterne overvejende ligger tæt på diagonalen indikerer, at OLS-modellen har en god præcision og er velegnet til at forudsige tilskuertallet syv dage før kamp.

```
p2 <- ggplot2::ggplot(plot_df, ggplot2::aes(x = actual, y = pred_ridge)) +
  ggplot2::geom_point() +
  ggplot2::geom_abline(slope = 1, intercept = 0) +
  ggplot2::labs(
    title = "D7 TEST: Actual vs Pred (Ridge)",
    x = "Faktiske tilskuere",
    y = "Forudsagte tilskuere"
  )
p2 <- add_eq_to_plot(p2, eq_ridge)
print(p2)
```



```
p3 <- ggplot2::ggplot(plot_df, ggplot2::aes(x = actual, y = pred_lasso)) +
  ggplot2::geom_point() +
  ggplot2::geom_abline(slope = 1, intercept = 0) +
  ggplot2::labs(
    title = "D7 TEST: Actual vs Pred (Lasso)",
    x = "Faktiske tilskuere",
    y = "Forudsagte tilskuere"
  )
p3 <- add_eq_to_plot(p3, eq_lasso)
print(p3)
```

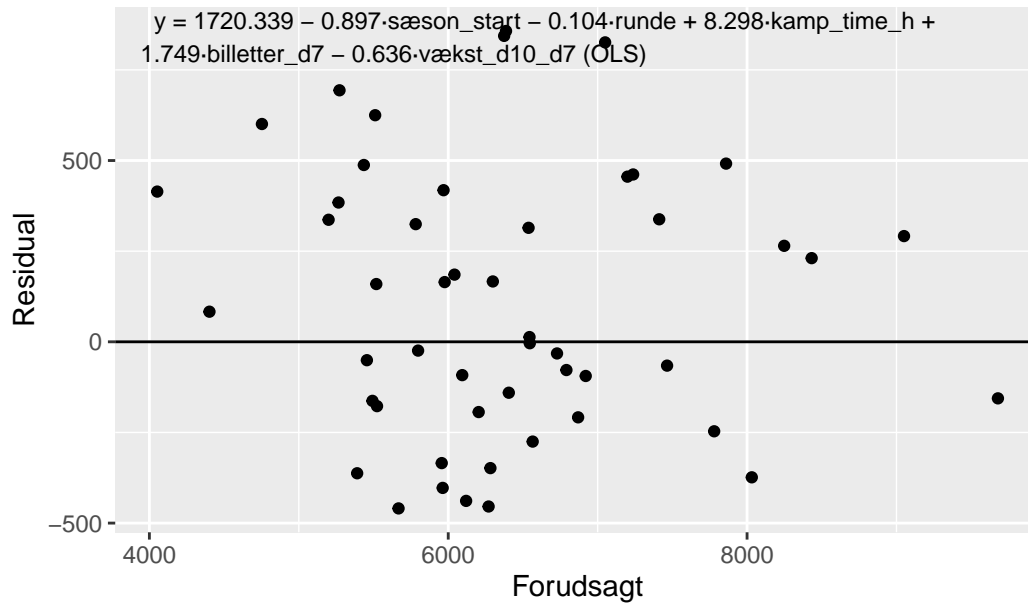
## D7 TEST: Actual vs Pred (Lasso)



I grafen “D7 TEST: Actual vs Pred (Ridge)” vises sammenhængen mellem faktiske og forudsagte tilskuertal for Ridge-modellen. Hvert punkt repræsenterer en kamp, hvor x-aksen angiver de faktiske tilskuertal og y-aksen de forudsagte. Punkterne ligger generelt længere fra diagonalen end i OLS-grafen, hvilket indikerer større usikkerhed og lavere præcision i modellens forudsigelser, særligt ved høje tilskuertal.

```
p4 <- ggplot2::ggplot(plot_df, ggplot2::aes(x = pred_lm, y = resid_lm)) +
  ggplot2::geom_point() +
  ggplot2::geom_hline(yintercept = 0) +
  ggplot2::labs(
    title = "D7 TEST: Residualer vs Pred (OLS)",
    x = "Forudsagt",
    y = "Residual"
  )
p4 <- add_eq_to_plot(p4, eq_ols)
print(p4)
```

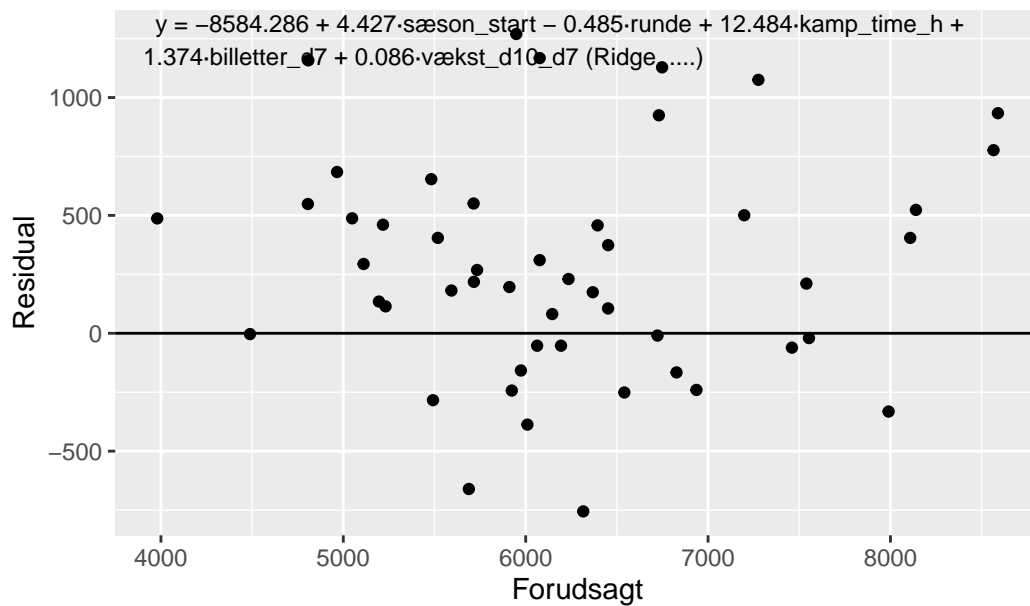
## D7 TEST: Residualer vs Pred (OLS)



I grafen “D7 TEST: Residualer vs Pred (Lasso)” vises residualerne som funktion af de forudsagte tilskuertal fra Lasso-modellen. Residualerne er overordnet centreret omkring nul-linjen uden et tydeligt systematisk mønster, hvilket indikerer, at modellen ikke har en klar strukturel bias. Der ses dog en relativt stor spredning, særligt i midterområdet af de forudsagte værdier, som tyder på, at Lasso-modellen har vanskeligere ved præcist at ramme individuelle kampe sammenlignet med OLS, selvom den overordnet følger niveauet rimeligt.

```
p5 <- ggplot2::ggplot(plot_df, ggplot2::aes(x = pred_ridge, y = resid_ridge)) +
  ggplot2::geom_point() +
  ggplot2::geom_hline(yintercept = 0) +
  ggplot2::labs(
    title = "D7 TEST: Residualer vs Pred (Ridge)",
    x = "Forudsagt",
    y = "Residual"
  )
p5 <- add_eq_to_plot(p5, eq_ridge)
print(p5)
```

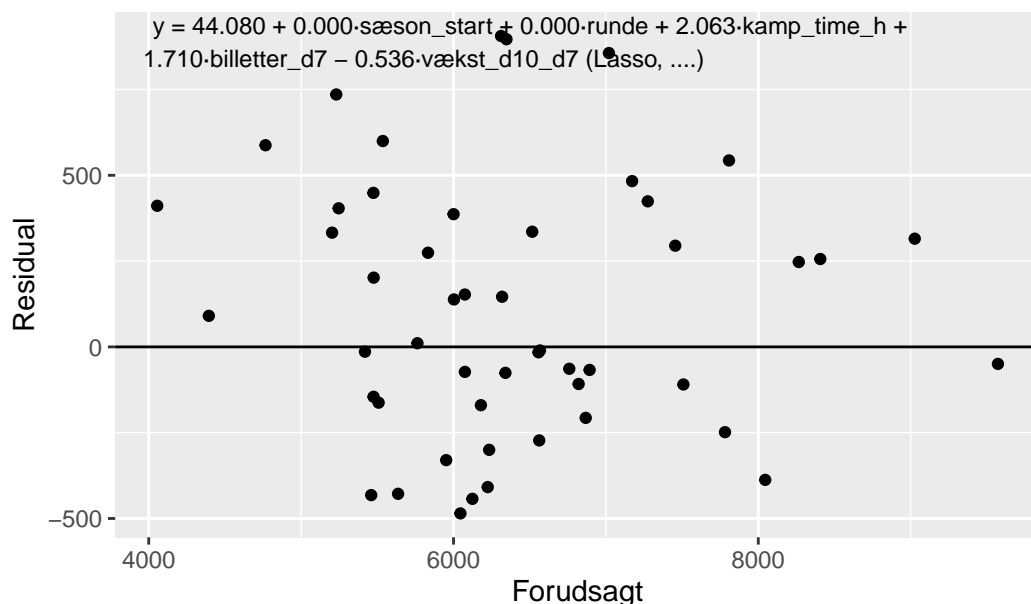
## D7 TEST: Residualer vs Pred (Ridge)



I grafen “D7 TEST: Residualer vs Pred (Ridge)” ses residualerne plottet mod de forudsagte tilskuertal fra Ridge-modellen. Residualerne ligger overordnet omkring nul, men der er en tydelig større spredning end ved OLS, især ved både lave og høje forudsagte værdier. Dette indikerer, at Ridge-modellen har sværere ved at ramme de enkelte kampe præcist og udviser tegn på systematiske afvigelser i yderområderne, hvilket stemmer overens med modellens ringere test-performance målt på RMSE og  $R^2$ .

```
p6 <- ggplot2::ggplot(plot_df, ggplot2::aes(x = pred_lasso, y = resid_lasso)) +
  ggplot2::geom_point() +
  ggplot2::geom_hline(yintercept = 0) +
  ggplot2::labs(
    title = "D7 TEST: Residualer vs Pred (Lasso)",
    x = "Forudsagt",
    y = "Residual"
  )
p6 <- add_eq_to_plot(p6, eq_lasso)
print(p6)
```

## D7 TEST: Residualer vs Pred (Lasso)



I grafen “D7 TEST: Residualer vs Pred (Lasso)” vises sammenhængen mellem de forudsagte tilskuertal og modellens residualer. Residualerne er overvejende centreret omkring nul og fordeler sig relativt jævnt på tværs af de forudsagte værdier, uden tydelige systematiske mønstre. Det indikerer, at Lasso-modellen generelt fanger strukturen i data fornuftigt, men med en lidt større spredning end OLS, især omkring midterområdet. Dette understøtter, at Lasso præsterer tæt på OLS, men uden at give en egentlig forbedring i præcision.

```
# - Resultat-objekt ---
d7_results <- list(
  perf      = perf_d7,
  lm_model  = m_d7_final_lm,
  ridge_lambda = lambda_ridge,
  lasso_lambda = lambda_lasso,
  plot_data  = plot_df,
  equations  = list(ols = eq_ols, ridge = eq_ridge, lasso = eq_lasso)
)

cat("D7 final resultater gemt i objektet: d7_results \n")
```

D7 final resultater gemt i objektet: d7\_results

### 4.2.20.3 Forberedelse af input-features til D7-modellen

Denne funktion udvælger og transformerer de nødvendige variable til et konsistent feature-sæt. Formålet er at sikre korrekt datatyper og genbrugelig klargøring før modellering og prediction.

```
prep_features_d7 <- function(df) {
  stopifnot(all(c(
    "kamp_id", "kamp_dato", "sæson", "runde", "kamp_time_h", "billetter_d7", "vækst_d10_d7"
  ) %in% names(df)))

  df |>
    dplyr::transmute(
```

```

    kamp_id      = as.character(kamp_id),
    kamp_dato     = as.Date(kamp_dato),
    sæson_chr    = as.character(sæson),
    sæson_start  = season_start_year(sæson_chr),
    runde        = as.integer(runde),
    kamp_time_h   = as.integer(kamp_time_h),
    billetter_d7 = as.numeric(billetter_d7),
    vækst_d10_d7 = as.numeric(vækst_d10_d7)
  )
}

```

#### 4.2.20.4 Prediction-helper til D7 (OLS)

Denne funktion genererer forudsigelser fra den endelige OLS-model på nye data. Negative værdier klippes til nul for at sikre realistiske prognoser.

```

predict_d7_lm <- function(model_lm, new_df) {
  new_x <- prep_features_d7(new_df)
  keep <- stats::complete.cases(new_x)

  out <- tibble::tibble(
    kamp_id   = new_x$kamp_id,
    kamp_dato = new_x$kamp_dato,
    pred      = NA_real_
  )

  out$pred[keep] <- clip_nonneg(as.numeric(stats::predict(model_lm, newdata = new_x[keep, ])))
  out
}

```

```

predict_d7_glmnet <- function(cv_model, new_df) {
  new_x <- prep_features_d7(new_df)
  keep <- stats::complete.cases(new_x)

  X <- stats::model.matrix(
    ~ sæson_start + runde + kamp_time_h + billetter_d7 + vækst_d10_d7,
    data = new_x
  )[, -1, drop = FALSE]

  out <- tibble::tibble(
    kamp_id   = new_x$kamp_id,
    kamp_dato = new_x$kamp_dato,
    pred      = NA_real_
  )

  out$pred[keep] <- clip_nonneg(as.numeric(stats::predict(
    cv_model,
    newx = X[keep, , drop = FALSE],
    s = "lambda.min"
  )))
}

```

```
  out
}
```

#### 4.2.20.5 Prediction-helper til D7 (Ridge/Lasso)

Denne funktion genererer forudsigelser fra Ridge- og Lasso-modeller på nye data. Den sikrer konsistent feature-opbygning og håndtering af manglende værdier.

```
d7_artifacts <- list(
  perf = perf_d7,
  models = list(
    lm = m_d7_final_lm,
    ridge = cv_ridge,
    lasso = cv_lasso
  ),
  split_info = list(
    test_prop = test_prop,
    cut_date_test_start = min(test_d7$kamp_dato),
    n_train = nrow(train_d7),
    n_test = nrow(test_d7)
  ),
  equations = list(ols = eq_ols, ridge = eq_ridge, lasso = eq_lasso)
)

cat("D7 pipeline klar  Artefakter gemt i: d7_artifacts\n")
```

```
D7 pipeline klar  Artefakter gemt i: d7_artifacts
```

```
##Shiny prognose App
```

#### 4.2.21 Afgrænsning af forudsigelser til stadionkapacitet

Denne funktion sikrer, at modelforudsigelser holdes inden for realistiske grænser. Forudsigelser klippes til intervallet fra 0 til stadionets maksimale kapacitet.

```
# CAP/CLIP (stadionloft) - klar til dit produktlag

STADIUM_CAP_VFF <- 10000

clip_to_cap <- function(x, cap = STADIUM_CAP_VFF) {
  pmin(pmax(as.numeric(x), 0), cap)
}
```

#### 4.2.22 Klargøring til Shiny-applikation

Denne kode forbereder miljøet til Shiny ved at indlæse nødvendige pakker og validere, at alle påkrævede objekter findes. Formålet er at sikre, at applikationen kan afvikles stabilt uden runtime-fejl.

```
# shiny
```

```
suppressPackageStartupMessages({  
  library(dplyr)  
  library(tibble)  
  library(ggplot2)  
  library(shiny)  
})
```

```
.must_exist <- function(x) exists(x, inherits = TRUE)
```

```
stopifnot(.must_exist("perf_d7"), .must_exist("plot_df"), .must_exist("train_d7"), .must_exist("test_d7"))  
stopifnot(.must_exist("m_d7_final_lm"), .must_exist("cv_lasso"))  
stopifnot(.must_exist("rmse_vec"), .must_exist("clip_nonneg"), .must_exist("clip_to_cap"))
```

```
stopifnot(all(c("model", "RMSE") %in% names(perf_d7)))  
stopifnot(all(c("actual", "pred_lm", "pred_lasso") %in% names(plot_df)))  
stopifnot(all(c("sæson_start", "runde", "kamp_time_h", "billetter_d7", "vækst_d10_d7") %in% names(train_d7)))
```

```
# shiny
```

```
# 1) TOP 2: vælg default (bedst) mellem OLS og Lasso ud fra perf_d7
```

```
perf_top2 <- perf_d7 |>  
  dplyr::filter(grepl("^OLS", model, ignore.case = TRUE) | grepl("^Lasso", model, ignore.case = TRUE))
```

```
# Fail-safe hvis labels er anderledes
```

```
if (nrow(perf_top2) == 0) {  
  perf_top2 <- tibble::tibble(  
    model = c("OLS (lm)", "Lasso (cv.glmnet)"),  
    RMSE = c(Inf, Inf),  
    MAE = c(NA_real_, NA_real_),  
    R2 = c(NA_real_, NA_real_)  
  )  
}
```

```
top2_best_row <- perf_top2 |> arrange(RMSE) |> slice(1)
```

```
default_key <- dplyr::case_when(  
  grepl("^OLS", top2_best_row$model[1], ignore.case = TRUE) ~ "lm",  
  grepl("^Lasso", top2_best_row$model[1], ignore.case = TRUE) ~ "lasso",  
  TRUE ~ "lm"  
)
```

```
cat(" Default model (bedst af OLS/LASSO):", as.character(top2_best_row$model[1]), "=>", default_key, "\n")
```

#### 4.2.23 Valg af standardmodel til Shiny

Denne kode identificerer den bedste model mellem OLS og Lasso baseret på RMSE. Den valgte model sættes som default i Shiny-applikationen med indbyggede failsafes.

```

# shiny

# =====
# 2) Hjælpere: format + ligningstekst
# =====

.format_num <- function(x, digits = 3) {
  format(round(as.numeric(x), digits), big.mark = ".", decimal.mark = ",", trim = TRUE)
}

lm_equation_text <- function(fit, digits = 3, max_terms = 8) {
  b <- coef(fit)
  b[is.na(b)] <- 0
  nm <- names(b)

  intercept <- if ("(Intercept)" %in% nm) b["(Intercept)"] else 0
  rest <- setdiff(nm, "(Intercept)")
  if (length(rest) == 0) return(paste0("ŷ = ", .format_num(intercept, digits)))

  ord <- order(abs(b[rest]), decreasing = TRUE)
  rest <- rest[ord]
  if (length(rest) > max_terms) rest <- rest[1:max_terms]

  rhs <- vapply(rest, function(v) paste0(v, ".", .format_num(b[v], digits)), character(1))
  paste0("ŷ = ", .format_num(intercept, digits), " + ", paste(rhs, collapse = " + "))
}

glmnet_equation_text <- function(cvobj, s = "lambda.min", digits = 3, max_terms = 10) {
  cm <- as.matrix(coef(cvobj, s = s))
  if (nrow(cm) == 0) return("Ingen koefficienter")

  coefs <- as.numeric(cm[, 1])
  names(coefs) <- rownames(cm)

  intercept <- if ("(Intercept)" %in% names(coefs)) coefs["(Intercept)"] else 0
  rest <- setdiff(names(coefs), "(Intercept)")

  nz <- rest[coefs[rest] != 0]
  if (length(nz) == 0) return(paste0("ŷ = ", .format_num(intercept, digits), " (alle øvrige = 0)"))

  ord <- order(abs(coefs[nz]), decreasing = TRUE)
  nz <- nz[ord]
  if (length(nz) > max_terms) nz <- nz[1:max_terms]

  rhs <- vapply(nz, function(v) paste0(v, ".", .format_num(coefs[v], digits)), character(1))
  paste0("ŷ = ", .format_num(intercept, digits), " + ", paste(rhs, collapse = " + "))
}

plot_pva_with_eq <- function(df, title, eq_text, rmse_val) {
  ggplot(df, aes(x = y, y = yhat)) +
    geom_point(alpha = 0.7) +

```

```

geom_abline(slope = 1, intercept = 0) +
labs(
  title = title,
  x = "Faktiske tilskuere",
  y = "Forudsagte tilskuere"
) +
annotate(
  "text",
  x = Inf, y = -Inf,
  hjust = 1.02, vjust = -0.2,
  label = paste0("RMSE = ", .format_num(rmse_val, 1)),
  size = 3.8
) +
annotate(
  "text",
  x = -Inf, y = Inf,
  hjust = -0.02, vjust = 1.15,
  label = eq_text,
  size = 3.2
) +
theme_minimal()
}

```

#### 4.2.24 Hjælpefunktioner til sæsoninput i Shiny

Disse funktioner håndterer konvertering og validering af sæsoninput fra brugeren, fx “2025/2026”. Formålet er at sikre robust fortolkning af sæsonen og give klare fallback-beskeder ved ugyldigt format.

```

# shiny

# =====
# 3) Sæson-helpers: "2025/2026" -> sæson_start = 2025
# =====
season_label <- function(season_start_int) {
  paste0(as.integer(season_start_int), "/", as.integer(season_start_int) + 1L)
}

parse_season_start <- function(season_text) {
  s <- trimws(as.character(season_text))
  ok <- grepl("^\\d{4}/\\d{4}$", s)

  if (!ok) {
    # fallback: prøv at finde første 4 cifre
    y <- suppressWarnings(as.integer(substr(s, 1, 4)))
    if (!is.finite(y)) return(list(start = NA_integer_, note = "Ugyldigt sæsonformat. Brug fx 2025/2026"))
    return(list(start = y, note = "Sæsonformat er ikke 'YYYY/YYYY'. Jeg bruger første år som sæson_start"))
  }

  y1 <- suppressWarnings(as.integer(substr(s, 1, 4)))
  y2 <- suppressWarnings(as.integer(substr(s, 6, 9)))
  if (!is.finite(y1) || !is.finite(y2)) return(list(start = NA_integer_, note = "Ugyldigt sæsonformat."))
}

```

```

if (y2 != y1 + 1L) {
  return(list(start = y1, note = "Andet år er ikke første+1. Jeg bruger første år som sæson_start."))
}
list(start = y1, note = "")
}

```

#### 4.2.25 Visualisering af bedste D7-model (OLS/Lasso)

Denne kode genererer ét samlet *Pred vs. Actual*-plot for den bedst performende model mellem OLS og Lasso. Valget sker automatisk ud fra test-RMSE, så visualiseringen altid afspejler den stærkeste D7-model.

```

# shiny

# =====
# 4) ÉN graf: Pred vs Actual for default bedste (OLS/LASSO) - sæson tekst neutral
# =====
y <- as.numeric(plot_df$actual)

if (default_key == "lm") {
  yhat <- as.numeric(plot_df$pred_lm)
  eq_best <- lm_equation_text(m_d7_final_lm, digits = 3, max_terms = 8)
  title_best <- "D7 - BEDSTE (OLS/LASSO): OLS (lm) - Pred vs Actual"
} else {
  yhat <- as.numeric(plot_df$pred_lasso)
  eq_best <- glmnet_equation_text(cv_lasso, s = "lambda.min", digits = 3, max_terms = 10)
  title_best <- "D7 - BEDSTE (OLS/LASSO): Lasso ( ) - Pred vs Actual"
}

df_best <- tibble::tibble(y = y, yhat = yhat)
rmse_best <- rmse_vec(df_best$y, df_best$yhat)

print(plot_pva_with_eq(df_best, title_best, eq_best, rmse_best))
cat("\n D7 top2-plot færdig\n\n")

```

#### 4.2.26 Shiny-applikation til D7-prognoser

Denne sektion definerer brugergrænsefladen til en Shiny-applikation, hvor D7-modellen anvendes operationelt. Brugeren kan vælge model, indsætte centrale kampinput og få en realistisk tilskuerprognose under hensyntagen til stadionkapacitet.

```

# shiny

# =====
# 5) SHINY APP - sæson er input (default 2025/2026)
# =====
.range_or <- function(x, fallback = c(0, 1)) {
  r <- range(as.numeric(x), na.rm = TRUE)
  if (any(!is.finite(r))) fallback else r
}

```

```

}

.med_or <- function(x, fallback = 0) {
  v <- stats::median(as.numeric(x), na.rm = TRUE)
  if (!is.finite(v)) fallback else v
}

.fill_bucket <- function(pct) {
  if (!is.finite(pct)) return("UKENDT")
  if (pct >= 85) return("HØJ")
  if (pct >= 60) return("MIDDEL")
  "LAV"
}

.predict_model <- function(model_key, sæson_start, runde, kamp_time_h, billetter_d7, vækst_d10_d7) {

  nd <- data.frame(
    sæson_start = as.integer(sæson_start),
    runde       = as.integer(runde),
    kamp_time_h = as.integer(kamp_time_h),
    billetter_d7 = as.numeric(billetter_d7),
    vækst_d10_d7 = as.numeric(vækst_d10_d7)
  )

  if (model_key == "lm") {
    yhat <- suppressWarnings(as.numeric(predict(m_d7_final_lm, newdata = nd)))
  } else {
    X <- model.matrix(~ sæson_start + runde + kamp_time_h + billetter_d7 + vækst_d10_d7, data = nd)[,
    yhat <- suppressWarnings(as.numeric(predict(cv_lasso, newx = X, s = "lambda.min")))]
  }

  if (length(yhat) == 0) yhat <- NA_real_
  clip_nonneg(yhat)
}

.eq_for_key <- function(model_key) {
  if (model_key == "lm") {
    lm_equation_text(m_d7_final_lm, digits = 3, max_terms = 8)
  } else {
    glmnet_equation_text(cv_lasso, s = "lambda.min", digits = 3, max_terms = 10)
  }
}

.model_name <- function(model_key) {
  if (model_key == "lm") "OLS (lm)" else "Lasso (cv.glmnet)"
}

ui <- fluidPage(
  titlePanel("D7 - Prediction (OLS vs Lasso)"),

  sidebarLayout(

```

```

sidebarPanel(
  tags$h4("Model"),
  selectInput(
    "model_choice",
    "Vælg model (default = bedst)",
    choices = c("OLS (lm)" = "lm", "Lasso (cv.glmnet)" = "lasso"),
    selected = default_key
  ),
  tags$hr(),

  tags$h4("Inputs"),
  textInput("season_txt", "Sæson (YYYY/YYYY)", value = "2025/2026"),

  numericInput(
    "runde", "runde",
    value = .med_or(train_d7$runde, fallback = 1),
    min = .range_or(train_d7$runde, fallback = c(1, 40))[1],
    max = .range_or(train_d7$runde, fallback = c(1, 40))[2]
  ),
  numericInput(
    "kamp_time_h", "kamp_time_h",
    value = .med_or(train_d7$kamp_time_h, fallback = 18),
    min = .range_or(train_d7$kamp_time_h, fallback = c(10, 22))[1],
    max = .range_or(train_d7$kamp_time_h, fallback = c(10, 22))[2]
  ),
  numericInput(
    "billetter_d7", "billetter_d7",
    value = .med_or(train_d7$billetter_d7, fallback = 0),
    min = .range_or(train_d7$billetter_d7, fallback = c(0, 12000))[1],
    max = .range_or(train_d7$billetter_d7, fallback = c(0, 12000))[2]
  ),
  numericInput(
    "vækst_d10_d7", "vækst_d10_d7",
    value = .med_or(train_d7$vækst_d10_d7, fallback = 0),
    min = 0,
    max = max(0, .range_or(train_d7$vækst_d10_d7, fallback = c(0, 12000))[2])
  ),

  tags$hr(),
  numericInput("capacity", "Stadion-kapacitet", value = 10000, min = 1),
  actionButton("go", "Predict", class = "btn-primary")
),

mainPanel(
  tags$h4("Output"),
  tags$div(style="font-size:20px; margin-bottom:8px;", textOutput("pred_txt")),
  tags$div(style="font-size:20px; margin-bottom:8px;", textOutput("pct_txt")),
  tags$div(style="font-size:20px; margin-bottom:8px;", textOutput("delta_txt")),
  tags$div(style="font-size:14px; margin-bottom:6px; color:#333;", textOutput("season_note_txt")),
  tags$div(style="font-size:14px; margin-bottom:14px; color:#333;", textOutput("eq_txt")),
  plotOutput("cap_plot", height = "520px")
)

```

```
)  
)  
)
```

#### 4.2.26.1 Serverlogik til Shiny-applikationen

Denne sektion definerer server-logikken for Shiny-applikationen, herunder beregning af prognoser, validering af input og dynamisk opdatering af output. Modellen anvendes operationelt til at levere tilskuerprognoser, kapacitetsudnyttelse og visualiseringer baseret på brugerens valg.

```
# shiny  
  
server <- function(input, output, session) {  
  
  pred_obj <- eventReactive(input$go, {  
  
    cap <- as.numeric(input$capacity)  
    if (!is.finite(cap) || cap <= 0) cap <- 1  
  
    key <- input$model_choice  
  
    season_parsed <- parse_season_start(input$season_txt)  
    sæson_start <- season_parsed$start  
  
    if (!is.finite(sæson_start)) {  
      return(list(  
        ok = FALSE,  
        msg = "Ugyldig sæson. Brug formatet 2025/2026.",  
        note = season_parsed$note  
      ))  
    }  
  
    yhat <- .predict_model(  
      model_key = key,  
      sæson_start = sæson_start,  
      runde = input$runde,  
      kamp_time_h = input$kamp_time_h,  
      billetter_d7 = input$billetter_d7,  
      vækst_d10_d7 = input$vækst_d10_d7  
    )  
  
    yhat_cap <- clip_to_cap(yhat, cap = cap)  
  
    pct <- (as.numeric(yhat_cap) / cap) * 100  
    pct_clip <- max(min(pct, 100), 0)  
  
    bil <- as.numeric(input$billetter_d7)  
    delta <- if (is.finite(bil)) (as.numeric(yhat_cap) - bil) else NA_real_  
  
    list(  

```

```

    ok = TRUE,
    pred = as.numeric(yhat_cap),
    pct = pct_clip,
    cap = cap,
    bil_d7 = bil,
    delta = delta,
    eq = .eq_for_key(key),
    model_name = .model_name(key),
    season_txt = season_label(sæson_start),
    note = season_parsed$note
  )
})

output$pred_txt <- renderText({
  x <- pred_obj()
  if (is.null(x)) return("Ingen prediction endnu.")
  if (!isTRUE(x$ok)) return(x$msg)

  paste0(
    "Model: ", x$model_name, " - sæson ", x$season_txt, "\n",
    "Forudsagt tilskuertal: ", format(round(x$pred), big.mark = ".", decimal.mark = ","))
  )
})

output$pct_txt <- renderText({
  x <- pred_obj()
  if (is.null(x) || !isTRUE(x$ok)) return("")
  paste0("Stadion fyldt: ", format(round(x$pct, 1), big.mark = ".", decimal.mark = ","), " %")
})

output$delta_txt <- renderText({
  x <- pred_obj()
  if (is.null(x) || !isTRUE(x$ok)) return("")
  if (!is.finite(x$bil_d7)) return("Ekstra billetter (D7 → kampdag): kan ikke beregnes (billetter_d7)")
  paste0(
    "Ekstra billetter (D7 → kampdag): ",
    format(round(x$delta), big.mark = ".", decimal.mark = ","),
    " ( $\hat{y}$  - billetter_d7)"
  )
})

output$season_note_txt <- renderText({
  x <- pred_obj()
  if (is.null(x) || !isTRUE(x$ok)) return("")
  if (is.null(x$note) || !nzchar(x$note)) return("")
  paste0("Note: ", x$note)
})

output$eq_txt <- renderText({
  x <- pred_obj()
  if (is.null(x) || !isTRUE(x$ok)) return("")

```

```

    paste0("Ligning (kort): ", x$eq)
  })

output$cap_plot <- renderPlot({
  x <- pred_obj()
  if (is.null(x) || !isTRUE(x$ok)) return(NULL)

  bucket <- .fill_bucket(x$pct)

  df <- data.frame(
    label = "Stadionkapacitet",
    y = x$pred,
    bucket = bucket
  )

  ggplot(df, aes(x = label, y = y, fill = bucket)) +
    geom_col(width = 0.6) +
    geom_hline(yintercept = x$cap, linewidth = 0.9, linetype = 2) +
    coord_cartesian(ylim = c(0, x$cap)) +
    scale_fill_manual(
      values = c("LAV" = "#d55e00", "MIDDEL" = "#e69f00", "HØJ" = "#009e73", "UKENDT" = "grey60"),
      breaks = c("LAV", "MIDDEL", "HØJ", "UKENDT"),
      name = "Fyldningsniveau"
    ) +
    labs(
      x = NULL,
      y = "Tilskuere (forudsagt)",
      title = "Forudsagt tilskuertal vs stadionkapacitet",
      subtitle = paste0(
        "Kapacitet (stiplet linje): ",
        format(round(x$cap), big.mark=".", decimal.mark=",")
      )
    ) +
    theme_minimal(base_size = 16) +
    theme(
      legend.position = "top",
      plot.title = element_text(face = "bold"),
      panel.grid.minor = element_blank()
    )
  })
}

shinyApp(ui, server)

```

#### 4.2.26.2 Case: D7-prediction som aktivt beslutningsværktøj

Der er en Superliga-hjemmekamp på Energi Viborg Arena om syv dage, og Palle, som er ansvarlig for bemanding og kampafvikling hos Viborg FF, skal i gang med planlægningen. Med et stadion, der rummer 10.000 pladser, er det afgørende tidligt at vurdere, om efterspørgslen peger mod et lavt, middel eller højt fremmøde.

Palle anvender D7-prediction-appen og indtaster de kendte oplysninger: sæson 2025/2026, runde 21, kampstart kl. 18 samt et aktuelt billetsalg på 2.680 billetter syv dage før kamp.

## D7 — Prediction (OLS vs Lasso)

**Model**  
Vælg model (default = bedst)  
OLS (lm)

**Inputs**  
Sæson (YYYY/YYYY)  
2025/2026  
runde  
21  
kamp\_time\_h  
18  
billetter\_d7  
2680  
vækst\_d10\_d7  
281  
Stadion-kapacitet  
10000  
Predict

### Output

Model: OLS (lm) — sæson 2025/2026 Forudsagt tilskuertal: 4.560

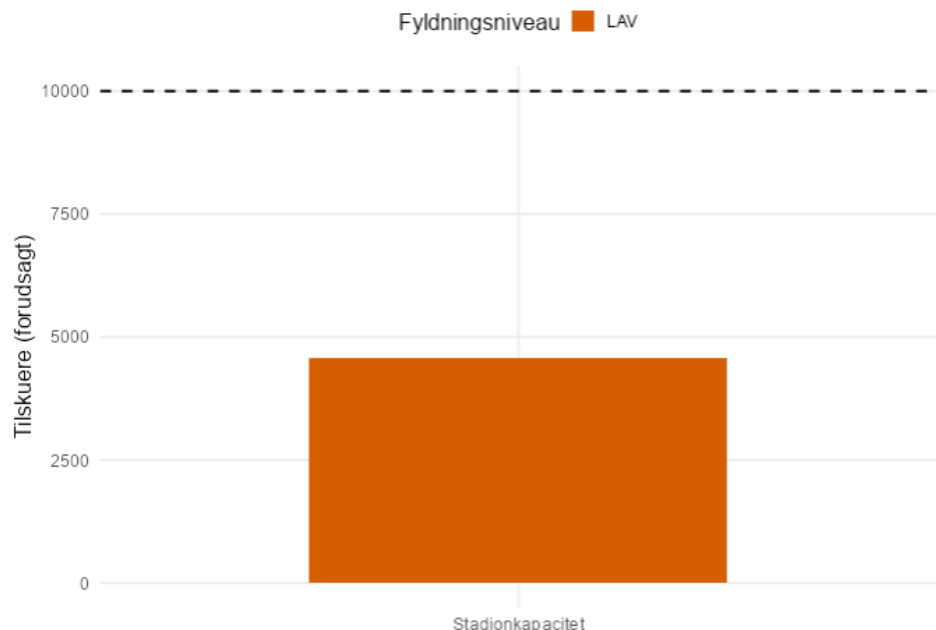
Stadion fyldt: 45,6 %

Ekstra billetter (D7 → kampdag): 1.880 ( $\hat{y}$  - billetter\_d7)

Ligning (kort):  $\hat{y} = 1.720,339 + \text{kamp\_time\_h} \cdot 8,298 + \text{billetter\_d7} \cdot 1,749 + \text{sæson\_start} \cdot -0,897 + \text{vækst\_d10\_d7} \cdot -0,636 + \text{runde} \cdot -0,104$

### Forudsagt tilskuertal vs stadionkapacitet

Kapacitet (stiplet linje): 10.000



OLS-model giver en prognose på ca. 4.560 tilskuere, svarende til en forventet kapacitetsudnyttelse på 45,6 %.

Denne prognose indikerer, at der fortsat er betydelig ledig kapacitet på stadion. På den baggrund vælger Palle at aktivere marketingindsatsen. Den SoMe-ansvarlige får grønt lys til at iværksætte målrettede kampagner på Facebook og Instagram, eksempelvis med fokus på kampstemning, familietilbud eller sidste-øjebliks-billetter, for at løfte efterspørgslen i dagene op til kampstart.

Samtidig estimerer modellen et forventet ekstra billetsalg på omkring 1.880 billetter frem mod kampdagen. Det giver Palle et realistisk pejlemærke for, hvor meget bemandingen skal kunne skales, hvis marketingindsatsen lykkes. D7-predictionen bruges dermed ikke kun passivt, men som et aktivt styringsværktøj, hvor prognosen danner grundlag for både marketingbeslutninger og operationel planlægning.

## 5 ML model D10

Dette afsnit samler hele D10-workflowet i ét sammenhængende script. Der bygges først en fælles baseline med én række pr. kamp og tilhørende features, hvorefter et D10-analysedatasæt konstrueres med billetsalg 10 dage før kamp. Herefter gennemføres systematiske variabeltests mod en fast nulmodel for at vurdere marginal effekt og data-omkostning. Afslutningsvis estimeres og evalueres endelige D10-modeller via et tidsbaseret split, hvor OLS, Ridge og Lasso sammenlignes, og den bedste model gøres operationel gennem artefakter, plots og en Shiny-applikation.

### 5.0.1 Pakker og opsætning

Her indlæses alle nødvendige pakker samlet via pacman, så miljøet er konsistent og reproducerbart. Samtidig fastsættes globale indstillinger, der sikrer læsbar output og stabil kørsel under hele analysen.

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

pacman::p_load(
  DBI,
  odbc,
  dplyr,
  lubridate,
  stringr,
  tibble,
  ggplot2,
  glmnet,
  rsample
)

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

Pakker er klar

### 5.0.2 Hjælpfunktioner

```
ensure_renviro <- function() {
  req <- c("AZURE_SQL_SERVER", "AZURE_SQL_DB", "AZURE_SQL_UID", "AZURE_SQL_PWD")
  vals <- Sys.getenv(req)
  miss <- req[vals == ""]
  if (length(miss) > 0) stop(" Manglende miljøvariabler i .Renviro: ", paste(miss, collapse = ", "))
  cat(" .Renviro OK\n\n")
}
```

#### 5.0.2.1 Miljøvariabler og adgang

Denne funktion sikrer, at alle nødvendige Azure-forbindelsesoplysninger er korrekt sat i .Renviro, før der arbejdes videre. Hvis noget mangler, stoppes scriptet med det samme for at undgå skjulte forbindelsesfejl senere i pipeline.

```
connect_azure_retry <- function(
  forsøg_max = 6,
  timeouts = c(60, 180, 240, 360, 600, 900),
  delay_sec = 10
) {
  for (i in seq_len(forsøg_max)) {
    timeout_brug <- timeouts[min(i, length(timeouts))]
    cat("Forsøg ", i, " / ", forsøg_max, " - ConnectionTimeout = ", timeout_brug, " sek\n", sep = "")
```

```

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"),
    Encrypt  = "yes",
    TrustServerCertificate = "no",
    ConnectionTimeout = timeout_brug
  ),
  silent = TRUE
)
if (!inherits(con_try, "try-error")) {
  cat(" Forbundet til Azure SQL\n\n")
  return(con_try)
}
if (i < forsøg_max) Sys.sleep(delay_sec)
}
stop(" Kunne ikke forbinde til Azure SQL efter ", forsøg_max, " forsøg.")
}

```

```

safe_dbReadTable <- function(con, schema, table, forsøg_max = 4, delay_sec = 3) {
  for (i in seq_len(forsøg_max)) {
    out <- try(DBI::dbReadTable(con, DBI::Id(schema = schema, table = table)), silent = TRUE)
    if (!inherits(out, "try-error")) return(out)
    msg <- as.character(out)
    cat(" dbReadTable fejlede (", i, " / ", forsøg_max, "): ", schema, ".", table, "\n", sep = "")
    cat(substr(msg, 1, 220), "... \n\n")
    if (i < forsøg_max) Sys.sleep(delay_sec)
  }
  stop(" Kunne ikke læse tabel: ", schema, ".", table)
}

```

### 5.0.2.2 Sikker datalæsning fra SQL

Funktionen læser en tabel fra Azure SQL med indbygget retry-logik. Hvis læsningen fejler midlertidigt, forsøger den igen et fast antal gange med korte pauser, så ustabile forbindelser ikke stopper hele pipeline unødigt.

```

time_chr_to_hour <- function(x) {
  x <- str_trim(as.character(x))
  x <- if_else(str_detect(x, "^\\d{1,2}:\\d{2}$"), paste0(x, ":00"), x)
  ok <- str_detect(x, "^\\d{1,2}:\\d{2}:\\d{2}$")
  out <- rep(NA_integer_, length(x))
  out[ok] <- as.integer(str_extract(x[ok], "^\\d{1,2}"))
  out
}

```

### 5.0.2.3 Dublet-håndtering pr kamp

Funktionen sikrer, at der kun findes én række pr kamp\_id. Hvis der er dubletter, beholdes den mest komplette række ved at vælge den med færrest manglende værdier, så datatabet minimeres.

```
dedup_1row_by_id <- function(df, id_col = "kamp_id") {
  stopifnot(id_col %in% names(df))
  df2 <- df
  df2$.na_count <- rowSums(is.na(df2))
  df2 |>
    arrange(.data[[id_col]], .na_count) |>
    group_by(.data[[id_col]]) |>
    slice(1) |>
    ungroup() |>
    select(-.na_count)
}
```

#### 5.0.2.4 Sikker datokonvertering

Funktionen konverterer værdier til datoformat på en robust måde. Hvis input allerede er en dato, returneres den uændret, og ellers forsøges en kontrolleret konvertering uden at afbryde koden ved fejl.

```
as_date_safely <- function(x) {
  if (inherits(x, "Date")) return(x)
  suppressWarnings(as.Date(x, format = "%d-%m-%Y"))
}
```

#### 5.0.2.5 Sæsonens startår

Funktionen udtrækker det første årstal fra en sæsonangivelse og bruger det som numerisk startår for sæsonen i modelleringen.

```
season_start_year <- function(season_chr) {
  x <- as.character(season_chr)
  suppressWarnings(as.integer(stringr::str_extract(x, "\\d{4}")))
}
```

#### 5.0.2.6 Negativ afskæring

Funktionen sikrer, at værdier ikke bliver negative ved at erstatte alle tal under nul med 0, så modellens output altid giver realistiske tilskuental.

```
clip_nonneg <- function(x) pmax(as.numeric(x), 0)
```

#### 5.0.2.7 Modelmål for præcision

Disse funktioner beregner standardmål for modelkvalitet. RMSE måler den typiske fejlstørrelse og straffer store fejl hårdt. MAE viser den gennemsnitlige absolutte fejl og er lettere at fortolke.  $R^2$  angiver, hvor stor en andel af variationen i tilskuentallet modellen forklarer.

```
rmse_vec <- function(y, yhat) sqrt(mean((y - yhat) ^ 2, na.rm = TRUE))

mae_vec <- function(y, yhat) mean(abs(y - yhat), na.rm = TRUE)

r2_vec <- function(y, yhat) {
  sse <- sum((y - yhat) ^ 2, na.rm = TRUE)
  sst <- sum((y - mean(y, na.rm = TRUE)) ^ 2, na.rm = TRUE)
  1 - (sse / sst)
}
```

### 5.0.3 Forbindelse til datagrundlag

Her kontrolleres det, at alle nødvendige loginoplysninger findes i .Renviron, hvorefter der oprettes en stabil forbindelse til Azure SQL-databasen med automatisk genforsøg ved fejl.

```
ensure_renviro()
```

```
.Renviron OK
```

```
con <- connect_azure_retry()
```

```
Forsøg 1 / 6 - ConnectionTimeout = 60 sek
Forbundet til Azure SQL
```

### 5.0.4 Indlæsning af baseline

Her indlæses den fælles baseline fra en gemt RDS-fil. Datasættet standardiseres, så kamp\_id og kamp\_dato har korrekt datatype, og der kontrolleres, at der findes præcis de centrale variable, som resten af modellen bygger videre på.

```
baseline_dir <- "C:/Users/janpe/OneDrive/Skrivebord/PBA Dataanalyse/01_Første semester/1 Semester proj
baseline_file <- file.path(baseline_dir, "baseline_azure.rds")
stopifnot(file.exists(baseline_file))

baseline <- readRDS(baseline_file) |>
  mutate(
    kamp_id = as.character(kamp_id),
    kamp_dato = as.Date(kamp_dato)
  )

stopifnot(all(c("kamp_id", "kamp_dato", "tilskuere") %in% names(baseline)))

cat("Baseline loaded   Rækker: ", nrow(baseline),
    "   Unikke kamp_id: ", n_distinct(baseline$kamp_id), "\n\n", sep = "")
```

```
Baseline loaded   Rækker: 259   Unikke kamp_id: 254
```

### 5.0.5 Dublettjek af baseline

Her kontrolleres det, om der findes flere rækker med samme `kamp_id`. Hvis der opdages dubletter, reduceres datasættet automatisk til én række pr. kamp ved at beholde den mest komplette observation. Dermed sikres en entydig baseline, som er stabil for al efterfølgende modellering.

```
dup_ids <- baseline |> count(kamp_id) |> filter(n > 1)
cat("--- BASELINE QA: DUPLIKAT-TJEK ---\n")
```

```
--- BASELINE QA: DUPLIKAT-TJEK ---
```

```
cat("Antal kamp_id med dubletter: ", nrow(dup_ids), "\n", sep = "")
```

```
Antal kamp_id med dubletter: 5
```

```
if (nrow(dup_ids) > 0) {
  baseline <- dedup_1row_by_id(baseline, "kamp_id")
  stopifnot(nrow(baseline) == n_distinct(baseline$kamp_id))
  cat(" Baseline deduplikeret (1 række pr kamp_id)\n\n")
} else {
  cat(" Ingen dubletter fundet\n\n")
}
```

```
Baseline deduplikeret (1 række pr kamp_id)
```

### 5.0.6 Sikring af kamp\_tid

Her tjekkes det, om `kamp_tid` findes i baseline. Hvis den mangler, hentes tidspunktet fra et supplerende baseline-datasæt og joines ind, så hver kamp får et entydigt kickoff-tidspunkt. Det er nødvendigt for korrekt feature-opbygning og senere modellering.

```
# 3B) Sikr kamp_tid (hvis baseline ikke har den)
if (!("kamp_tid" %in% names(baseline))) {
  cat(" baseline_azure.rds mangler 'kamp_tid' - henter fra future_baseline_feature_azure.rds\n")
  fb_file <- file.path(baseline_dir, "future_baseline_feature_azure.rds")
  stopifnot(file.exists(fb_file))

  fb <- readRDS(fb_file) |>
    mutate(
      kamp_id = as.character(kamp_id),
      kamp_dato = as.Date(kamp_dato)
    ) |>
    select(kamp_id, kamp_tid)

  fb <- dedup_1row_by_id(fb, "kamp_id")
  baseline <- baseline |> left_join(fb, by = "kamp_id")
  cat(" kamp_tid joinet ind\n\n")
  rm(fb)
}
```

### 5.0.7 Oprettelse af kamp\_time\_h

Her standardiseres kamp\_tid til et ensartet tidsformat og omsættes til en numerisk timeværdi. Det gør kampstart sammenlignelig på tværs af kampe og klar til brug som forklarende variabel i modellerne.

```
# 3C) Opret kamp_time_h
baseline <- baseline |>
  mutate(
    kamp_tid_chr = str_trim(as.character(kamp_tid)),
    kamp_tid_chr = if_else(str_detect(kamp_tid_chr, "^\\d{1,2}:\\d{2}$"), paste0(kamp_tid_chr, ":00"),
    kamp_time_h = time_chr_to_hour(kamp_tid_chr)
  )

cat("Kamp-rækker uden parsebar kamp_time_h: ", sum(is.na(baseline$kamp_time_h)), "\n\n", sep = "")
```

Kamp-rækker uden parsebar kamp\_time\_h: 0

### 5.0.8 Features

#### 5.0.8.1 Helligdage

Her kobles officielle helligdage på kampdatoen, og der oprettes en binær indikator for, om kampen afvikles på en helligdag. Formålet er at indfange effekten af fridage, som kan påvirke tilskuertallet gennem ændret tilgængelighed og adfærd.

```
hellig_raw <- safe_dbReadTable(con, "PBA01_Raw", "dim_helligdage_dkk_raw")
stopifnot(all(c("dato", "helligdag_navn") %in% names(hellig_raw)))

hellig_min <- hellig_raw |>
  mutate(hellig_dato = as.Date(dato)) |>
  filter(!is.na(hellig_dato)) |>
  distinct(hellig_dato) |>
  transmute(hellig_dato, er_helligdag = 1L)

baseline <- baseline |>
  left_join(hellig_min, by = c("kamp_dato" = "hellig_dato")) |>
  mutate(er_helligdag = if_else(is.na(er_helligdag), 0L, er_helligdag))

cat("Helligdage joinet \n\n")
```

Helligdage joinet

#### 5.0.8.2 Håndboldkamp samme dag (SAH)

Her identificeres dage med SAH-håndboldkampe og der oprettes både en indikator for om der spilles håndbold samme dag samt et antal kampe. Variablen bruges til at fange potentiel konkurrence om publikums opmærksomhed på kampdagen.

```

sah_raw <- safe_dbReadTable(con, "PBA02_Clean", "fact_håndboldkampe_SAH_clean")
stopifnot(all(c("kamp_dato", "kamp_tid", "Event") %in% names(sah_raw)))

sah_min <- sah_raw |>
  mutate(sah_dato = as.Date(kamp_dato)) |>
  filter(!is.na(sah_dato)) |>
  group_by(sah_dato) |>
  summarise(
    er_håndboldkamp_SAH = 1L,
    antal_håndboldkampe = n(),
    .groups = "drop"
  )

baseline <- baseline |>
  left_join(sah_min, by = c("kamp_dato" = "sah_dato")) |>
  mutate(
    er_håndboldkamp_SAH = if_else(is.na(er_håndboldkamp_SAH), 0L, er_håndboldkamp_SAH),
    antal_håndboldkampe = if_else(is.na(antal_håndboldkampe), 0L, antal_håndboldkampe)
  )

cat("Håndbold SAH joinet \n\n")

```

Håndbold SAH joinet

### 5.0.8.3 Befolkning

Her kobles Viborgs befolkningstal på kampene ved at anvende det senest kendte kvartalstal før kampdatoen. Formålet er at give modellen et strukturelt mål for det potentielle publikum uden at introducere fremtidsinformation.

```

bef_raw <- safe_dbReadTable(con, "PBA03_JoinReady", "fact_Viborg_befolkning_join_ready")
stopifnot(all(c("køn", "civilstand", "dato", "befolkningstal") %in% names(bef_raw)))

bef_all <- bef_raw |>
  mutate(
    køn = str_to_lower(str_trim(as.character(køn))),
    civilstand = str_to_lower(str_trim(as.character(civilstand))),
    dato = as.Date(dato),
    befolkningstal = as.numeric(befolkningstal)
  ) |>
  filter(
    !is.na(dato),
    !is.na(befolkningstal),
    køn %in% c("i alt", "alt"),
    civilstand %in% c("i alt", "alt")
  ) |>
  distinct(dato, .keep_all = TRUE) |>
  arrange(dato) |>
  transmute(bef_dato = dato, befolkningstal = befolkningstal)

baseline_bef <- baseline |>
  transmute(kamp_id, kamp_dato) |>

```

```

left_join(bef_all, join_by(closest(kamp_dato >= bef_dato)))

baseline <- baseline |>
  left_join(baseline_bef |> select(kamp_id, befolkningstal), by = "kamp_id")

cat("Befolkning jointet    NA: ", sum(is.na(baseline$befolkningstal)), "\n\n", sep = "")

```

```
Befolkning jointet    NA: 123
```

#### 5.0.8.4 Vejr

Her kobles vejrobservationer på hver kamp ved at vælge den observation, der ligger tættest på kampstart blandt faste tidspunkter på dagen. Hvis en præcis match ikke findes, anvendes et simpelt fallback fra samme dato. Samtidig markeres manglende eller ikke-observerbare vejrdato eksplícit, så modellen kan håndtere usikkerhed uden at miste observationer.

```
cat("--- FEATURE: VEJR (nærmeste blandt 08/11/14/17) ---\n")
```

```
--- FEATURE: VEJR (nærmeste blandt 08/11/14/17) ---
```

```

vejr_sql <- safe_dbReadTable(con, "PBA03_JoinReady", "fact_vejr_join_ready")
stopifnot(all(c("obs_dato", "tid", "vejrkode", "vejrbeskrivelse") %in% names(vejr_sql)))

vejr_grid_tider_h <- c(8L, 11L, 14L, 17L)

vejr_pre <- vejr_sql |>
  mutate(
    vejr_dato = as_date_safely(obs_dato),
    vejr_tid_chr = str_trim(as.character(tid)),
    vejr_tid_chr = if_else(str_detect(vejr_tid_chr, "^\\d{1,2}:\\d{2}$"), paste0(vejr_tid_chr, ":00"),
    vejr_time_h = time_chr_to_hour(vejr_tid_chr),
    vejrkode = suppressWarnings(as.integer(vejrkode)),
    vejrbeskrivelse = as.character(vejrbeskrivelse)
  ) |>
  filter(!is.na(vejr_dato), !is.na(vejr_time_h))

vejr_grid <- vejr_pre |>
  filter(vejr_time_h %in% vejr_grid_tider_h) |>
  group_by(vejr_dato, vejr_time_h) |>
  slice(1) |>
  ungroup()

vejr_fallback_1 <- vejr_pre |>
  arrange(vejr_dato, vejr_time_h) |>
  group_by(vejr_dato) |>
  slice(1) |>
  ungroup()

baseline_key <- baseline |>
  transmute(kamp_id, kamp_dato, kamp_time_h)

```

```

kandidater_vejr <- baseline_key |>
  filter(!is.na(kamp_dato), !is.na(kamp_time_h)) |>
  inner_join(
    vejr_grid |> select(vejr_dato, vejr_time_h, vejrcode, vejrbeskrivelse),
    by = c("kamp_dato" = "vejr_dato")
  ) |>
  mutate(dist_hours_to_kickoff = abs(kamp_time_h - vejr_time_h)) |>
  group_by(kamp_id) |>
  arrange(dist_hours_to_kickoff, .by_group = TRUE) |>
  slice(1) |>
  ungroup() |>
  transmute(
    kamp_id,
    vejr_tid_h = vejr_time_h,
    vejrcode,
    vejrbeskrivelse,
    dist_hours_to_kickoff,
    vejr_match_type = "grid_nearest"
  )

fallback_vejr <- baseline_key |>
  anti_join(kandidater_vejr |> select(kamp_id), by = "kamp_id") |>
  left_join(
    vejr_fallback_1 |> select(vejr_dato, vejr_time_h, vejrcode, vejrbeskrivelse),
    by = c("kamp_dato" = "vejr_dato")
  ) |>
  transmute(
    kamp_id,
    vejr_tid_h = vejr_time_h,
    vejrcode,
    vejrbeskrivelse,
    dist_hours_to_kickoff = NA_integer_,
    vejr_match_type = "fallback_dato"
  )

vejr_match <- bind_rows(kandidater_vejr, fallback_vejr)
stopifnot(sum(duplicated(vejr_match$kamp_id)) == 0)

baseline <- baseline |> left_join(vejr_match, by = "kamp_id")

cat("Vejr joinet    NA vejrcode: ", sum(is.na(baseline$vejrcode)),
    "   vejrcode=0: ", sum(baseline$vejrcode == 0, na.rm = TRUE), "\n", sep = "")

```

Vejr joinet NA vejrcode: 38 vejrcode=0: 70

```

baseline <- baseline |>
  mutate(
    vejr_mangler_obs      = if_else(is.na(vejrcode), 1L, 0L),
    vejr_ikke_observerbar = if_else(!is.na(vejrcode) & vejrcode == 0, 1L, 0L),
    vejr_mangler_info     = if_else(vejr_mangler_obs == 1L | vejr_ikke_observerbar == 1L, 1L, 0L),

```

```

vejrkode_model      = if_else(vejr_mangler_info == 1L, -1L, as.integer(vejrkode)),
vejrbeskrivelse_model = case_when(
  vejr_mangler_obs == 1L      ~ "UKENDT",
  vejr_ikke_observerbar == 1L ~ "IKKE_OBSERVERBAR",
  TRUE                        ~ as.character(vejrbeskrivelse)
),
vejr_tid_h_model     = if_else(vejr_mangler_info == 1L, -1L, as.integer(vejr_tid_h)),
dist_hours_to_kickoff_model = if_else(vejr_mangler_info == 1L, -1L, as.integer(dist_hours_to_kickoff)),
vejr_match_type_model = if_else(is.na(vejr_match_type), "NO_WEATHER", as.character(vejr_match_type))
)

cat("VEJR: mangler_obs=", sum(baseline$vejr_mangler_obs == 1L, na.rm = TRUE),
    " ikke_observerbar=", sum(baseline$vejr_ikke_observerbar == 1L, na.rm = TRUE),
    " mangler_info=", sum(baseline$vejr_mangler_info == 1L, na.rm = TRUE), "\n\n", sep = "")

```

VEJR: mangler\_obs=38 ikke\_observerbar=70 mangler\_info=108

### 5.0.8.5 Temperatur

Her kobles temperatur på hver kamp ved at vælge den seneste observerede temperatur før kickoff samme dag. Der anvendes en fast bagud-trappe i tid, så temperaturen hentes fra kl. 18, 15, 12 eller 9 afhængigt af kampstart. Hvis ingen observation findes før kickoff, markeres dette eksplicit, så datatab undgås og usikkerhed håndteres systematisk i modellen.

```
cat("---- FEATURE: TEMPERATUR (bagud-trappe 18→15→12→09) ---\n")
```

--- FEATURE: TEMPERATUR (bagud-trappe 18→15→12→09) ---

```

temp_sql <- safe_dbReadTable(con, "PBA03_JoinReady", "fact_temperatur_join_ready")
stopifnot(all(c("obs_dato", "tid", "temperatur") %in% names(temp_sql)))

temp_tider_h <- c(9L, 12L, 15L, 18L)

temp_grid <- temp_sql |>
  mutate(
    temp_dato = as_date_safely(obs_dato),
    temp_tid_chr = str_trim(as.character(tid)),
    temp_tid_chr = if_else(str_detect(temp_tid_chr, "^\\d{1,2}:\\d{2}$"), paste0(temp_tid_chr, ":00"),
    temp_time_h = time_chr_to_hour(temp_tid_chr),
    temperatur = suppressWarnings(as.numeric(temperatur))
  ) |>
  filter(!is.na(temp_dato), !is.na(temp_time_h), temp_time_h %in% temp_tider_h) |>
  group_by(temp_dato, temp_time_h) |>
  slice(1) |>
  ungroup()

temp_kandidater <- baseline_key |>
  inner_join(
    temp_grid |> select(temp_dato, temp_time_h, temperatur),

```

```

    by = c("kamp_dato" = "temp_dato")
  ) |>
  filter(!is.na(kamp_time_h), temp_time_h <= kamp_time_h) |>
  mutate(temp_dist_hours_to_kickoff = kamp_time_h - temp_time_h)

temp_valgt <- temp_kandidater |>
  group_by(kamp_id) |>
  arrange(desc(temp_time_h), .by_group = TRUE) |>
  slice(1) |>
  ungroup() |>
  transmute(
    kamp_id,
    temp_tid_h = temp_time_h,
    temperatur,
    temp_dist_hours_to_kickoff = as.integer(temp_dist_hours_to_kickoff),
    temp_match_type = "step_back_same_day"
  )

temp_mangler <- baseline_key |>
  anti_join(temp_valgt |> select(kamp_id), by = "kamp_id") |>
  transmute(
    kamp_id,
    temp_tid_h = NA_integer_,
    temperatur = NA_real_,
    temp_dist_hours_to_kickoff = NA_integer_,
    temp_match_type = case_when(
      is.na(kamp_time_h) ~ "NO_KICKOFF_TIME",
      TRUE ~ "NO_TEMP_BEFORE_KICKOFF"
    )
  )

temp_match <- bind_rows(temp_valgt, temp_mangler)
stopifnot(sum(duplicated(temp_match$kamp_id)) == 0)

baseline <- baseline |>
  left_join(temp_match, by = "kamp_id") |>
  mutate(
    temp_mangler_obs = if_else(is.na(temperatur), 1L, 0L),
    temperatur_model = as.numeric(temperatur),
    temp_tid_h_model = as.integer(temp_tid_h),
    temp_dist_hours_to_kickoff_model = as.integer(temp_dist_hours_to_kickoff),
    temp_match_type_model = if_else(is.na(temp_match_type), "NO_TEMP", as.character(temp_match_type))
  )

cat("Temperatur jointet    NA temperatur: ", sum(is.na(baseline$temperatur)),
    "  step_back_same_day: ", sum(baseline$temp_match_type == "step_back_same_day", na.rm = TRUE),
    "  NO_KICKOFF_TIME: ", sum(baseline$temp_match_type == "NO_KICKOFF_TIME", na.rm = TRUE),
    "  NO_TEMP_BEFORE_KICKOFF: ", sum(baseline$temp_match_type == "NO_TEMP_BEFORE_KICKOFF", na.rm = TRUE),
    "\n\n", sep = "")

```

Temperatur jointet NA temperatur: 1 step\_back\_same\_day: 253 NO\_KICKOFF\_TIME: 0 NO\_TEMP\_BEFORE\_KICKOFF: 0

### 5.0.8.6 Billetsalg

Her indlæses billetsalgsdata fra Azure med fokus på D10-horisonten. For hver kamp reduceres data til én række ved at vælge det højeste registrerede billetsalg, så hvert kamp\_id kun optræder én gang og datasættet er konsistent til videre analyse og modellering.

```
vff_billetsalg <- safe_dbReadTable(con, "PBA03_JoinReady", "fact_VFF_Billetsalg_join_ready") |>
  mutate(
    kamp_id      = as.character(kamp_id),
    d10_tilskuere = as.numeric(d10_tilskuere)
  )

stopifnot(all(c("kamp_id", "d10_tilskuere") %in% names(vff_billetsalg)))
cat("Billetsalg hentet   Rækker: ", nrow(vff_billetsalg), "\n\n", sep = "")
```

Billetsalg hentet Rækker: 259

```
vff_billetsalg_1row <- vff_billetsalg |>
  group_by(kamp_id) |>
  summarise(
    d10_tilskuere = suppressWarnings(max(d10_tilskuere, na.rm = TRUE)),
    .groups = "drop"
  ) |>
  mutate(
    d10_tilskuere = if_else(is.infinite(d10_tilskuere), NA_real_, d10_tilskuere)
  )

cat("Billetsalg dedup (1 række pr kamp_id) \n\n")
```

Billetsalg dedup (1 række pr kamp\_id)

### 5.0.9 D10-datasæt og analysegrundlag

Her samles baseline og D10-billetsalg til ét samlet datasæt. Der køres et kort kvalitetstjek for at få overblik over mangler i nøgleværdier og features. Til sidst filtreres datasættet ned til de observationer, der er fuldt anvendelige, og `analysis_df_d10` klargøres som endeligt input til modellering.

```
tickets_feats_d10 <- vff_billetsalg_1row |>
  transmute(
    kamp_id,
    billetter_d10 = d10_tilskuere
  )

d10_dataset <- baseline |> left_join(tickets_feats_d10, by = "kamp_id")

cat("D10 join   Rækker: ", nrow(d10_dataset),
    "   Unikke kamp_id: ", n_distinct(d10_dataset$kamp_id), "\n\n", sep = "")
```

D10 join Rækker: 254 Unikke kamp\_id: 254

```
diag_d10 <- d10_dataset |>
  summarise(
    rækker_total = n(),
    unikke_kamp_id = n_distinct(kamp_id),
    uden_d10 = sum(is.na(billetter_d10)),
    uden_tilskuere = sum(is.na(tilskuere)),
    uden_time_h = sum(is.na(kamp_time_h)),
    uden_bef = sum(is.na(befolkningstal)),
    uden_vejrkode_raw = sum(is.na(vejrkode)),
    vejrkode0_raw = sum(vejrkode == 0, na.rm = TRUE),
    uden_temp_raw = sum(is.na(temperatur)),
    vejrg_grid_match = sum(vejrg_match_type == "grid_nearest", na.rm = TRUE),
    vejrg_fallback = sum(vejrg_match_type == "fallback_dato", na.rm = TRUE),
    temp_stepback = sum(temp_match_type == "step_back_same_day", na.rm = TRUE),
    temp_nomatch = sum(temp_match_type == "NO_TEMP_BEFORE_KICKOFF", na.rm = TRUE)
  )
print(diag_d10)
```

```
# A tibble: 1 x 13
  rækker_total unikke_kamp_id uden_d10 uden_tilskuere uden_time_h uden_bef
      <int>         <int>    <int>         <int>         <int>    <int>
1       254           254        0             0             0       123
# i 7 more variables: uden_vejrkode_raw <int>, vejrkode0_raw <int>,
#   uden_temp_raw <int>, vejrg_grid_match <int>, vejrg_fallback <int>,
#   temp_stepback <int>, temp_nomatch <int>
```

```
analysis_df_d10 <- d10_dataset |>
  filter(
    !is.na(billetter_d10),
    !is.na(kamp_time_h),
    !is.na(tilskuere),
    !is.na(sæson),
    !is.na(runde)
  ) |>
  mutate(
    vejrkode_model = if_else(is.na(vejrkode_model), -1L, vejrkode_model),
    vejrbeskrivelse_model = if_else(is.na(vejrbeskrivelse_model), "UKENDT", vejrbeskrivelse_model),
    vejrg_tid_h_model = if_else(is.na(vejrg_tid_h_model), -1L, vejrg_tid_h_model),
    dist_hours_to_kickoff_model = if_else(is.na(dist_hours_to_kickoff_model), -1L, dist_hours_to_kickoff_model),
    vejrg_match_type_model = if_else(is.na(vejrg_match_type_model), "NO_WEATHER", vejrg_match_type_model)

    temperatur_model = as.numeric(temperatur_model),
    temp_tid_h_model = as.integer(temp_tid_h_model),
    temp_dist_hours_to_kickoff_model = as.integer(temp_dist_hours_to_kickoff_model),
    temp_match_type_model = if_else(is.na(temp_match_type_model), "NO_TEMP", as.character(temp_match_type_model))

    billetter_d10 = as.numeric(billetter_d10)
  )

cat("analysis_df_d10 klar   Rækker: ", nrow(analysis_df_d10), "\n\n", sep = "")
```

analysis\_df\_d10 klar      Rækker: 254

```
# (valgfrit) manuel check - slå til hvis du vil
# View(analysis_df_d10)
# glimpse(analysis_df_d10)
```

### 5.0.10 Testpipeline for D10-modellen

Her opstilles en enkel nulmodel baseret på sæson, runde, kampstartstidspunkt og billetsalg ti dage før kamp. Derefter testes ekstra variable én ad gangen under samme datagrundlag, så sammenligningen er fair og effekten kan vurderes isoleret.

#### 5.0.10.1 Hjælpefunktioner til modeltest

Disse funktioner bruges til at udtrække p-værdier fra modellerne. Den ene henter p-værdien fra en ANOVA-sammenligning mellem to modeller, og den anden henter p-værdien for en specifik variabel direkte fra modelkoefficienterne. Formålet er at kunne vurdere statistisk betydning på en ensartet og automatiseret måde.

```
safe_p_from_anova <- function(a) suppressWarnings(as.numeric(a[nrow(a), ncol(a)]))

coef_p <- function(mod, term) {
  sm <- summary(mod)
  cf <- sm$coefficients
  if (!(term %in% rownames(cf))) return(NA_real_)
  suppressWarnings(as.numeric(cf[term, "Pr(>|t|)"]))
}
```

#### 5.0.10.2 Enkelt variabeltest mod nulmodel

Denne funktion tester én ekstra variabel ad gangen mod nulmodellen. Den sikrer, at begge modeller estimeres på præcis samme datasæt, så sammenligningen er fair. Resultatet opsummerer antal observationer, p-værdier og ændring i AIC, hvilket gør det muligt systematisk at vurdere, om variablen bidrager med reel forklaringskraft.

```
safe_p_from_anova <- function(a) suppressWarnings(as.numeric(a[nrow(a), ncol(a)]))

coef_p <- function(mod, term) {
  sm <- summary(mod)
  cf <- sm$coefficients
  if (!(term %in% rownames(cf))) return(NA_real_)
  suppressWarnings(as.numeric(cf[term, "Pr(>|t|)"]))
}

run_single_test <- function(df_base, extra_var, label = extra_var) {
  need_cols <- c("tilskuere", "sæson", "runde", "kamp_time_h", "billetter_d10", extra_var)
  miss <- setdiff(need_cols, names(df_base))
  if (length(miss) > 0) stop(" Mangler kolonner i df_base: ", paste(miss, collapse = ", "))

  df_common <- df_base |>
    dplyr::transmute(
      tilskuere = as.numeric(tilskuere),
```

```

    sæson          = as.factor(sæson),
    runde          = as.integer(runde),
    kamp_time_h    = as.integer(kamp_time_h),
    billetter_d10  = as.numeric(billetter_d10),
    extra          = .data[[extra_var]]
  ) |>
  dplyr::filter(
    !is.na(tilskuere),
    !is.na(sæson),
    !is.na(runde),
    !is.na(kamp_time_h),
    !is.na(billetter_d10),
    !is.na(extra)
  )

m0 <- lm(tilskuere ~ sæson + runde + kamp_time_h + billetter_d10, data = df_common)
m1 <- lm(tilskuere ~ sæson + runde + kamp_time_h + billetter_d10 + extra, data = df_common)
a <- anova(m0, m1)

out <- tibble::tibble(
  variabel          = label,
  n_test           = nrow(df_common),
  p_koefficient    = coef_p(m1, "extra"),
  p_anova          = safe_p_from_anova(a),
  delta_AIC        = as.numeric(AIC(m1) - AIC(m0)),
  AIC_nul          = as.numeric(AIC(m0)),
  AIC_plus         = as.numeric(AIC(m1))
)

list(out = out, df_common = df_common, m0 = m0, m1 = m1, anova = a)
}

```

### 5.0.11 Klargøring af D10-basisdatasæt

Her udvælges og typesikres de variable, der indgår i D10-analyserne. Datasættet reduceres til én konsistent struktur med både basisforklarende variable og potentielle testvariable. Til sidst filtreres der til et rent nulmodel-datasæt uden manglende værdier, som bruges som fælles udgangspunkt for alle efterfølgende model- og variabeltests.

```

df_d10_base <- analysis_df_d10 |>
  dplyr::transmute(
    kamp_id        = as.character(kamp_id),
    tilskuere      = as.numeric(tilskuere),
    sæson          = as.factor(sæson),
    runde          = as.integer(runde),
    kamp_time_h    = as.integer(kamp_time_h),
    billetter_d10  = as.numeric(billetter_d10),

    er_helligdag   = as.integer(er_helligdag),
    er_håndboldkamp_SAH = as.integer(er_håndboldkamp_SAH),
    temperatur_model = as.numeric(temperatur_model),
    vejr_mangler_info = as.integer(vejr_mangler_info),

```

```

    befolkningstal      = as.numeric(befolkningstal)
  )
df_d10_0 <- df_d10_base |>
  dplyr::filter(
    !is.na(tilskuere),
    !is.na(sæson),
    !is.na(runde),
    !is.na(kamp_time_h),
    !is.na(billetter_d10)
  )

```

### 5.0.12 D10 nulmodel

Her estimeres den grundlæggende D10-model, som forklarer det endelige tilskuertal ud fra sæson, runde, kampstartstidspunkt og billetsalg ti dage før kamp. Denne model fungerer som reference, når ekstra variable efterfølgende vurderes.

```
cat("D10 base datasæt klar    N=", nrow(df_d10_0), "\n", sep = "")
```

```
D10 base datasæt klar    N=254
```

```
m_d10_0 <- lm(tilskuere ~ sæson + runde + kamp_time_h + billetter_d10, data = df_d10_0)
```

### 5.0.13 Test af ekstra forklarende variable

Her udvides D10-nulmodellen én variabel ad gangen. For hver test sammenlignes nulmodellen med en udvidet model på samme datagrundlag, så effekten kan vurderes fair via ANOVA og ændring i AIC. Formålet er at afgøre, om helligdage, håndboldkampe, temperatur, manglende vejrdato eller sportslig placering giver en reel forbedring af modellen i forhold til den simple baseline.

```
cat("\n===== \n")
```

```
=====
```

```
cat("MODEL_D10_00 - NULMODEL (D10)\n")
```

```
MODEL_D10_00 - NULMODEL (D10)
```

```
cat("===== \n")
```

```
=====
```

```
print(summary(m_d10_0))
```

```
Call:
lm(formula = tilskuere ~ sæson + runde + kamp_time_h + billetter_d10,
    data = df_d10_0)
```

Residuals:

Min	1Q	Median	3Q	Max
-1147.83	-301.91	-21.44	308.36	1671.04

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	-618.08980	323.76729	-1.909	0.057473 .
sæson2001/2002	126.70766	175.96625	0.720	0.472199
sæson2002/2003	452.05296	180.35585	2.506	0.012872 *
sæson2003/2004	235.71084	183.44027	1.285	0.200076
sæson2004/2005	235.02123	187.20901	1.255	0.210582
sæson2005/2006	484.18931	185.49260	2.610	0.009628 **
sæson2006/2007	280.09631	180.07029	1.555	0.121177
sæson2007/2008	441.25096	183.55990	2.404	0.016999 *
sæson2013/2014	290.87257	191.06028	1.522	0.129250
sæson2015/2016	475.08419	182.60641	2.602	0.009866 **
sæson2016/2017	116.04342	177.68988	0.653	0.514351
sæson2021/2022	342.24218	186.35790	1.836	0.067550 .
sæson2022/2023	423.04079	198.18583	2.135	0.033832 *
sæson2023/2024	580.43075	195.35579	2.971	0.003275 **
sæson2024/2025	358.02784	198.14708	1.807	0.072060 .
sæson2025/2026	760.96777	223.71264	3.402	0.000787 ***
runde	2.99042	3.50872	0.852	0.394926
kamp_time_h	46.08687	19.10927	2.412	0.016644 *
billetter_d10	1.99540	0.04941	40.386	< 0.0000000000000002 ***

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 511.4 on 235 degrees of freedom

Multiple R-squared: 0.9298, Adjusted R-squared: 0.9244

F-statistic: 173 on 18 and 235 DF, p-value: < 0.00000000000000022

```
cat("\nAIC nulmodel:", as.numeric(AIC(m_d10_0)), "\n\n")
```

AIC nulmodel: 3909.574

```
results_list <- list()
```

```
cat("=====\n")
```

=====

```
cat("MODEL_D10_01 - TEST: er_helligdag\n")
```

MODEL\_D10\_01 - TEST: er\_helligdag

```
cat("=====\n")
```

=====

```
t_holiday <- run_single_test(df_d10_base, extra_var = "er_helligdag", label = "er_helligdag")
print(t_holiday$anova)
```

Analysis of Variance Table

```
Model 1: tilskuere ~ sæson + runde + kamp_time_h + billetter_d10
Model 2: tilskuere ~ sæson + runde + kamp_time_h + billetter_d10 + extra
```

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	235	61466298				
2	234	61422800	1	43498	0.1657	0.6843

```
results_list[["er_helligdag"]] <- t_holiday$out
```

```
cat("\n=====\n")
```

=====

```
cat("MODEL_D10_02 - TEST: er_håndboldkamp_SAH\n")
```

MODEL\_D10\_02 - TEST: er\_håndboldkamp\_SAH

```
cat("=====\n")
```

=====

```
t_sah <- run_single_test(df_d10_base, extra_var = "er_håndboldkamp_SAH", label = "er_håndboldkamp_SAH")
print(t_sah$anova)
```

Analysis of Variance Table

```
Model 1: tilskuere ~ sæson + runde + kamp_time_h + billetter_d10
Model 2: tilskuere ~ sæson + runde + kamp_time_h + billetter_d10 + extra
```

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	235	61466298				
2	234	61447913	1	18386	0.07	0.7915

```
results_list[["er_håndboldkamp_SAH"]] <- t_sah$out
```

```
cat("\n=====\n")
```

=====

```
cat("MODEL_D10_03 - TEST: temperatur_model\n")
```

MODEL\_D10\_03 - TEST: temperatur\_model

```
cat("=====\n")
```

=====

```
t_temp <- run_single_test(df_d10_base, extra_var = "temperatur_model", label = "temperatur_model")
print(t_temp$anova)
```

#### Analysis of Variance Table

Model 1: tilskuere ~ sæson + runde + kamp\_time\_h + billetter\_d10

Model 2: tilskuere ~ sæson + runde + kamp\_time\_h + billetter\_d10 + extra

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	234	60832717				
2	233	60827831	1	4885.6	0.0187	0.8913

```
results_list[["temperatur_model"]] <- t_temp$out
```

```
cat("\n=====\n")
```

=====

```
cat("MODEL_D10_04 - TEST: vejr_mangler_info\n")
```

MODEL\_D10\_04 - TEST: vejr\_mangler\_info

```
cat("=====\n")
```

=====

```
t_weather_miss <- run_single_test(df_d10_base, extra_var = "vejr_mangler_info", label = "vejr_mangler_info")
print(t_weather_miss$anova)
```

#### Analysis of Variance Table

Model 1: tilskuere ~ sæson + runde + kamp\_time\_h + billetter\_d10

Model 2: tilskuere ~ sæson + runde + kamp\_time\_h + billetter\_d10 + extra

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	235	61466298				
2	234	61383271	1	83027	0.3165	0.5743

```
results_list[["vejr_mangler_info"]] <- t_weather_miss$out
```

D10-nulmodellen er estimeret med sæson, runde, kampstartstidspunkt og billetsalg ti dage før kamp som forklarende variable. Modellen forklarer en meget stor andel af variationen i tilskuertallet med en justeret forklaringsgrad på 0,924 og en residual standard error på ca. 511 tilskuere. Samlet set er modellen stærkt signifikant, og særligt billetter\_d10 fremstår som den klart dominerende forklarende variabel med en høj og meget signifikant effekt.

Kampstartstidspunkt har ligeledes en signifikant positiv effekt, mens runde ikke bidrager signifikant i denne specifikation. Flere sæson-dummys er signifikante, hvilket indikerer strukturelle forskelle i tilskuerniveau mellem sæsoner, som ikke alene kan forklares af de øvrige variable.

På baggrund af nulmodellen er der gennemført enkeltvariabeltests, hvor hver ekstra variabel er tilføjet én ad gangen og evalueret via ANOVA. Resultaterne viser, at er\_helligdag, er\_håndboldkamp\_SAH, temperatur\_model og vejr\_mangler\_info ikke bidrager med en signifikant reduktion i residual sum of squares. For samtlige tests er p-værdierne høje, og der observeres ingen meningsfuld forbedring af modellens forklaringskraft.

Samlet set indikerer resultaterne, at nulmodellen allerede indfanger de væsentligste systematiske variationer i D10-horisonten, og at de testede ekstra variable ikke tilfører selvstændig forklaringsværdi. På denne baggrund fravælges disse variable i den videre modellering.

### 5.0.13.1 Test af placering før kamp

Her tilføjes holdets placering før kampen til D10-nulmodellen. Placeringen joines på kamp\_id og reduceres til én observation pr. kamp, hvorefter modellen sammenlignes med nulmodellen på samme datasæt. Formålet er at vurdere, om den sportslige situation før kickoff bidrager med forklaringskraft ud over billetsalg og kampens rammer.

```
cat("\n===== \n")
```

```
=====
```

```
cat("MODEL_D10_05 - TEST: placering_før_kamp\n")
```

```
MODEL_D10_05 - TEST: placering_før_kamp
```

```
cat("===== \n")
```

```
=====
```

```
plac_raw <- safe_dbReadTable(con, "PBA02_Clean", "fact_vff_rundeplaceringer_clean") |>
  dplyr::mutate(kamp_id = as.character(kamp_id))

need_cols <- c("kamp_id", "placering_før_kamp")
miss <- setdiff(need_cols, names(plac_raw))
if (length(miss) > 0) stop(" Mangler kolonner i placeringstabellen: ", paste(miss, collapse = ", "))

plac_1row <- plac_raw |>
  dplyr::transmute(
    kamp_id = as.character(kamp_id),
    placering_før_kamp = suppressWarnings(as.integer(placering_før_kamp))
```

```

) |>
dplyr::arrange(kamp_id) |>
dplyr::group_by(kamp_id) |>
dplyr::slice(1) |>
dplyr::ungroup()

df_d10_pos <- df_d10_base |>
  dplyr::left_join(plac_1row, by = "kamp_id")

t_pos <- run_single_test(df_d10_pos, extra_var = "placering_før_kamp", label = "placering_før_kamp")
print(t_pos$anova)

```

### Analysis of Variance Table

```

Model 1: tilskuere ~ sæson + runde + kamp_time_h + billetter_d10
Model 2: tilskuere ~ sæson + runde + kamp_time_h + billetter_d10 + extra
  Res.Df      RSS Df Sum of Sq    F Pr(>F)
1     230 60552807
2     229 59980656   1    572151 2.1844 0.1408

```

```
results_list[["placering_før_kamp"]] <- t_pos$out
```

Placering før kamp er testet som ekstra forklarende variabel i D10-modellen ved at blive tilføjet nulmodellen bestående af sæson, runde, kampstartstidspunkt og billetsalg ti dage før kamp.

ANOVA-testen viser, at variablen reducerer residual sum of squares marginalt, men effekten er ikke statistisk signifikant. Den observerede F-statistik på 2,18 svarer til en p-værdi på 0,141, hvilket ligger klart over det sædvanlige signifikansniveau.

Resultatet indikerer, at placering før kamp ikke tilfører selvstændig forklaringskraft, når der allerede kontrolleres for sæsonstruktur og det observerede billetsalg på D10-horisonten. Variablen fravælges derfor i den videre modellering.

### 5.0.13.2 Test af befolkningstal i D10-modellen (cost-benefit)

Befolkningstal testes her som potentiel ekstra forklarende variabel i D10-modellen. Analysen kombinerer en klassisk enkeltvariabeltest med en eksplisit cost-benefit-vurdering, hvor både statistisk effekt og tab af observationer indgår. Ud over ANOVA-resultatet evalueres variablen også via out-of-sample performance på et fælles datasæt for at vurdere, om en eventuel effekt er robust og operationelt relevant.

```
cat("\n===== \n")
```

```
=====
```

```
cat("MODEL_D10_06 - TEST: befolkningstal (cost vs benefit)\n")
```

```
MODEL_D10_06 - TEST: befolkningstal (cost vs benefit)
```

```
cat("=====\n")
```

```
=====
```

```
df_d10_full <- df_d10_base |>
  dplyr::filter(
    !is.na(tilskuere),
    !is.na(sæson),
    !is.na(runde),
    !is.na(kamp_time_h),
    !is.na(billetter_d10)
  )

df_d10_common_pop <- df_d10_base |>
  dplyr::filter(
    !is.na(tilskuere),
    !is.na(sæson),
    !is.na(runde),
    !is.na(kamp_time_h),
    !is.na(billetter_d10),
    !is.na(befolkningstal)
  )

cat("Data-omkostning:\n")
```

Data-omkostning:

```
cat("FULL N=", nrow(df_d10_full), "\n", sep = "")
```

FULL N=254

```
cat("COMMON N=", nrow(df_d10_common_pop), "\n", sep = "")
```

COMMON N=131

```
cat("Mistede rækker=", nrow(df_d10_full) - nrow(df_d10_common_pop), "\n\n", sep = "")
```

Mistede rækker=123

```
t_pop <- run_single_test(df_d10_base, extra_var = "befolkningstal", label = "befolkningstal")
print(t_pop$anova)
```

Analysis of Variance Table

Model 1: tilskuere ~ sæson + runde + kamp\_time\_h + billetter\_d10

Model 2: tilskuere ~ sæson + runde + kamp\_time\_h + billetter\_d10 + extra

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	119	35591024				
2	118	35576661	1	14363	0.0476	0.8276

```

results_list[["befolkningstal"]] <- t_pop$out

set.seed(42)
split_common <- rsample::initial_split(df_d10_common_pop, prop = 0.80)
train_common <- rsample::training(split_common)
test_common <- rsample::testing(split_common)

m_no_pop_common <- lm(tilskuere ~ sæson + runde + kamp_time_h + billetter_d10, data = train_common)
m_pop_common <- lm(tilskuere ~ sæson + runde + kamp_time_h + billetter_d10 + befolkningstal, data = train_common)

pred_ref <- predict(m_no_pop_common, newdata = test_common)
pred_pop <- predict(m_pop_common, newdata = test_common)

cat("\nOut-of-sample (COMMON)\n")

```

Out-of-sample (COMMON)

```

cat("Uden befolkning: RMSE=", rmse_vec(test_common$tilskuere, pred_ref),
    " MAE=", mae_vec(test_common$tilskuere, pred_ref),
    " R2=", r2_vec(test_common$tilskuere, pred_ref), "\n", sep = "")

```

Uden befolkning: RMSE=538.2499 MAE=421.7079 R2=0.902069

```

cat("Med befolkning: RMSE=", rmse_vec(test_common$tilskuere, pred_pop),
    " MAE=", mae_vec(test_common$tilskuere, pred_pop),
    " R2=", r2_vec(test_common$tilskuere, pred_pop), "\n\n", sep = "")

```

Med befolkning: RMSE=566.9792 MAE=467.5427 R2=0.8913358

Inklusion af befolkningstal i D10-modellen medfører en betydelig data-omkostning, idet antallet af observationer reduceres fra 254 til 131. Dette svarer til et tab på næsten halvdelen af datagrundlaget.

Den statistiske test viser, at befolkningstal ikke har nogen signifikant effekt på tilskuertallet. ANOVA-testen giver en meget lav F-værdi og en høj p-værdi, hvilket indikerer, at variabelen ikke bidrager med selvstændig forklaringskraft, når der allerede kontrolleres for sæson, runde, kampstartstidspunkt og billetsalg ti dage før kamp.

Out-of-sample evalueringen på det fælles datasæt understøtter dette resultat. Modellen med befolkningstal præsterer dårligere end reference-modellen uden variabelen, med højere RMSE og MAE samt lavere  $R^2$ .

Samlet set overstiger omkostningen i tabt data klart den potentielle gevinst, og befolkningstal vurderes derfor ikke at være en hensigtsmæssig variabel i D10-modellen.

```

variabeltest_oversigt_d10 <- dplyr::bind_rows(results_list) |>
  dplyr::mutate(
    beslutning = dplyr::case_when(
      is.na(p_anova) ~ "Ikke testet",
      p_anova < 0.05 & delta_AIC < 0 ~ "Behold",
      TRUE ~ "Forkast"
    ),

```

```

    begrundelse = dplyr::case_when(
      variabel == "befolkningstal" ~ "Ingen robust effekt og stor data-omkostning",
      beslutning == "Behold" ~ "Signifikant forbedring og lavere AIC",
      TRUE ~ "Ingen signifikant forbedring eller AIC bliver dårligere"
    )
  ) |>
  dplyr::select(variabel, n_test, p_koefficient, p_anova, delta_AIC, beslutning, begrundelse) |>
  dplyr::arrange(delta_AIC)

cat("=====\n")

```

```

cat("VARIABLETEST-OVERSIGT (D10)\n")

```

```

VARIABLETEST-OVERSIGT (D10)

```

```

cat("=====\n")

```

```

print(variabeltest_oversigt_d10)

```

```

# A tibble: 6 x 7
  variabel      n_test p_koefficient p_anova delta_AIC beslutning begrundelse
  <chr>      <int>      <dbl>    <dbl>    <dbl> <chr>      <chr>
1 placering_før_k~    249      0.141    0.141    -0.364 Forkast   Ingen sign~
2 vejr_mangler_in~    254      0.574    0.574     1.66 Forkast   Ingen sign~
3 er_helligdag        254      0.684    0.684     1.82 Forkast   Ingen sign~
4 er_håndboldkamp~    254      0.792    0.792     1.92 Forkast   Ingen sign~
5 befolkningstal     131      0.828    0.828     1.95 Forkast   Ingen robu~
6 temperatur_model    253      0.891    0.891     1.98 Forkast   Ingen sign~

```

```

cat("\n")

```

Samtlige testede ekstra variable forkastes i D10-modellen. Ingen af variableerne udviser statistisk signifikant effekt, og ingen bidrager med en konsistent forbedring af modelkvaliteten målt ved AIC.

Flere variable medfører desuden en forværring af modellens informationskriterium, og i tilfældet **befolkningstal** er der samtidig en betydelig data-omkostning i form af et stort tab af observationer, uden tilsvarende gevinst i forklarings- eller prædiktiv styrke.

Resultaterne understøtter, at D10-nulmodellen allerede indfanger de væsentligste forklaringsmekanismer gennem sæsonstruktur, kampkarakteristika og observeret billetsalg. Af hensyn til modelrobusthed, generaliserbarhed og operationel anvendelighed fastholdes derfor en **lean D10-model uden de testede ekstra variable**.

```

cat("\n=====\n")

```

```

=====

```

```
cat("MODEL_D10_06 - TEST: befolkningstal (cost vs benefit)\n")
```

MODEL\_D10\_06 - TEST: befolkningstal (cost vs benefit)

```
cat("=====\n")
```

=====

```
df_d10_full <- df_d10_base |>
  dplyr::filter(
    !is.na(tilskuere),
    !is.na(sæson),
    !is.na(runde),
    !is.na(kamp_time_h),
    !is.na(billetter_d10)
  )

df_d10_common_pop <- df_d10_base |>
  dplyr::filter(
    !is.na(tilskuere),
    !is.na(sæson),
    !is.na(runde),
    !is.na(kamp_time_h),
    !is.na(billetter_d10),
    !is.na(befolkningstal)
  )

cat("Data-omkostning:\n")
```

Data-omkostning:

```
cat("FULL N=", nrow(df_d10_full), "\n", sep = "")
```

FULL N=254

```
cat("COMMON N=", nrow(df_d10_common_pop), "\n", sep = "")
```

COMMON N=131

```
cat("Mistede rækker=", nrow(df_d10_full) - nrow(df_d10_common_pop), "\n\n", sep = "")
```

Mistede rækker=123

```
t_pop <- run_single_test(df_d10_base, extra_var = "befolkningstal", label = "befolkningstal")
print(t_pop$anova)
```

## Analysis of Variance Table

Model 1: tilskuere ~ sæson + runde + kamp\_time\_h + billetter\_d10

Model 2: tilskuere ~ sæson + runde + kamp\_time\_h + billetter\_d10 + extra

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	119	35591024				
2	118	35576661	1	14363	0.0476	0.8276

```
results_list[["befolkningstal"]] <- t_pop$out
```

```
set.seed(42)
```

```
split_common <- rsample::initial_split(df_d10_common_pop, prop = 0.80)
```

```
train_common <- rsample::training(split_common)
```

```
test_common <- rsample::testing(split_common)
```

```
m_no_pop_common <- lm(tilskuere ~ sæson + runde + kamp_time_h + billetter_d10, data = train_common)
```

```
m_pop_common <- lm(tilskuere ~ sæson + runde + kamp_time_h + billetter_d10 + befolkningstal, data =
```

```
pred_ref <- predict(m_no_pop_common, newdata = test_common)
```

```
pred_pop <- predict(m_pop_common, newdata = test_common)
```

```
cat("\nOut-of-sample (COMMON)\n")
```

Out-of-sample (COMMON)

```
cat("Uden befolkning: RMSE=", rmse_vec(test_common$tilskuere, pred_ref),  
    " MAE=", mae_vec(test_common$tilskuere, pred_ref),  
    " R2=", r2_vec(test_common$tilskuere, pred_ref), "\n", sep = "")
```

Uden befolkning: RMSE=538.2499 MAE=421.7079 R2=0.902069

```
cat("Med befolkning: RMSE=", rmse_vec(test_common$tilskuere, pred_pop),  
    " MAE=", mae_vec(test_common$tilskuere, pred_pop),  
    " R2=", r2_vec(test_common$tilskuere, pred_pop), "\n\n", sep = "")
```

Med befolkning: RMSE=566.9792 MAE=467.5427 R2=0.8913358

```
variabeltest_oversigt_d10 <- dplyr::bind_rows(results_list) |>  
  dplyr::mutate(  
    beslutning = dplyr::case_when(  
      is.na(p_anova) ~ "Ikke testet",  
      p_anova < 0.05 & delta_AIC < 0 ~ "Behold",  
      TRUE ~ "Forkast"  
    ),  
    begrundelse = dplyr::case_when(  
      variabel == "befolkningstal" ~ "Ingen robust effekt og stor data-omkostning",  
      beslutning == "Behold" ~ "Signifikant forbedring og lavere AIC",  
      TRUE ~ "Ingen signifikant forbedring eller AIC bliver dårligere"
```

```
)
) |>
dplyr::select(variabel, n_test, p_koefficient, p_anova, delta_AIC, beslutning, begrundelse) |>
dplyr::arrange(delta_AIC)

cat("=====\n")
```

```
=====
```

```
cat("VARIABLETEST-OVERSIGT (D10)\n")
```

```
VARIABLETEST-OVERSIGT (D10)
```

```
cat("=====\n")
```

```
=====
```

```
print(variabeltest_oversigt_d10)
```

```
# A tibble: 6 x 7
  variabel      n_test p_koefficient p_anova delta_AIC beslutning begrundelse
  <chr>        <int>      <dbl>    <dbl>    <dbl> <chr>      <chr>
1 placering_før_k~    249      0.141    0.141    -0.364 Forkast   Ingen sign~
2 vejr_mangler_in~    254      0.574    0.574     1.66 Forkast   Ingen sign~
3 er_helligdag        254      0.684    0.684     1.82 Forkast   Ingen sign~
4 er_håndboldkamp~    254      0.792    0.792     1.92 Forkast   Ingen sign~
5 befolkningstal      131      0.828    0.828     1.95 Forkast   Ingen robu~
6 temperatur_model    253      0.891    0.891     1.98 Forkast   Ingen sign~
```

```
cat("\n")
```

Alle testede ekstra variable i D10-modellen forkastes. Ingen af variablerne opnår statistisk signifikans, og samtlige medfører enten ingen forbedring eller en forværring af modelkvaliteten målt. For befolkningstal ses desuden en markant data-omkostning, hvor antallet af observationer reduceres fra 254 til 131.

Out-of-sample evalueringen understøtter resultaterne fra ANOVA- testene. Modellen med befolkningstal præsterer dårligere end reference-modellen uden variabelen, med højere RMSE og MAE samt lavere  $R^2$ .

D10-nulmodellen vurderes at være tilstrækkelig og robust. Af hensyn til modelkvalitet, datadækning og operationel anvendelighed fastholdes en lean D10-model uden de testede ekstra variable.

## 5.1 OLS & Ridge & Lasso

Datasættet klargøres som det endelige analysegrundlag for D10-modellen, hvor relevante variable er udvalgt, datatyper standardiseret og observationer med manglende værdier fjernet. Resultatet er et konsistent, kronologisk sorteret datasæt med én række pr. kamp, klar til modellering.

```

stopifnot(exists("analysis_df_d10"))

df_d10_final <- analysis_df_d10 |>
  dplyr::transmute(
    kamp_id      = as.character(kamp_id),
    kamp_dato    = as.Date(kamp_dato),
    tilskuere    = as.numeric(tilskuere),

    sæson_chr    = as.character(sæson),
    sæson_start  = season_start_year(sæson_chr),

    runde        = as.integer(runde),
    kamp_time_h  = as.integer(kamp_time_h),
    billetter_d10 = as.numeric(billetter_d10)
  ) |>
  dplyr::filter(
    !is.na(kamp_dato),
    !is.na(tilskuere),
    !is.na(sæson_start),
    !is.na(runde),
    !is.na(kamp_time_h),
    !is.na(billetter_d10)
  ) |>
  dplyr::arrange(kamp_dato)

cat(
  "df_d10_final klar    N=", nrow(df_d10_final),
  "   Periode=", as.character(min(df_d10_final$kamp_dato)), "→", as.character(max(df_d10_final$kamp_dato)),
  "   Sæsoner=", dplyr::n_distinct(df_d10_final$sæson_chr), "\n\n",
  sep = ""
)

```

```
df_d10_final klar    N=254   Periode=2000-07-30→2025-12-07   Sæsoner=16
```

### 5.1.1 Tidsbaseret trænings- og testsplit (D10)

Datasættet opdeles i et tidsbaseret trænings- og testsæt, hvor de seneste 20 % af observationerne anvendes som testdata. Opdelingen sikrer, at modellen evalueres på fremtidige kampe i forhold til træningsperioden og undgår informationslækage, samtidig med at der altid bevares mindst én observation i testsættet.

```

set.seed(42)
test_prop <- 0.20

n_total <- nrow(df_d10_final)
n_test  <- max(1L, as.integer(floor(n_total * test_prop)))
cut_idx <- max(1L, n_total - n_test)

train_d10 <- df_d10_final[seq_len(cut_idx), , drop = FALSE]
test_d10  <- df_d10_final[(cut_idx + 1L):n_total, , drop = FALSE]

```

```

if (nrow(test_d10) == 0) {
  n_test <- 1L
  cut_idx <- n_total - n_test
  train_d10 <- df_d10_final[seq_len(cut_idx), , drop = FALSE]
  test_d10 <- df_d10_final[(cut_idx + 1L):n_total, , drop = FALSE]
}

cat("Tidssplit klar   Train=", nrow(train_d10), "   Test=", nrow(test_d10),
    "   Test starter=", as.character(min(test_d10$kamp_dato)), "\n\n", sep = "")

```

Tidssplit klar Train=204 Test=50 Test starter=2022-11-06

### 5.1.2 Estimering af D10 OLS-baseline

D10-nulmodellen estimeres som en lineær regressionsmodel på træningsdata og evalueres på testsættet ved hjælp af RMSE, MAE og  $R^2$ . Samme modelspecifikation anvendes til at konstruere designmatricer for trænings- og testdata, hvilket sikrer konsistens i den efterfølgende modellering og performance-sammenligning.

```

f_d10 <- tilskuere ~ sæson_start + runde + kamp_time_h + billetter_d10

m_lm <- lm(f_d10, data = train_d10)
pred_lm <- as.numeric(predict(m_lm, newdata = test_d10))

perf_lm <- tibble::tibble(
  model = "OLS (lm)",
  RMSE = rmse_vec(test_d10$tilskuere, pred_lm),
  MAE = mae_vec(test_d10$tilskuere, pred_lm),
  R2 = r2_vec(test_d10$tilskuere, pred_lm)
)

x_train <- stats::model.matrix(f_d10, data = train_d10)[, -1, drop = FALSE]
y_train <- train_d10$tilskuere
x_test <- stats::model.matrix(f_d10, data = test_d10)[, -1, drop = FALSE]
y_test <- test_d10$tilskuere

```

### 5.1.3 Regulariserede modeller: Ridge og Lasso

Ridge- og Lasso-modeller estimeres ved hjælp af 10-fold krydsvalidering, hvor den optimale regulariseringsparameter vælges automatisk. Modellerne evalueres på testsættet med samme performance-mål som OLS, hvilket muliggør en direkte og fair sammenligning af prædiktiv kvalitet på D10-horisonten.

```

set.seed(42)
cv_ridge <- glmnet::cv.glmnet(x_train, y_train, alpha = 0, nfolds = 10, standardize = TRUE)
pred_ridge <- as.numeric(predict(cv_ridge, newx = x_test, s = "lambda.min"))

set.seed(42)
cv_lasso <- glmnet::cv.glmnet(x_train, y_train, alpha = 1, nfolds = 10, standardize = TRUE)
pred_lasso <- as.numeric(predict(cv_lasso, newx = x_test, s = "lambda.min"))

```

```

perf_glmnet <- tibble::tibble(
  model = c("Ridge (cv.glmnet)", "Lasso (cv.glmnet)"),
  RMSE = c(rmse_vec(y_test, pred_ridge), rmse_vec(y_test, pred_lasso)),
  MAE = c(mae_vec(y_test, pred_ridge), mae_vec(y_test, pred_lasso)),
  R2 = c(r2_vec(y_test, pred_ridge), r2_vec(y_test, pred_lasso)),
  lambda_min = c(cv_ridge$lambda.min, cv_lasso$lambda.min)
)

perf_d10 <- dplyr::bind_rows(perf_lm, perf_glmnet) |>
  dplyr::arrange(RMSE)

```

```
cat("=====\n")
```

```
=====
```

```
cat("D10 - SAMLET TEST PERFORMANCE\n")
```

```
D10 - SAMLET TEST PERFORMANCE
```

```
cat("=====\n")
```

```
=====
```

```
print(perf_d10)
```

```

# A tibble: 3 x 5
  model      RMSE  MAE  R2 lambda_min
  <chr>      <dbl> <dbl> <dbl>      <dbl>
1 OLS (lm)    634.  526. 0.692         NA
2 Lasso (cv.glmnet) 634.  529. 0.691         6.31
3 Ridge (cv.glmnet) 638.  554. 0.687        167.

```

```
cat("\n")
```

```

best_model_name <- as.character(perf_d10$model[1])
cat("Best model (lavest RMSE): ", best_model_name, "\n\n", sep = "")

```

```
Best model (lavest RMSE): OLS (lm)
```

```

prep_features_d10 <- function(df) {
  req <- c("kamp_id", "kamp_dato", "sæson", "runde", "kamp_time_h", "billetter_d10")
  miss <- setdiff(req, names(df))
  if (length(miss) > 0) stop(" Forecast input mangler: ", paste(miss, collapse = ", "))

  df |>
    dplyr::transmute(
      kamp_id = as.character(kamp_id),

```

```

    kamp_dato      = as.Date(kamp_dato),
    sæson_chr      = as.character(sæson),
    sæson_start    = season_start_year(sæson_chr),
    runde          = as.integer(runde),
    kamp_time_h    = as.integer(kamp_time_h),
    billetter_d10  = as.numeric(billetter_d10)
  )
}

predict_d10_best <- function(artifacts, new_df) {
  new_x <- prep_features_d10(new_df)
  keep <- stats::complete.cases(new_x[, c("sæson_start", "runde", "kamp_time_h", "billetter_d10")])

  out <- tibble::tibble(
    kamp_id      = new_x$kamp_id,
    kamp_dato    = new_x$kamp_dato,
    pred         = NA_real_
  )

  if (sum(keep) == 0) return(out)

  best <- artifacts$best$name

  if (best == "OLS (lm)") {
    out$pred[keep] <- clip_nonneg(as.numeric(predict(artifacts$models$lm, newdata = new_x[keep, , drop = F
    return(out)
  }

  X <- stats::model.matrix(
    ~ sæson_start + runde + kamp_time_h + billetter_d10,
    data = new_x
  )[, -1, drop = FALSE]

  if (best == "Ridge (cv.glmnet)") {
    out$pred[keep] <- clip_nonneg(as.numeric(predict(artifacts$models$ridge, newx = X[keep, , drop = F
    return(out)
  }

  if (best == "Lasso (cv.glmnet)") {
    out$pred[keep] <- clip_nonneg(as.numeric(predict(artifacts$models$lasso, newx = X[keep, , drop = F
    return(out)
  }

  stop(" Ukendt best model: ", best)
}

d10_artifacts <- list(
  perf = perf_d10,
  models = list(
    lm      = m_lm,
    ridge   = cv_ridge,

```

```

    lasso = cv_lasso
  ),
  best = list(
    name = best_model_name
  ),
  split_info = list(
    test_prop = test_prop,
    cut_date_test_start = min(test_d10$kamp_dato),
    n_train = nrow(train_d10),
    n_test = nrow(test_d10),
    date_min = min(df_d10_final$kamp_dato),
    date_max = max(df_d10_final$kamp_dato)
  ),
  predict = list(
    prep_features = prep_features_d10,
    predict_best = predict_d10_best
  )
)

cat("D10 artifacts klar   Objekt: d10_artifacts\n\n")

```

```
D10 artifacts klar   Objekt: d10_artifacts
```

### 5.1.3.1 Modelperformance og valg af bedste D10-model

OLS-, Ridge- og Lasso-modellerne præsterer meget ens på testsættet. OLS-modellen opnår den laveste RMSE og den højeste  $R^2$ , mens de regulariserede modeller ikke giver en målbar forbedring i prædiktiv performance. Ridge-modellen udviser en svag forringelse, og Lasso reducerer ikke fejlen trods regularisering.

Lasso fastholder alle forklarende variable ved `_min`, hvilket indikerer, at der ikke er behov for variabelselektion i denne modelspecifikation. Ridge dæmper koefficienterne, men uden gevinst i præcision.

På denne baggrund vælges OLS-modellen som den bedste D10-model. Den er både den mest præcise og den mest fortolkelige, hvilket gør den velegnet til operationel anvendelse og videre prognoser.

## 5.2 D10-model Endelig

Denne blok fastlægger den endelige D10-modelpipeline med tidsbaseret datasplit og estimering af OLS-, Ridge- og Lasso-modeller. Samtidig sikres, at det nødvendige datagrundlag og de krævede variable er til stede, så modellen kan estimeres reproducerbart og efterfølgende anvendes til evaluering, visualisering og prognoser.

### 5.2.1 Hjælpfunktioner til modelligninger i plots

Denne sektion definerer hjælpefunktioner til automatisk at generere og formatere ligningstekster for OLS-, Ridge- og Lasso-modeller. Funktionerne sikrer ensartet notation og visuel præsentation af modellernes estimer, som kan indsættes direkte i plots på samme måde som i D7 FINAL.

```

wrap_text <- function(s, width = 85) {
  paste(strwrap(as.character(s), width = width), collapse = "\n")
}

build_lm_equation_text <- function(lm_model, yhat_name = "ŷ") {
  cf <- stats::coef(lm_model)
  cf <- cf[!is.na(cf)]
  nm <- names(cf)

  intercept <- unname(cf[1])
  terms <- nm[-1]
  betas <- unname(cf[-1])

  fmt <- function(x) formatC(x, format = "f", digits = 3)

  rhs <- paste0(fmt(intercept))
  if (length(terms) > 0) {
    for (i in seq_along(terms)) {
      b <- betas[i]
      sign_txt <- if (b >= 0) " + " else " - "
      rhs <- paste0(rhs, sign_txt, fmt(abs(b)), ".", terms[i])
    }
  }

  paste0(yhat_name, " = ", rhs, " (OLS)")
}

build_glmnet_equation_text <- function(cv_model, x_names, model_label = "GLMNET", yhat_name = "ŷ") {
  cm <- as.matrix(stats::coef(cv_model, s = "lambda.min"))
  b <- as.numeric(cm[, 1])
  names(b) <- rownames(cm)

  intercept <- b["(Intercept)"]
  beta <- b[setdiff(names(b), "(Intercept)")]
  beta <- beta[x_names]

  fmt <- function(x) formatC(x, format = "f", digits = 3)

  rhs <- paste0(fmt(intercept))
  if (length(beta) > 0) {
    for (i in seq_along(beta)) {
      bi <- beta[i]
      sign_txt <- if (bi >= 0) " + " else " - "
      rhs <- paste0(rhs, sign_txt, fmt(abs(bi)), ".", names(beta)[i])
    }
  }

  paste0(yhat_name, " = ", rhs, " (", model_label, ", )")
}

add_eq_to_plot <- function(p, eq_text, size = 3) {

```

```

p + ggplot2::annotate(
  "text",
  x = -Inf, y = Inf,
  label = wrap_text(eq_text, width = 85),
  hjust = -0.05, vjust = 1.1,
  size = size
)
}

```

## 5.2.2 Klargøring af D10-modelleringsdatasæt

Datasættet klargøres til modellering med én række pr. kamp, hvor relevante variable udvælges, datatyper standardiseres, og observationer med manglende værdier fjernes. Data sorteres kronologisk, så datasættet er konsistent og klar til tidsbaseret modellering.

```

df_d10_final <- analysis_df_d10 |>
  dplyr::transmute(
    kamp_id      = as.character(kamp_id),
    kamp_dato    = as.Date(kamp_dato),
    tilskuere    = as.numeric(tilskuere),
    sæson_chr    = as.character(sæson),
    sæson_start  = season_start_year(sæson_chr),
    runde        = as.integer(runde),
    kamp_time_h  = as.integer(kamp_time_h),
    billetter_d10 = as.numeric(billetter_d10)
  ) |>
  dplyr::filter(dplyr::if_all(dplyr::everything(), ~ !is.na(.))) |>
  dplyr::arrange(kamp_dato)

cat("df_d10_final klar   Rækker:", nrow(df_d10_final),
    "Periode:", as.character(min(df_d10_final$kamp_dato)), "→", as.character(max(df_d10_final$kamp_dato)), "\n")

```

```
df_d10_final klar   Rækker: 254 Periode: 2000-07-30 → 2025-12-07
```

## 5.2.3 Tidsbaseret trænings- og testsplit

Datasættet opdeles tidsmæssigt, hvor de seneste 20 % af kampene anvendes som testdata. Opdelingen sikrer en realistisk evaluering af modellens prædiktive performance på fremtidige kampe og eliminerer risikoen for informationsslækage fra træningsperioden.

```

set.seed(42)
test_prop <- 0.20

n_total <- nrow(df_d10_final)
n_test  <- max(1, floor(n_total * test_prop))
cut_idx <- n_total - n_test

train_d10 <- df_d10_final[1:cut_idx, ]
test_d10  <- df_d10_final[(cut_idx + 1):n_total, ]

```

```
cat("Tidssplit klar \n")
```

Tidssplit klar

```
cat("Train:", nrow(train_d10), " Test:", nrow(test_d10), "\n")
```

Train: 204 Test: 50

```
cat("Test-periode starter:", as.character(min(test_d10$kamp_dato)), "\n\n")
```

Test-periode starter: 2022-11-06

### 5.2.4 Endelig D10 OLS-model

Den endelige D10-model estimeres som en lineær regressionsmodel på træningsdata og evalueres på testsættet. Modellens prædiktive performance vurderes ved hjælp af RMSE, MAE og  $R^2$ , som danner referencegrundlag for sammenligning med de regulariserede modeller.

```
m_d10_final_lm <- lm(
  tilskuere ~ sæson_start + runde + kamp_time_h + billetter_d10,
  data = train_d10
)

pred_lm <- as.numeric(predict(m_d10_final_lm, newdata = test_d10))

rmse_lm <- rmse_vec(test_d10$tilskuere, pred_lm)
mae_lm  <- mae_vec(test_d10$tilskuere, pred_lm)
r2_lm   <- r2_vec(test_d10$tilskuere, pred_lm)

cat("=====\n")
```

```
cat("D10 FINAL OLS - TEST METRICS\n")
```

D10 FINAL OLS - TEST METRICS

```
cat("=====\n")
```

```
cat("RMSE:", rmse_lm, " MAE:", mae_lm, " R2:", r2_lm, "\n\n")
```

RMSE: 633.8464 MAE: 526.3995 R2: 0.6916656

OLS-modellen leverer den bedste samlede performance på D10-horisonten med lavest RMSE og MAE samt den højeste forklaringsgrad blandt de testede modeller. Resultatet viser, at en enkel og fortolkelig lineær model er tilstrækkelig til at fange de væsentligste sammenhænge i data. På denne baggrund vælges **OLS (lm)** som den endelige D10-model. Men vi tester de andre foruden for at være sikre.

### 5.2.5 Designmatricer til Ridge- og Lasso-modeller

Designmatricer konstrueres for trænings- og testsættet med samme modelspecifikation som OLS-modellen. Dette sikrer konsistens i input til Ridge- og Lasso-estimering og muliggør en direkte og retfærdig sammenligning af modellernes performance.

```
x_train <- model.matrix(
  tilskuere ~ sæson_start + runde + kamp_time_h + billetter_d10,
  data = train_d10
)[, -1, drop = FALSE]
y_train <- train_d10$tilskuere

x_test <- model.matrix(
  tilskuere ~ sæson_start + runde + kamp_time_h + billetter_d10,
  data = test_d10
)[, -1, drop = FALSE]
y_test <- test_d10$tilskuere

x_names <- colnames(x_train)
```

## 5.3 Ridge-regression (D10)

Ridge-modellen estimeres med 10-fold krydsvalidering, hvor den optimale regulariseringsparameter vælges automatisk. Modellen evalueres på testsættet ved hjælp af RMSE, MAE og  $R^2$  for at vurdere, om regularisering forbedrer den prædiktive performance i forhold til OLS-modellen.

```
set.seed(42)
cv_ridge <- glmnet::cv.glmnet(
  x = x_train, y = y_train,
  alpha = 0,
  nfolds = 10,
  standardize = TRUE
)

lambda_ridge <- cv_ridge$lambda.min
pred_ridge <- as.numeric(predict(cv_ridge, newx = x_test, s = "lambda.min"))

rmse_ridge <- rmse_vec(y_test, pred_ridge)
mae_ridge <- mae_vec(y_test, pred_ridge)
r2_ridge <- r2_vec(y_test, pred_ridge)

cat("=====\n")
```

=====

```
cat("D10 RIDGE - TEST METRICS\n")
```

D10 RIDGE - TEST METRICS

```
cat("=====\n")
```

```
cat("lambda.min:", lambda_ridge, "\n")
```

```
lambda.min: 167.4733
```

```
cat("RMSE:", rmse_ridge, " MAE:", mae_ridge, " R2:", r2_ridge, "\n\n")
```

```
RMSE: 638.4862 MAE: 553.718 R2: 0.6871351
```

Ridge-modellen præsterer svagere end både OLS- og Lasso-modellerne på D10-horisonten med højere RMSE og MAE samt lavere  $R^2$ . Selvom regularisering reducerer koefficienternes følsomhed, medfører den ingen forbedring i prædiktiv performance. Ridge fravælges derfor som endelig model.

## 5.4 Lasso-regression (D10)

Lasso-modellen estimeres med 10-fold krydsvalidering, hvor den optimale regulariseringsparameter identificeres automatisk. Modellen evalueres på testsættet ved hjælp af RMSE, MAE og  $R^2$  for at vurdere, om variabelselektion og regularisering giver en forbedring i prædiktiv performance sammenlignet med OLS- og Ridge-modellerne.

```
# 6) Lasso (alpha = 1)
```

```
set.seed(42)
cv_lasso <- glmnet::cv.glmnet(
  x = x_train, y = y_train,
  alpha = 1,
  nfolds = 10,
  standardize = TRUE
)

lambda_lasso <- cv_lasso$lambda.min
pred_lasso <- as.numeric(predict(cv_lasso, newx = x_test, s = "lambda.min"))

rmse_lasso <- rmse_vec(y_test, pred_lasso)
mae_lasso <- mae_vec(y_test, pred_lasso)
r2_lasso <- r2_vec(y_test, pred_lasso)

cat("=====\n")
```

```
cat("D10 LASSO - TEST METRICS\n")
```

```
D10 LASSO - TEST METRICS
```

```
cat("=====\n")
```

```
cat("lambda.min:", lambda_lasso, "\n")
```

```
lambda.min: 6.305263
```

```
cat("RMSE:", rmse_lasso, " MAE:", mae_lasso, " R2:", r2_lasso, "\n\n")
```

```
RMSE: 634.2758 MAE: 529.0097 R2: 0.6912478
```

Lasso-modellen opnår en performance, der ligger meget tæt på OLS-modellen, men uden at give en forbedring i hverken RMSE, MAE eller  $R^2$ . Regularisering og potentiel variabelselektion tilfører således ingen ekstra prædiktiv gevinst på D10-horisonten. På denne baggrund fravælges Lasso som endelig model til fordel for OLS.

## 5.5 Samlet performance for D10-modeller

Performance for OLS-, Ridge- og Lasso-modellerne sammenfattes i en fælles tabel baseret på testsættet. Sammenligningen gør det muligt at identificere den model, der leverer den bedste prædiktive kvalitet på D10-horisonten på tværs af fejlkriterierne RMSE, MAE og  $R^2$ .

```
# =====
# 7) Samlet performance-tabel
# =====
perf_d10 <- tibble::tibble(
  model = c("OLS (lm)", "Ridge (cv.glmnet)", "Lasso (cv.glmnet)"),
  RMSE = c(rmse_lm, rmse_ridge, rmse_lasso),
  MAE = c(mae_lm, mae_ridge, mae_lasso),
  R2 = c(r2_lm, r2_ridge, r2_lasso)
)

cat("=====\n")
```

```
cat("D10 - SAMLET TEST PERFORMANCE\n")
```

```
D10 - SAMLET TEST PERFORMANCE
```

```
cat("=====\n")
```

```
print(perf_d10)
```

```
# A tibble: 3 x 4
  model      RMSE  MAE   R2
  <chr>    <dbl> <dbl> <dbl>
1 OLS (lm)    634.  526. 0.692
2 Ridge (cv.glmnet) 638.  554. 0.687
3 Lasso (cv.glmnet) 634.  529. 0.691
```

```
cat("\n")
```

OLS- og Lasso-modellerne præsterer stort set identisk og leverer den bedste prædiktive kvalitet på D10-horisonten. Ridge-modellen klarer sig marginalt dårligere på alle tre mål. Da OLS opnår lavest RMSE og højest  $R^2$  samtidig med størst fortolkelighed, vælges **OLS (lm)** som den endelige D10-model.

## 5.6 Visualisering og gemning af D10-resultater

Der genereres standardplots for D10-modellerne, herunder *Actual vs. Predicted* og *residualplots* for OLS, Ridge og Lasso. Modelligningerne vises direkte i figurerne for at understøtte fortolkning og transparens. Alle centrale resultater, herunder performance, modeller, plots og ligningstekster, samles og gemmes i ét fælles objekt til videre brug og prognoser.

```
plot_df <- tibble::tibble(
  kamp_dato = test_d10$kamp_dato,
  actual     = y_test,
  pred_lm    = pred_lm,
  pred_ridge = pred_ridge,
  pred_lasso = pred_lasso
) |>
  dplyr::mutate(
    resid_lm    = actual - pred_lm,
    resid_ridge = actual - pred_ridge,
    resid_lasso = actual - pred_lasso
  )

eq_ols  <- build_lm_equation_text(m_d10_final_lm, yhat_name = "ŷ")
eq_ridge <- build_glmnet_equation_text(cv_ridge, x_names = x_names, model_label = "Ridge", yhat_name = "ŷ")
eq_lasso <- build_glmnet_equation_text(cv_lasso, x_names = x_names, model_label = "Lasso", yhat_name = "ŷ")

p1 <- ggplot2::ggplot(plot_df, ggplot2::aes(x = actual, y = pred_lm)) +
  ggplot2::geom_point() +
  ggplot2::geom_abline(slope = 1, intercept = 0) +
  ggplot2::labs(
    title = "D10 TEST: Actual vs Pred (OLS)",
    x = "Faktiske tilskuere",
    y = "Forudsagte tilskuere"
  )
p1 <- add_eq_to_plot(p1, eq_ols)

p2 <- ggplot2::ggplot(plot_df, ggplot2::aes(x = actual, y = pred_ridge)) +
```

```

ggplot2::geom_point() +
ggplot2::geom_abline(slope = 1, intercept = 0) +
ggplot2::labs(
  title = "D10 TEST: Actual vs Pred (Ridge)",
  x = "Faktiske tilskuere",
  y = "Forudsagte tilskuere"
)
p2 <- add_eq_to_plot(p2, eq_ridge)

p3 <- ggplot2::ggplot(plot_df, ggplot2::aes(x = actual, y = pred_lasso)) +
  ggplot2::geom_point() +
  ggplot2::geom_abline(slope = 1, intercept = 0) +
  ggplot2::labs(
    title = "D10 TEST: Actual vs Pred (Lasso)",
    x = "Faktiske tilskuere",
    y = "Forudsagte tilskuere"
  )
p3 <- add_eq_to_plot(p3, eq_lasso)

p4 <- ggplot2::ggplot(plot_df, ggplot2::aes(x = pred_lm, y = resid_lm)) +
  ggplot2::geom_point() +
  ggplot2::geom_hline(yintercept = 0) +
  ggplot2::labs(
    title = "D10 TEST: Residualer vs Pred (OLS)",
    x = "Forudsagt",
    y = "Residual"
  )
p4 <- add_eq_to_plot(p4, eq_ols)

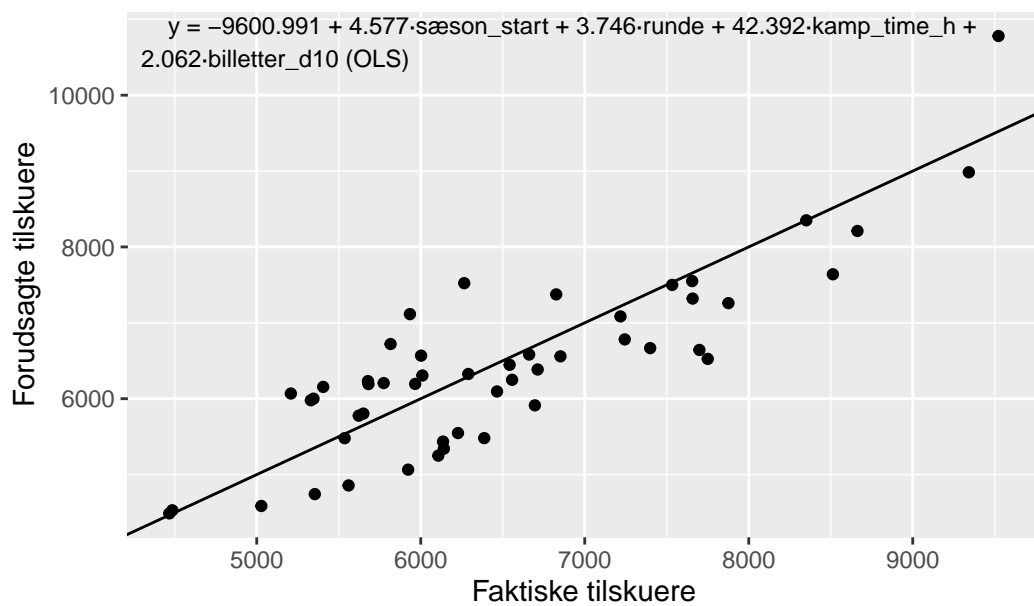
p5 <- ggplot2::ggplot(plot_df, ggplot2::aes(x = pred_ridge, y = resid_ridge)) +
  ggplot2::geom_point() +
  ggplot2::geom_hline(yintercept = 0) +
  ggplot2::labs(
    title = "D10 TEST: Residualer vs Pred (Ridge)",
    x = "Forudsagt",
    y = "Residual"
  )
p5 <- add_eq_to_plot(p5, eq_ridge)

p6 <- ggplot2::ggplot(plot_df, ggplot2::aes(x = pred_lasso, y = resid_lasso)) +
  ggplot2::geom_point() +
  ggplot2::geom_hline(yintercept = 0) +
  ggplot2::labs(
    title = "D10 TEST: Residualer vs Pred (Lasso)",
    x = "Forudsagt",
    y = "Residual"
  )
p6 <- add_eq_to_plot(p6, eq_lasso)

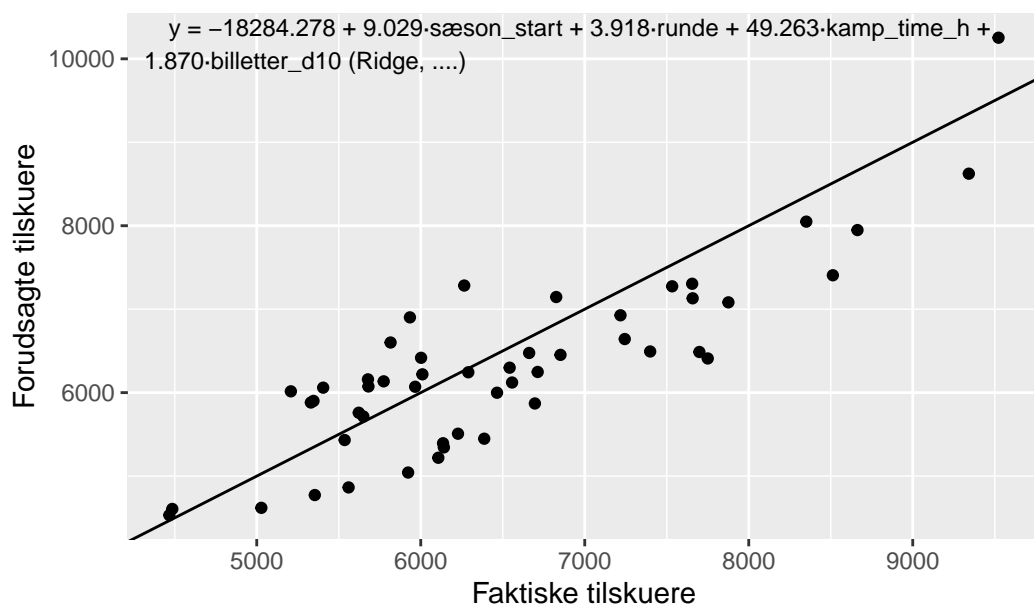
print(p1); print(p2); print(p3); print(p4); print(p5); print(p6)

```

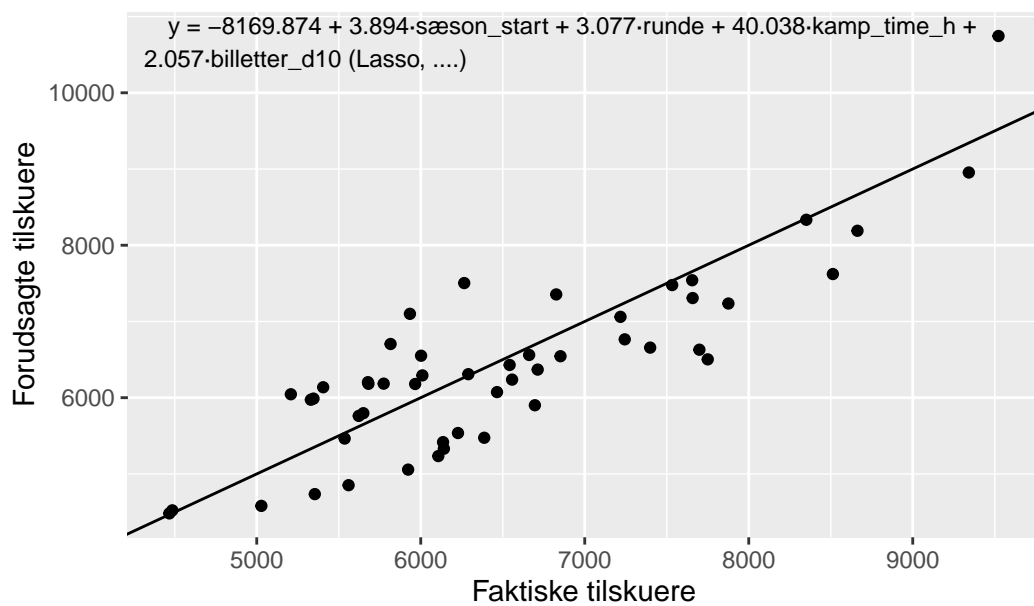
### D10 TEST: Actual vs Pred (OLS)



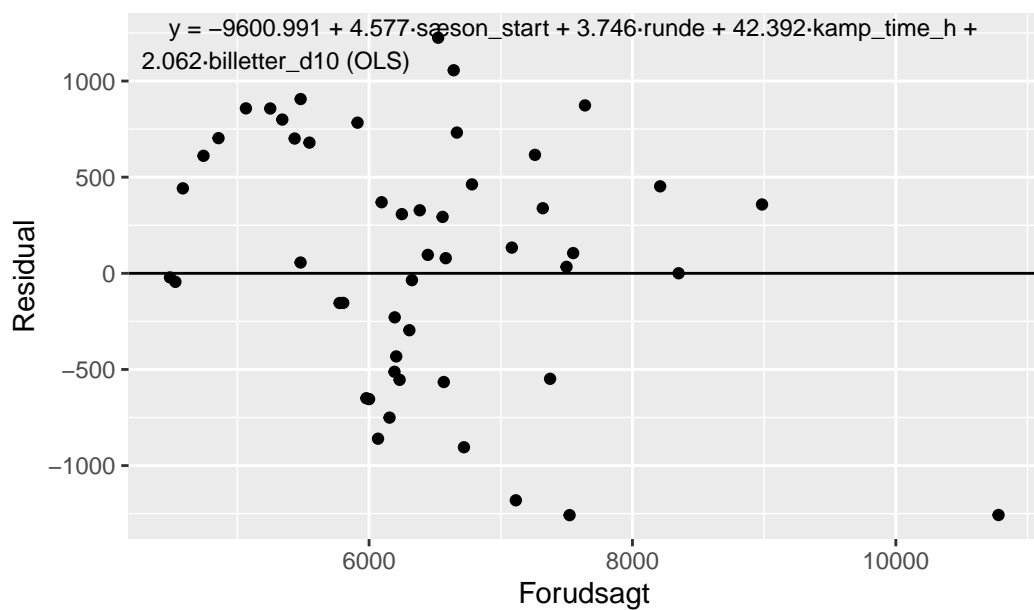
### D10 TEST: Actual vs Pred (Ridge)



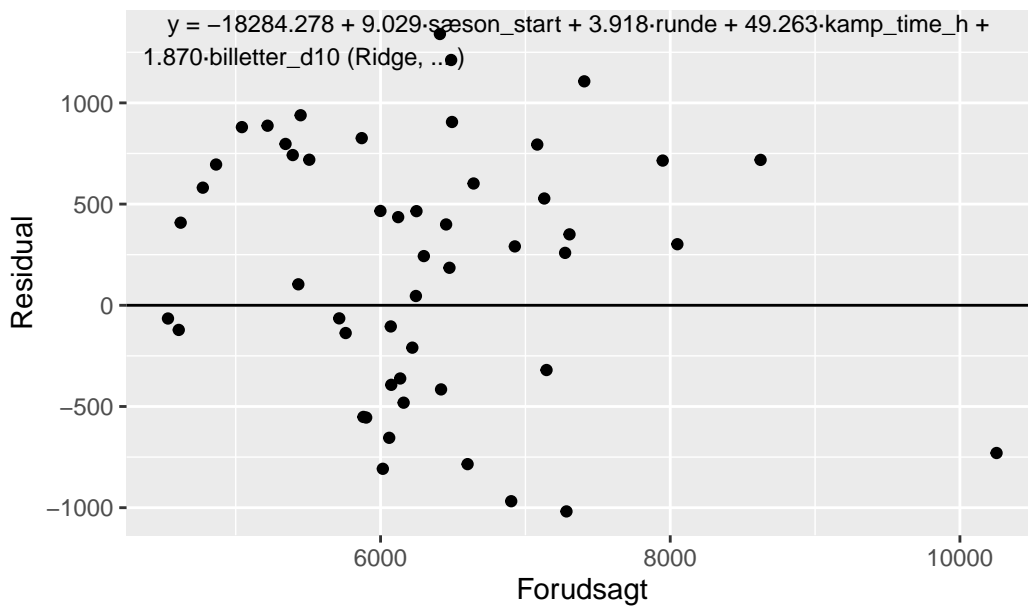
### D10 TEST: Actual vs Pred (Lasso)



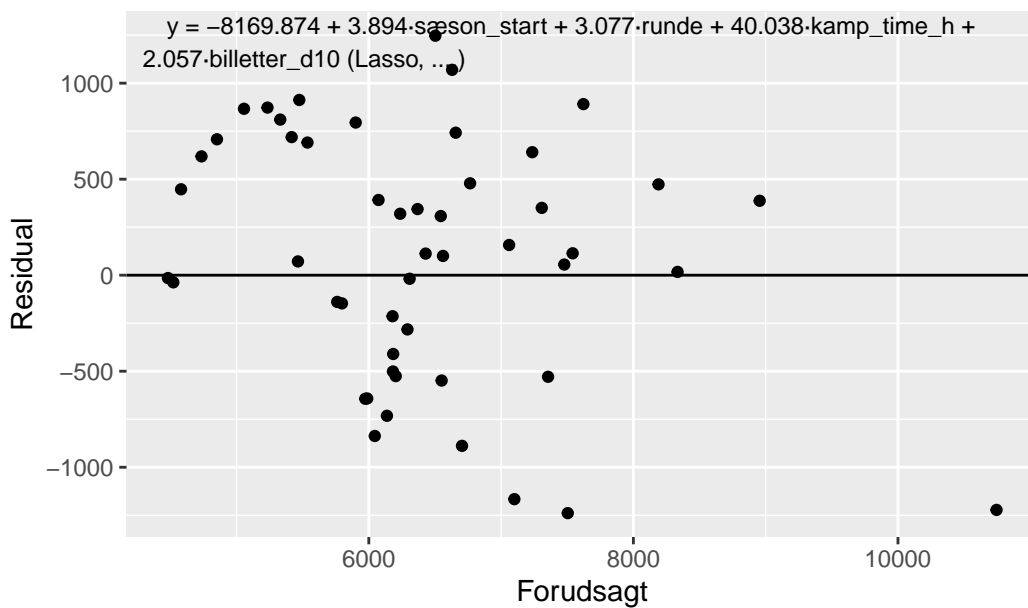
### D10 TEST: Residualer vs Pred (OLS)



### D10 TEST: Residualer vs Pred (Ridge)



### D10 TEST: Residualer vs Pred (Lasso)

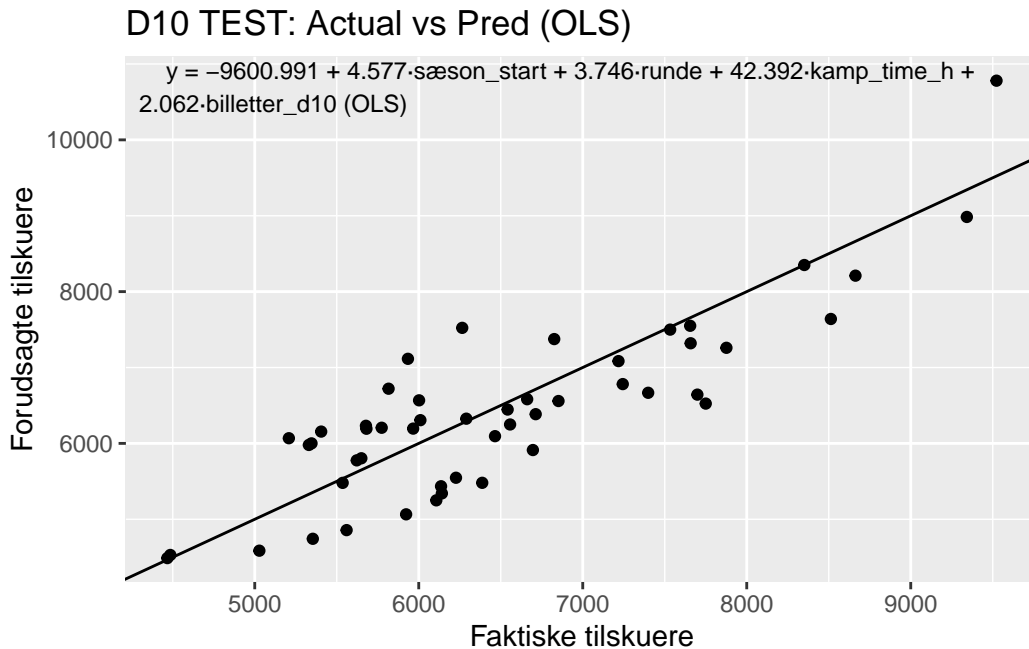


```
d10_results <- list(
  perf      = perf_d10,
  lm_model  = m_d10_final_lm,
  ridge_lambda = lambda_ridege,
  lasso_lambda = lambda_lasso,
  plot_data  = plot_df,
  equations  = list(ols = eq_ols, ridge = eq_ridege, lasso = eq_lasso)
)

cat("D10 final resultater gemt i objektet: d10_results \n")
```

D10 final resultater gemt i objektet: d10\_results

```
p1 <- ggplot2::ggplot(plot_df, ggplot2::aes(x = actual, y = pred_lm)) +
  ggplot2::geom_point() +
  ggplot2::geom_abline(slope = 1, intercept = 0) +
  ggplot2::labs(
    title = "D10 TEST: Actual vs Pred (OLS)",
    x = "Faktiske tilskuere",
    y = "Forudsagte tilskuere"
  )
p1 <- add_eq_to_plot(p1, eq_ols)
print(p1)
```

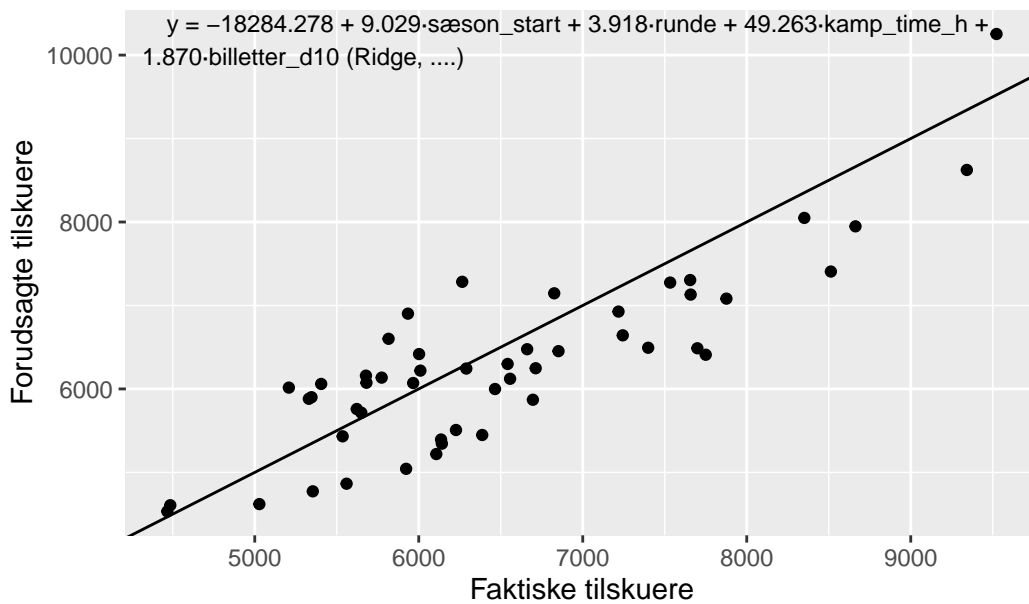


Denne graf viser sammenhængen mellem faktiske og forudsagte tilskuertal for D10 OLS-modellen på testsættet. Hver prik repræsenterer én kamp, hvor den horisontale akse angiver det faktiske tilskuertal, og den vertikale akse viser modellens forudsigelse. Den diagonale linje repræsenterer perfekt prædiktion.

Punkterne ligger overordnet tæt omkring diagonalen, hvilket indikerer en god modeltilpasning og en stærk lineær sammenhæng mellem observerede og forudsagte værdier. Afvigelserne øges en smule ved de højeste tilskuertal, men der ses ingen systematisk bias. Samlet set bekræfter figuren, at OLS-modellen leverer robuste og operationelt anvendelige forudsigelser på D10-horisonten.

```
p2 <- ggplot2::ggplot(plot_df, ggplot2::aes(x = actual, y = pred_ridge)) +
  ggplot2::geom_point() +
  ggplot2::geom_abline(slope = 1, intercept = 0) +
  ggplot2::labs(
    title = "D10 TEST: Actual vs Pred (Ridge)",
    x = "Faktiske tilskuere",
    y = "Forudsagte tilskuere"
  )
p2 <- add_eq_to_plot(p2, eq_ridge)
print(p2)
```

## D10 TEST: Actual vs Pred (Ridge)

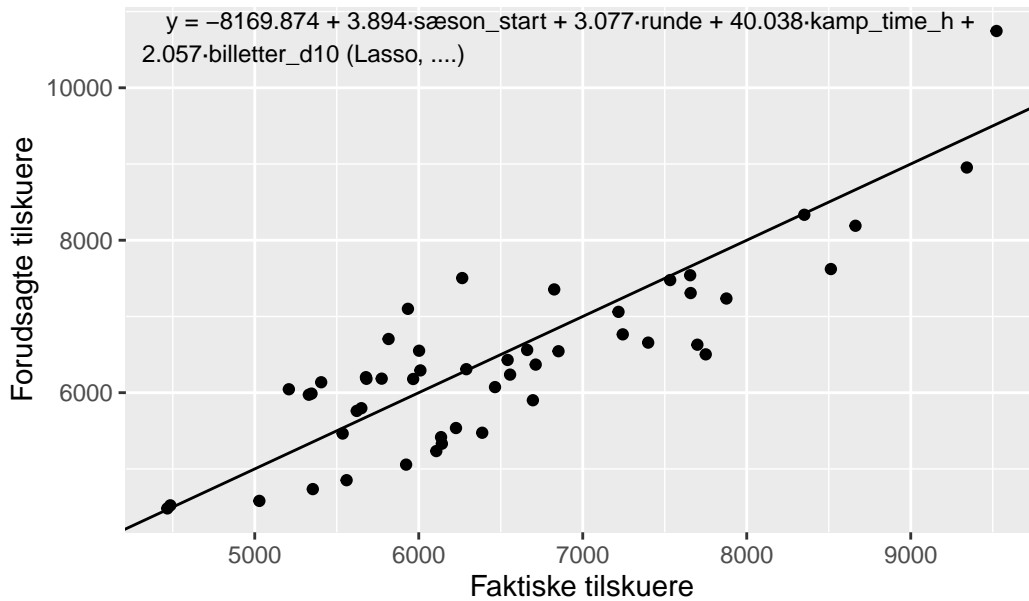


D10 Ridge-modellen sammenligner her faktiske og forudsagte tilskuertal på testsættet. Hver observation repræsenterer én kamp, hvor den vandrette akse viser det faktiske tilskuertal, og den lodrette akse viser modellens forudsigelse. Den diagonale linje angiver perfekt prædiktio.

Punkterne følger generelt den diagonale reference, hvilket viser, at modellen opfanger den overordnede lineære sammenhæng. Der ses dog større spredning end i OLS-modellen, særligt ved højere tilskuertal, hvor forudsigelserne i flere tilfælde afviger mere fra de observerede værdier. Samlet set bekræfter dette, at Ridge ikke giver en forbedring i prædiktiv kvalitet i forhold til OLS på D10-horisonten.

```
p3 <- ggplot2::ggplot(plot_df, ggplot2::aes(x = actual, y = pred_lasso)) +
  ggplot2::geom_point() +
  ggplot2::geom_abline(slope = 1, intercept = 0) +
  ggplot2::labs(
    title = "D10 TEST: Actual vs Pred (Lasso)",
    x = "Faktiske tilskuere",
    y = "Forudsagte tilskuere"
  )
p3 <- add_eq_to_plot(p3, eq_lasso)
print(p3)
```

## D10 TEST: Actual vs Pred (Lasso)

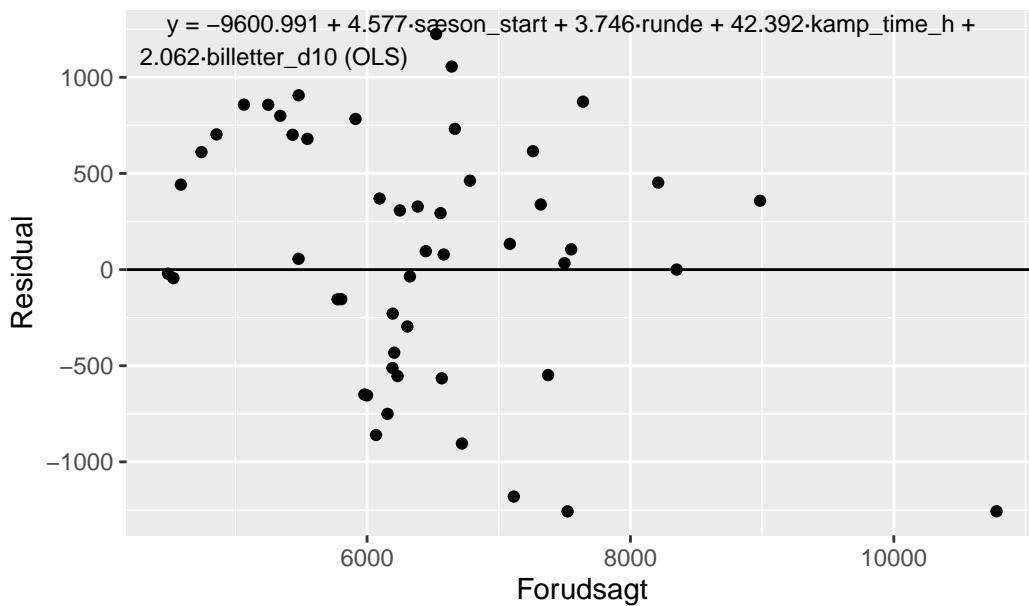


D10 Lasso-modellen viser her sammenhængen mellem faktiske og forudsagte tilskuertal på testsættet. Den vandrette akse angiver det faktiske tilskuertal, mens den lodrette akse viser modellens forudsigelse. Den diagonale linje repræsenterer perfekt overensstemmelse.

Observationerne ligger generelt tæt omkring diagonalen og viser en prædiktiv kvalitet, der er meget lig OLS-modellen. Der ses ingen tydelig systematisk bias, men heller ingen klar forbedring i forhold til OLS. Resultatet understøtter, at Lasso ikke tilfører ekstra prædiktiv styrke på D10-horisonten, men leverer sammenlignelige – dog ikke bedre – forudsigelser.

```
p4 <- ggplot2::ggplot(plot_df, ggplot2::aes(x = pred_lm, y = resid_lm)) +  
  ggplot2::geom_point() +  
  ggplot2::geom_hline(yintercept = 0) +  
  ggplot2::labs(  
    title = "D10 TEST: Residualer vs Pred (OLS)",  
    x = "Forudsagt",  
    y = "Residual"  
  )  
p4 <- add_eq_to_plot(p4, eq_ols)  
print(p4)
```

## D10 TEST: Residualer vs Pred (OLS)

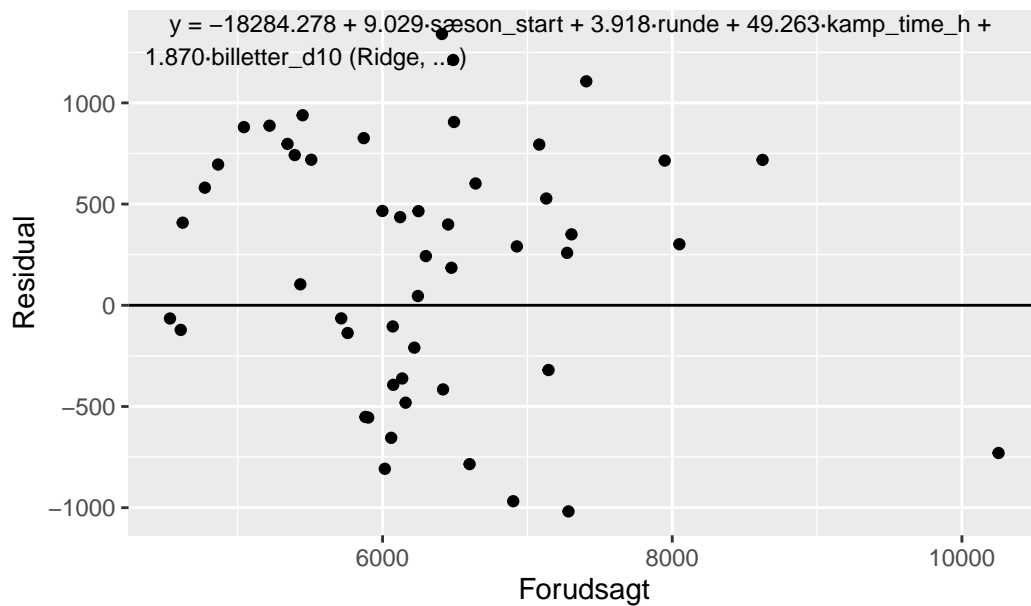


D10 OLS-modellen vises her med residualerne plottet mod de forudsagte tilskuertal på testsættet. Den vandrette nul-linje angiver perfekt prædiktions, hvor positive residualer svarer til undervurdering, og negative residualer til overvurdering.

Residualerne er overordnet tilfældigt fordelt omkring nul uden tydelige systematiske mønstre, hvilket indikerer, at modellens lineære antagelse er rimelig på D10-horisonten. Der ses dog enkelte større afvigelser ved høje forudsagte tilskuertal, som bidrager til den samlede fejl, men uden at pege på strukturel bias. Samlet set understøtter plottet, at OLS-modellen er stabil og velegnet som endelig D10-model.

```
p5 <- ggplot2::ggplot(plot_df, ggplot2::aes(x = pred_ridge, y = resid_ridge)) +  
  ggplot2::geom_point() +  
  ggplot2::geom_hline(yintercept = 0) +  
  ggplot2::labs(  
    title = "D10 TEST: Residualer vs Pred (Ridge)",  
    x = "Forudsagt",  
    y = "Residual"  
  )  
p5 <- add_eq_to_plot(p5, eq_ridge)  
print(p5)
```

## D10 TEST: Residualer vs Pred (Ridge)

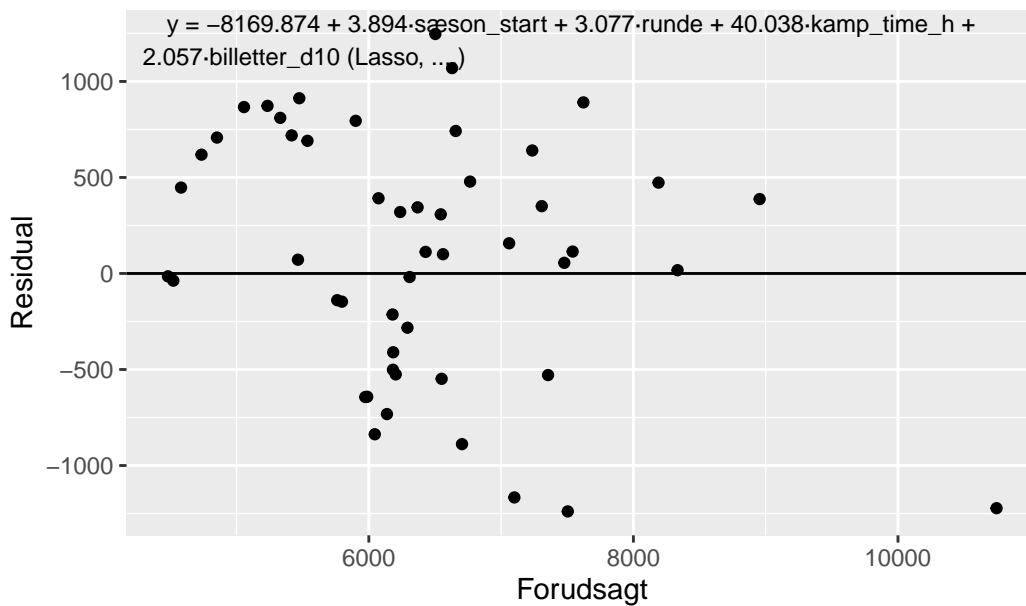


D10 Ridge-modellen vises her med residualer plottet mod de forudsagte tilskuertal på testsættet. Den vandrette nul-linje markerer perfekt prædiktion, hvor positive residualer indikerer undervurdering og negative residualer indikerer overvurdering.

Residualerne er fortsat spredt omkring nul, men udviser større variation end i OLS-modellen, særligt ved højere forudsagte tilskuertal. Der ses flere markante negative residualer, hvilket tyder på en tendens til overvurdering i den øvre ende. Plottet understøtter dermed, at Ridge-regularisering ikke forbedrer modellens stabilitet eller prædiktive kvalitet i D10-sammenhæng.

```
p6 <- ggplot2::ggplot(plot_df, ggplot2::aes(x = pred_lasso, y = resid_lasso)) +
  ggplot2::geom_point() +
  ggplot2::geom_hline(yintercept = 0) +
  ggplot2::labs(
    title = "D10 TEST: Residualer vs Pred (Lasso)",
    x = "Forudsagt",
    y = "Residual"
  )
p6 <- add_eq_to_plot(p6, eq_lasso)
print(p6)
```

## D10 TEST: Residualer vs Pred (Lasso)



D10 Lasso-modellen vises her med residualer plottet mod de forudsagte tilskuertal på testsættet. Den vandrette nul-linje angiver perfekt prædiktion, hvor positive residualer svarer til undervurdering og negative residualer til overvurdering.

Residualerne er overordnet centreret omkring nul og viser et mønster, der minder meget om OLS-modellen. Der ses ingen tydelig systematisk struktur, men enkelte større negative residualer ved høje forudsagte tilskuertal indikerer, at modellen i visse tilfælde overvurderer publikumspotentialet. Samlet set bekræfter plottet, at Lasso ikke forbedrer modellens stabilitet eller fejlskæthed i forhold til OLS på D10-horisonten.

## 5.7 Produktlag og interaktiv prognose (Shiny)

Produktlaget samler de to bedst præsterende D10-modeller (OLS og Lasso) i én operationel løsning. Den bedste model vælges automatisk som standard, men kan skiftes manuelt i en Shiny-app, hvor brugeren kan indtaste kamp- og billet-situation og få en forudsagt tilskuermængde, fyldningsgrad af stadion samt forventet billetsalg fra D10 til kampdag. Løsningen er designet til praktisk beslutningsstøtte og visualiserer resultatet klart i forhold til stadionkapacitet.

```
# shiny

# =====
# D10 PRODUKTLAG - TOP 2 MODELLER (OLS + LASSO) + ÉN PLOT + SHINY APP
# =====

suppressPackageStartupMessages({
  library(dplyr)
  library(tibble)
  library(ggplot2)
  library(shiny)
})

.must_exist <- function(x) exists(x, inherits = TRUE)
```

```

stopifnot(.must_exist("perf_d10"), .must_exist("plot_df"), .must_exist("train_d10"), .must_exist("test_d10"))
stopifnot(.must_exist("m_d10_final_lm"), .must_exist("cv_lasso"))
stopifnot(.must_exist("rmse_vec"), .must_exist("clip_nonneg"))
stopifnot(.must_exist("wrap_text"), .must_exist("eq_ols"), .must_exist("eq_lasso"))

STADIUM_CAP_VFF <- 10000
clip_to_cap <- function(x, cap = STADIUM_CAP_VFF) pmin(pmax(as.numeric(x), 0), cap)

perf_top2 <- perf_d10 |>
  dplyr::filter(grepl("^OLS", model, ignore.case = TRUE) | grepl("^Lasso", model, ignore.case = TRUE))

if (nrow(perf_top2) == 0) {
  perf_top2 <- tibble::tibble(
    model = c("OLS (lm)", "Lasso (cv.glmnet)"),
    RMSE = c(Inf, Inf),
    MAE = c(NA_real_, NA_real_),
    R2 = c(NA_real_, NA_real_)
  )
}

top2_best_row <- perf_top2 |> arrange(RMSE) |> slice(1)

default_key <- dplyr::case_when(
  grepl("^OLS", top2_best_row$model[1], ignore.case = TRUE) ~ "lm",
  grepl("^Lasso", top2_best_row$model[1], ignore.case = TRUE) ~ "lasso",
  TRUE ~ "lm"
)

cat(" Default model (bedste): ", as.character(top2_best_row$model[1]),
    " => ", default_key, "\n\n", sep = "")

.format_num <- function(x, digits = 1) {
  format(round(as.numeric(x), digits), big.mark = ".", decimal.mark = ",", trim = TRUE)
}

plot_pva_with_eq <- function(df, title, eq_text, rmse_val) {
  ggplot(df, aes(x = y, y = yhat)) +
    geom_point(alpha = 0.7) +
    geom_abline(slope = 1, intercept = 0) +
    labs(
      title = title,
      x = "Faktiske tilskuere",
      y = "Forudsagte tilskuere"
    ) +
    annotate(
      "text",
      x = Inf, y = -Inf,
      hjust = 1.02, vjust = -0.2,
      label = paste0("RMSE = ", .format_num(rmse_val, 1)),
      size = 3.8
    ) +

```

```

    annotate(
      "text",
      x = -Inf, y = Inf,
      hjust = -0.02, vjust = 1.15,
      label = wrap_text(eq_text, width = 85),
      size = 3.2
    ) +
    theme_minimal()
  }

y <- as.numeric(plot_df$actual)

if (default_key == "lm") {
  yhat <- as.numeric(plot_df$pred_lm)
  eq_best <- eq_ols
  title_best <- "D10 - BEDSTE MODEL: OLS (lm) - Pred vs Actual"
} else {
  yhat <- as.numeric(plot_df$pred_lasso)
  eq_best <- eq_lasso
  title_best <- "D10 - BEDSTE MODEL: Lasso (min) - Pred vs Actual"
}

df_best <- tibble::tibble(y = y, yhat = yhat)
rmse_best <- rmse_vec(df_best$y, df_best$yhat)

print(plot_pva_with_eq(df_best, title_best, eq_best, rmse_best))
cat("\n D10 bedste-model plot færdig\n\n")

season_label <- function(season_start_int) {
  paste0(as.integer(season_start_int), "/", as.integer(season_start_int) + 1L)
}

parse_season_start <- function(season_text) {
  s <- trimws(as.character(season_text))
  ok <- grepl("^\\d{4}/\\d{4}$", s)

  if (!ok) {
    y <- suppressWarnings(as.integer(substr(s, 1, 4)))
    if (!is.finite(y)) return(list(start = NA_integer_, note = "Ugyldigt sæsonformat. Brug fx 2025/2026"))
    return(list(start = y, note = "Sæsonformat er ikke 'YYYY/YYYY'. Jeg bruger første år som sæson_start"))
  }

  y1 <- suppressWarnings(as.integer(substr(s, 1, 4)))
  y2 <- suppressWarnings(as.integer(substr(s, 6, 9)))
  if (!is.finite(y1) || !is.finite(y2)) return(list(start = NA_integer_, note = "Ugyldigt sæsonformat."))
  if (y2 != y1 + 1L) return(list(start = y1, note = "Andet år er ikke første+1. Jeg bruger første år som sæson_start"))
  list(start = y1, note = "")
}

.range_or <- function(x, fallback = c(0, 1)) {
  r <- range(as.numeric(x), na.rm = TRUE)

```

```

    if (any(!is.finite(r))) fallback else r
  }
.med_or <- function(x, fallback = 0) {
  v <- stats::median(as.numeric(x), na.rm = TRUE)
  if (!is.finite(v)) fallback else v
}

.int_or <- function(x, fallback = 0L) {
  v <- suppressWarnings(as.numeric(x))
  v <- v[is.finite(v)]
  if (length(v) == 0) return(as.integer(fallback))
  as.integer(round(stats::median(v, na.rm = TRUE)))
}

.int_range_or <- function(x, fallback = c(0L, 12000L)) {
  v <- suppressWarnings(as.numeric(x))
  v <- v[is.finite(v)]
  if (length(v) == 0) return(as.integer(fallback))
  r <- range(v, na.rm = TRUE)
  as.integer(c(floor(r[1]), ceiling(r[2])))
}

.fill_bucket <- function(pct) {
  if (!is.finite(pct)) return("UKENDT")
  if (pct >= 85) return("HØJ")
  if (pct >= 60) return("MIDDEL")
  "LAV"
}

.predict_model <- function(model_key, sæson_start, runde, kamp_time_h, billetter_d10) {
  billetter_int <- as.integer(round(as.numeric(billetter_d10)))

  nd <- data.frame(
    sæson_start = as.integer(sæson_start),
    runde = as.integer(runde),
    kamp_time_h = as.integer(kamp_time_h),
    billetter_d10 = billetter_int
  )

  if (model_key == "lm") {
    yhat <- suppressWarnings(as.numeric(predict(m_d10_final_lm, newdata = nd)))
  } else {
    X <- model.matrix(~ sæson_start + runde + kamp_time_h + billetter_d10, data = nd)[, -1, drop = FALSE]
    yhat <- suppressWarnings(as.numeric(predict(cv_lasso, newx = X, s = "lambda.min")))
  }

  if (length(yhat) == 0) yhat <- NA_real_
  clip_nonneg(yhat)
}

.eq_for_key <- function(model_key) {
  if (model_key == "lm") eq_ols else eq_lasso
}

```

```

}
.model_name <- function(model_key) {
  if (model_key == "lm") "OLS (lm)" else "Lasso (cv.glmnet)"
}

ui <- fluidPage(
  titlePanel("D10 - Prediction (vælg model)"),

  sidebarLayout(
    sidebarPanel(
      tags$h4("Model"),
      selectInput(
        "model_choice",
        "Vælg model (default = bedste)",
        choices = c("OLS (lm)" = "lm", "Lasso (cv.glmnet)" = "lasso"),
        selected = default_key
      ),
      tags$hr(),

      tags$h4("Inputs"),
      textInput("season_txt", "Sæson (YYYY/YYYY)", value = "2025/2026"),

      numericInput(
        "runde", "runde",
        value = .med_or(train_d10$runde, fallback = 1),
        min = .range_or(train_d10$runde, fallback = c(1, 40))[1],
        max = .range_or(train_d10$runde, fallback = c(1, 40))[2],
        step = 1
      ),
      numericInput(
        "kamp_time_h", "kamp_time_h",
        value = .med_or(train_d10$kamp_time_h, fallback = 18),
        min = .range_or(train_d10$kamp_time_h, fallback = c(10, 22))[1],
        max = .range_or(train_d10$kamp_time_h, fallback = c(10, 22))[2],
        step = 1
      ),
      numericInput(
        "billetter_d10", "billetter_d10 (heltal)",
        value = .int_or(train_d10$billetter_d10, fallback = 0L),
        min = .int_range_or(train_d10$billetter_d10, fallback = c(0L, 12000L))[1],
        max = .int_range_or(train_d10$billetter_d10, fallback = c(0L, 12000L))[2],
        step = 1
      ),

      tags$hr(),
      numericInput("capacity", "Stadion-kapacitet", value = 10000, min = 1, step = 1),
      actionButton("go", "Predict", class = "btn-primary")
    ),

    mainPanel(
      tags$h4("Output"),

```

```

tags$div(style="font-size:20px; margin-bottom:8px;", textOutput("pred_txt")),
tags$div(style="font-size:20px; margin-bottom:8px;", textOutput("pct_txt")),
tags$div(style="font-size:20px; margin-bottom:8px;", textOutput("delta_txt")),
tags$div(style="font-size:14px; margin-bottom:6px; color:#333;", textOutput("season_note_txt")),
tags$div(style="font-size:14px; margin-bottom:14px; color:#333;", textOutput("eq_txt")),
plotOutput("cap_plot", height = "520px")
)
)
)

server <- function(input, output, session) {

  pred_obj <- eventReactive(input$go, {

    cap <- as.numeric(input$capacity)
    if (!is.finite(cap) || cap <= 0) cap <- 1

    key <- input$model_choice

    season_parsed <- parse_season_start(input$season_txt)
    sæson_start <- season_parsed$start

    if (!is.finite(sæson_start)) {
      return(list(
        ok = FALSE,
        msg = "Ugyldig sæson. Brug formatet 2025/2026.",
        note = season_parsed$note
      ))
    }

    bil_int <- as.integer(round(as.numeric(input$billetter_d10)))

    yhat <- .predict_model(
      model_key      = key,
      sæson_start    = sæson_start,
      runde          = input$runde,
      kamp_time_h    = input$kamp_time_h,
      billetter_d10  = bil_int
    )

    yhat_cap <- clip_to_cap(yhat, cap = cap)

    pct <- (as.numeric(yhat_cap) / cap) * 100
    pct_clip <- max(min(pct, 100), 0)

    delta <- if (is.finite(bil_int)) (as.numeric(yhat_cap) - bil_int) else NA_real_

    list(
      ok = TRUE,
      pred = as.numeric(yhat_cap),
      pct = pct_clip,

```

```

    cap = cap,
    bil_d10 = bil_int,
    delta = delta,
    eq = .eq_for_key(key),
    model_name = .model_name(key),
    season_txt = season_label(sæson_start),
    note = season_parsed$note
  )
})

output$pred_txt <- renderText({
  x <- pred_obj()
  if (is.null(x)) return("Ingen prediction endnu.")
  if (!isTRUE(x$ok)) return(x$msg)

  paste0(
    "Model: ", x$model_name, " - sæson ", x$season_txt, "\n",
    "Forudsagt tilskuertal: ", format(round(x$pred), big.mark = ".", decimal.mark = ",")
  )
})

output$pct_txt <- renderText({
  x <- pred_obj()
  if (is.null(x) || !isTRUE(x$ok)) return("")
  paste0("Stadion fyldt: ", format(round(x$pct, 1), big.mark = ".", decimal.mark = ","), " %")
})

output$delta_txt <- renderText({
  x <- pred_obj()
  if (is.null(x) || !isTRUE(x$ok)) return("")
  if (!is.finite(x$bil_d10)) return("Ekstra billetter (D10 → kampdag): kan ikke beregnes (billetter_")
  paste0(
    "Ekstra billetter (D10 → kampdag): ",
    format(round(x$delta), big.mark = ".", decimal.mark = ","),
    " (ŷ - billetter_d10)"
  )
})

# FIX HER
output$season_note_txt <- renderText({
  x <- pred_obj()
  if (is.null(x) || !isTRUE(x$ok)) return("")
  if (is.null(x$note) || !nzchar(x$note)) return("")
  paste0("Note: ", x$note)
})

output$eq_txt <- renderText({
  x <- pred_obj()
  if (is.null(x) || !isTRUE(x$ok)) return("")
  paste0("Ligning (kort): ", x$eq)
})

```

```

output$cap_plot <- renderPlot({
  x <- pred_obj()
  if (is.null(x) || !isTRUE(x$ok)) return(NULL)

  bucket <- .fill_bucket(x$pct)

  df <- data.frame(
    label = "Stadionkapacitet",
    y = x$pred,
    bucket = bucket
  )

  ggplot(df, aes(x = label, y = y, fill = bucket)) +
    geom_col(width = 0.6) +
    geom_hline(yintercept = x$cap, linewidth = 0.9, linetype = 2) +
    coord_cartesian(ylim = c(0, x$cap)) +
    scale_fill_manual(
      values = c("LAV" = "#d55e00", "MIDDEL" = "#e69f00", "HØJ" = "#009e73", "UKENDT" = "grey60"),
      breaks = c("LAV", "MIDDEL", "HØJ", "UKENDT"),
      name = "Fyldningsniveau"
    ) +
    labs(
      x = NULL,
      y = "Tilskuere (forudsagt)",
      title = "Forudsagt tilskuertal vs stadionkapacitet",
      subtitle = paste0("Kapacitet (stiplet linje): ", format(round(x$cap), big.mark=".", decimal.ma
    ) +
    theme_minimal(base_size = 16) +
    theme(
      legend.position = "top",
      plot.title = element_text(face = "bold"),
      panel.grid.minor = element_blank()
    )
  })
}

shinyApp(ui, server)

```

## 5.8 Scenarie: Prognose som beslutningsværktøj før kampdag

I forbindelse med planlægningen af en kommende Superliga-kamp i sæson 2025/2026 står Palle, som har ansvar for den overordnede planlægning og koordinering omkring kampafvikling, over for en konkret beslutning: Skal der iværksættes ekstra tiltag op mod kampdagen – eksempelvis øget markedsføring, ekstra bemanning eller udvidet drift i boder og faciliteter – eller er det tilstrækkeligt at fastholde det nuværende setup?

Palle anvender D10 Prediction App'en som beslutningsværktøj. På baggrund af allerede solgte billetter 10 dage før kamp (2.733), kampens tidspunkt (kl. 18), runde og sæson estimerer modellen et forventet tilskuertal på ca. 6.081, svarende til 60,8 % af stadionkapaciteten.

## D10 — Prediction (vælg model)

**Model**  
Vælg model (default = bedste)  
OLS (lm)

**Inputs**  
Sæson (YYYY/YYYY)  
2025/2026  
runde  
4  
kamp\_time\_h  
18  
billetter\_d10 (heltal)  
2733  
Stadion-kapacitet  
10000  
Predict

### Output

Model: OLS (lm) — sæson 2025/2026 Forudsagt tilskuertal: 6.081

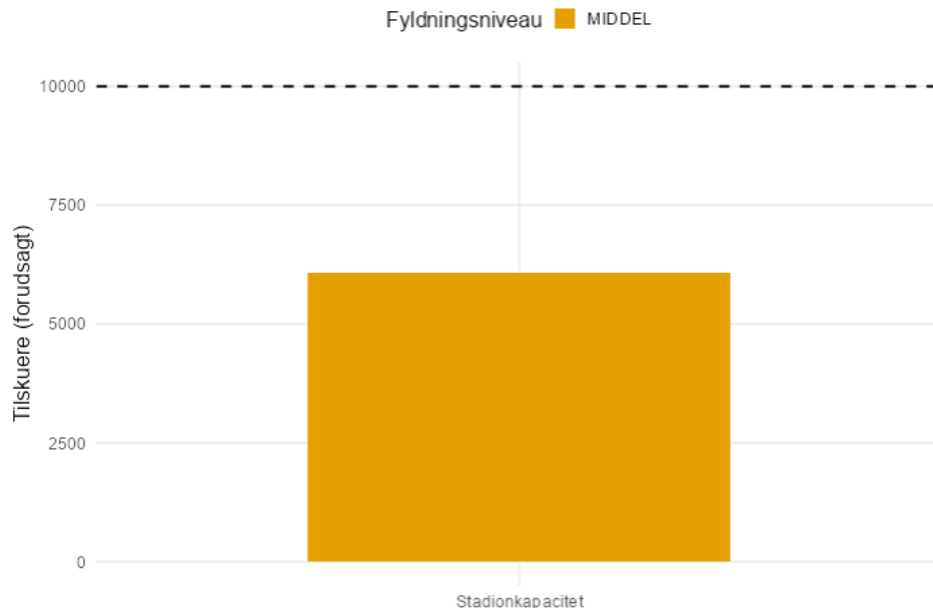
Stadion fyldt: 60,8 %

Ekstra billetter (D10 → kampdag): 3.348 ( $\hat{y}$  – billetter\_d10)

Ligning (kort):  $\hat{y} = -9600.991 + 4.577 \cdot \text{sæson\_start} + 3.746 \cdot \text{runde} + 42.392 \cdot \text{kamp\_time\_h} + 2.062 \cdot \text{billetter\_d10}$  (OLS)

### Forudsagt tilskuertal vs stadionkapacitet

Kapacitet (stiplet linje): 10.000



App'en indikerer samtidig, at der forventes et yderligere billetsalg på omkring 3.348 billetter frem mod kampdagen. Denne viden giver Palle et solidt, datadrevet grundlag for at vurdere, om der er behov for at opskalere driften – men også for at argumentere over for ledelsen for, at en fuld kapacitetsudvidelse ikke er nødvendig i dette scenarie.

Dermed fungerer prediction-modellen ikke blot som en prognose, men som et operationelt beslutningsstøtteværktøj, hvor forskellige scenarier hurtigt kan testes ved at justere input som billetstatus, kampstartstid og stadionkapacitet.

## 6 LONG-model uden leakage

Dette afsnit introducerer den samlede LONG-model for prognoser af tilskuertal. Scriptet samler historiske informationer om tilskuere og billetsalg i ét konsistent workflow, hvor alle tidsafhængige features er konstrueret uden data leakage. Datagrundlaget hentes fra Azure SQL og kobles til en deduplikeret baseline med én række pr. kamp. Modellen udvides med sæson- og rundeafhængig historik samt rolling kamp-historik, der udelukkende bygger på tidligere kampe. Den færdige LONG-model evalueres via year-split og danner grundlag for både statistisk modelvalg og en forklarlig, scenariebaseret Shiny-applikation.

## 6.1 Pakker og globale indstillinger

Denne kodeblok håndterer indlæsning og validering af alle pakker, som anvendes i LONG-workflowet. Scriptet stopper eksplicit, hvis en nødvendig pakke mangler, så fejl opdages tidligt. Derudover fastsættes globale indstillinger for output-formattering samt fælles konstanter, som bruges på tværs af resten af analysen.

```
needed_pkgs <- c("DBI","odbc","dplyr","lubridate","stringr","tibble","ggplot2","glmnet","slider")
missing_pkgs <- needed_pkgs[!vapply(needed_pkgs, requireNamespace, logical(1), quietly = TRUE)]
if (length(missing_pkgs) > 0) stop(" Manglende pakker: ", paste(missing_pkgs, collapse = ", "))

suppressPackageStartupMessages({
  library(DBI); library(odbc); library(dplyr); library(lubridate)
  library(stringr); library(tibble); library(ggplot2); library(glmnet)
  library(slider)
})

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

Pakker er klar

```
N_SHOW <- 20
```

## 6.2 Hjælpefunktioner

### 6.2.0.1 Validering af miljøvariabler

Kontrollerer at alle nødvendige miljøvariabler til Azure SQL er korrekt defineret i .Renviron og afbryder kørslen, hvis der mangler nogen.

```
ensure_renviron <- function() {
  req <- c("AZURE_SQL_SERVER","AZURE_SQL_DB","AZURE_SQL_UID","AZURE_SQL_PWD")
  vals <- Sys.getenv(req)
  miss <- req[vals == ""]
  if (length(miss) > 0) stop(" Manglende miljøvariabler i .Renviron: ", paste(miss, collapse = ", "))
  cat(" .Renviron OK\n\n")
}
```

### 6.2.0.2 Azure SQL-forbindelse med retry

Etablerer forbindelse til Azure SQL med gentagne forsøg og gradvist stigende timeout for at håndtere midlertidige forbindelsesproblemer.

```
connect_azure_retry <- function(
  forsøg_max = 6,
  timeouts = c(60, 180, 240, 360, 600, 900),
  delay_sec = 10
) {
  for (i in seq_len(forsøg_max)) {
    timeout_brug <- timeouts[min(i, length(timeouts))]
```

```

cat("Forsøg ", i, " / ", forsøg_max, " - ConnectionTimeout = ", timeout_brug, " sek\n", sep = "")
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"),
    Encrypt = "yes",
    TrustServerCertificate = "no",
    ConnectionTimeout = timeout_brug
  ),
  silent = TRUE
)
if (!inherits(con_try, "try-error")) {
  cat(" Forbundet til Azure SQL\n\n")
  return(con_try)
}
if (i < forsøg_max) Sys.sleep(delay_sec)
}
stop(" Kunne ikke forbinde til Azure SQL efter ", forsøg_max, " forsøg.")
}

```

### 6.2.0.3 Sikker indlæsning af tabel fra SQL

Forsøger at læse en tabel fra databasen med gentagne forsøg og korte pauser imellem, og stopper først hvis indlæsningen gentagne gange fejler.

```

safe_dbReadTable <- function(con, schema, table, forsøg_max = 4, delay_sec = 3) {
  for (i in seq_len(forsøg_max)) {
    out <- try(DBI::dbReadTable(con, DBI::Id(schema = schema, table = table)), silent = TRUE)
    if (!inherits(out, "try-error")) return(out)
    msg <- as.character(out)
    cat(" dbReadTable fejlede ", i, " / ", forsøg_max, "): ", schema, ".", table, "\n", sep = "")
    cat(substr(msg, 1, 220), "... \n\n")
    if (i < forsøg_max) Sys.sleep(delay_sec)
  }
  stop(" Kunne ikke læse tabel: ", schema, ".", table)
}

```

Deduplikering til én række pr. kamp

Reducerer datasættet til én række pr. kamp-id ved at vælge den observation med færrest manglende værdier.

```

dedup_1row_by_id <- function(df, id_col = "kamp_id") {
  stopifnot(id_col %in% names(df))
  df2 <- df
  df2$.na_count <- rowSums(is.na(df2))
  df2 |>
    arrange(.data[[id_col]], .na_count) |>

```

```

group_by(.data[[id_col]]) |>
slice(1) |>
ungroup() |>
select(-.na_count)
}

```

#### 6.2.0.4 Udtræk af sæsonens startår

Udleder sæsonens startår ved at udtrække det første årstal fra sæsonangivelsen.

```

season_start_year <- function(season_chr) {
  x <- as.character(season_chr)
  suppressWarnings(as.integer(stringr::str_extract(x, "\\d{4}")))
}

```

#### 6.2.0.5 Evalueringsmål for modelperformance

Beregner RMSE, MAE og  $R^2$  til vurdering af modellens prædiktive nøjagtighed.

```

rmse_vec <- function(y, yhat) sqrt(mean((y - yhat) ^ 2, na.rm = TRUE))
mae_vec  <- function(y, yhat) mean(abs(y - yhat), na.rm = TRUE)
r2_vec  <- function(y, yhat) {
  sse <- sum((y - yhat) ^ 2, na.rm = TRUE)
  sst <- sum((y - mean(y, na.rm = TRUE)) ^ 2, na.rm = TRUE)
  1 - (sse / sst)
}

```

#### 6.2.0.6 Year-split til træning og test

Opdeler datasættet i trænings- og testdata baseret på hele år, så de seneste år anvendes som testperiode.

```

make_year_split <- function(df, year_col = "år", n_test_years = 2) {
  stopifnot(year_col %in% names(df))
  yrs <- sort(unique(df[[year_col]]))
  if (length(yrs) < (n_test_years + 1)) stop(" For få år til et stabilt year-split.")
  test_years <- tail(yrs, n_test_years)
  train <- df |> filter(!(.data[[year_col]] %in% test_years))
  test  <- df |> filter(  .data[[year_col]] %in% test_years )
  list(train = train, test = test, test_years = test_years)
}

```

Estimering af OLS-model

Fitter en lineær regressionsmodel på træningsdata med de angivne forklarende variable.

```

fit_ols <- function(train_df, y, x_vars) {
  x_vars <- x_vars[x_vars %in% names(train_df)]
  f <- as.formula(paste(y, "~", paste(x_vars, collapse = " + ")))
  lm(f, data = train_df)
}

```

### 6.2.0.7 Opbygning af modelmatrix

Omdanner de valgte forklarende variable til en numerisk modelmatrix egnet til regulariserede modeller.

```
make_model_matrix <- function(df, x_vars) {  
  x_vars <- x_vars[x_vars %in% names(df)]  
  f <- as.formula(paste("~", paste(x_vars, collapse = " + ")))  
  mm <- model.matrix(f, df)  
  mm[, -1, drop = FALSE]  
}
```

### 6.2.0.8 Tilpasning af feature-kolonner

Sikrer at modelmatrixen matcher de forventede kolonner ved at tilføje manglende kolonner med nul og fjerne overskydende.

```
align_to_cols <- function(x, target_cols) {  
  x_cols <- colnames(x)  
  miss <- setdiff(target_cols, x_cols)  
  if (length(miss) > 0) {  
    add <- matrix(0, nrow = nrow(x), ncol = length(miss))  
    colnames(add) <- miss  
    x <- cbind(x, add)  
  }  
  extra <- setdiff(x_cols, target_cols)  
  if (length(extra) > 0) {  
    x <- x[, setdiff(colnames(x), extra), drop = FALSE]  
  }  
  x <- x[, target_cols, drop = FALSE]  
  x  
}
```

### 6.2.0.9 Evaluering af modeloutput

Beregner RMSE, MAE og  $R^2$  for testdata baseret på de genererede predictioner.

```
eval_models <- function(test_df, y, pred) {  
  yv <- as.numeric(test_df[[y]])  
  tibble(  
    RMSE = rmse_vec(yv, pred),  
    MAE = mae_vec(yv, pred),  
    R2 = r2_vec(yv, pred)  
  )  
}
```

### 6.2.0.10 Robust gennemsnitsberegning

Beregner gennemsnittet og returnerer NA, hvis der ikke findes gyldige numeriske værdier.

```
safe_mean <- function(x) {
  v <- as.numeric(x)
  v <- v[is.finite(v)]
  if (length(v) == 0) return(NA_real_)
  mean(v)
}
```

#### 6.2.0.11 Robust medianberegning

Beregner medianen og returnerer NA, hvis der ikke findes gyldige numeriske værdier.

```
safe_median <- function(x) {
  v <- as.numeric(x)
  v <- v[is.finite(v)]
  if (length(v) == 0) return(NA_real_)
  stats::median(v)
}
```

#### 6.2.0.12 Robust standardafvigelse

Beregner standardafvigelsen og returnerer NA, hvis der er for få gyldige observationer.

```
safe_sd <- function(x) {
  v <- as.numeric(x)
  v <- v[is.finite(v)]
  if (length(v) < 2) return(NA_real_)
  stats::sd(v)
}
```

#### 6.2.0.13 Rolling historik uden leakage

Beregner glidende gennemsnit og standardafvigelse udelukkende på tidligere observationer ved altid at forskyde data med ét lag før beregning.

```
# Rolling (NO LEAKAGE): altid LAG(1) før vi ruller
roll_mean_prev <- function(x, k) {
  x1 <- dplyr::lag(as.numeric(x), 1)
  slider::slide_dbl(x1, ~ mean(.x, na.rm = TRUE), .before = k - 1, .complete = FALSE)
}
roll_sd_prev <- function(x, k) {
  x1 <- dplyr::lag(as.numeric(x), 1)
  slider::slide_dbl(x1, ~ stats::sd(.x, na.rm = TRUE), .before = k - 1, .complete = FALSE)
}
```

### 6.3 Initialisering af miljø og databaseforbindelse

Validerer miljøvariabler, opretter forbindelse til Azure SQL og sikrer, at forbindelsen altid lukkes korrekt ved afslutning af scriptet.

```
# =====
# 2) .Renviron + connection
# =====
ensure_renvirion()
```

.Renviron OK

```
con <- connect_azure_retry()
```

Forsøg 1 / 6 - ConnectionTimeout = 60 sek  
 Forbundet til Azure SQL

## 6.4 Indlæsning, validering og klargøring af baseline

Indlæser baseline-datasættet fra disk, sikrer korrekte datatyper og kontrollerer, at alle nødvendige variable er til stede. Herefter udføres kvalitetssikring med fokus på dubletter, så datasættet reduceres til én række pr. kamp. Afslutningsvis udledes sæsonens startår, som anvendes i de efterfølgende historik- og lag-beregninger.

### 6.4.0.1 Indlæsning af baseline-fil

Definerer sti til baseline-datasættet og sikrer, at filen findes, før den anvendes i resten af workflowet.

```
baseline_dir <- "C:/Users/janpe/OneDrive/Skrivebord/PBA Dataanalyse/01_Første semester/1 Semester proj
baseline_file <- file.path(baseline_dir, "baseline_azure.rds")
stopifnot(file.exists(baseline_file))
```

### 6.4.0.2 Klargøring af baseline-data

Indlæser baseline-datasættet og sikrer korrekt datatyper for kamp-id og kampdato.

```
baseline <- readRDS(baseline_file) |>
  mutate(
    kamp_id = as.character(kamp_id),
    kamp_dato = as.Date(kamp_dato)
  )
```

### 6.4.0.3 Validering af baseline-struktur

Sikrer at alle nødvendige kolonner findes i baseline-datasættet og logger antal rækker samt unikke kamp-id'er.

```
stopifnot(all(c("kamp_id", "kamp_dato", "tilskuere", "sæson", "runde", "ugedag") %in% names(baseline)))
cat("Baseline loaded Rækker: ", nrow(baseline),
    " Unikke kamp_id: ", n_distinct(baseline$kamp_id), "\n\n", sep = "")
```

Baseline loaded Rækker: 259 Unikke kamp\_id: 254

#### 6.4.0.4 Tjek for dubletter i baseline

Identificerer kamp-id'er med flere observationer og rapporterer antallet som led i kvalitetssikring af baseline.

```
dup_ids <- baseline |> count(kamp_id) |> filter(n > 1)
cat("--- BASELINE QA: DUPLIKAT-TJEK ---\n")
```

```
--- BASELINE QA: DUPLIKAT-TJEK ---
```

```
cat("Antal kamp_id med dubletter: ", nrow(dup_ids), "\n\n", sep = "")
```

```
Antal kamp_id med dubletter: 5
```

#### 6.4.0.5 Håndtering af dubletter i baseline

Fjerner eventuelle dubletter, så der kun er én række pr. kamp-id, og bekræfter resultatet via et konsistentstjek.

```
if (nrow(dup_ids) > 0) {
  baseline <- dedup_1row_by_id(baseline, "kamp_id")
  stopifnot(nrow(baseline) == n_distinct(baseline$kamp_id))
  cat(" Baseline deduplikeret (1 række pr kamp_id)\n\n")
} else {
  cat(" Ingen dubletter fundet\n\n")
}
```

```
Baseline deduplikeret (1 række pr kamp_id)
```

#### Udledning af sæsonens startår

Tilføjer sæsonens startår til baseline og logger antallet af manglende værdier som et simpelt kvalitetstjek.

```
baseline <- baseline |>
  mutate(sæson_startår = season_start_year(sæson))
```

### 6.5 Feature load & transform

#### 6.5.1 Helligdage

Tilføjer en binær indikator for helligdag ved at koble kampdatoer til helligdagsdimensionen og sætte værdien til 1 på helligdage og 0 ellers.

```
# =====
# 4) FEATURE: Helligdage (1/0)
# =====
hellig_raw <- safe_dbReadTable(con, "PBA01_Raw", "dim_helligdage_dkk_raw")
stopifnot(all(c("dato", "helligdag_navn") %in% names(hellig_raw)))

hellig_min <- hellig_raw |>
  mutate(hellig_dato = as.Date(dato)) |>
```

```

filter(!is.na(hellig_dato)) |>
distinct(hellig_dato) |>
transmute(hellig_dato, er_helligdag = 1L)

baseline <- baseline |>
left_join(hellig_min, by = c("kamp_dato" = "hellig_dato")) |>
mutate(er_helligdag = if_else(is.na(er_helligdag), 0L, er_helligdag))

cat("Helligdage joinet \n\n")

```

Helligdage joinet

### 6.5.2 SAH-variabler sat til default

Nulstiller SAH-relaterede variable, så håndboldkampe ikke indgår som forklarende faktor i den videre analyse.

```

# =====
# 5) SAH droppes (default)
# =====
baseline <- baseline |>
  mutate(
    er_håndboldkamp_SAH = 0L,
    antal_håndboldkampe = 0L
  )
cat("SAH droppet (default) \n\n")

```

SAH droppet (default)

### 6.5.3 Feature: befolkningstal (LOCF)

Tilføjer befolkningstal til hver kamp ved at koble på den senest kendte befolkningsobservation før kampdatoen, så tidsmæssig konsistens bevares uden leakage.

### 6.5.4 Indlæsning af befolkningsdata

Henter befolkningsdata fra databasen og sikrer, at de nødvendige kolonner er til stede.

```

bef_raw <- safe_dbReadTable(con, "PBA03_JoinReady", "fact_Viborg_befolkning_join_ready")
stopifnot(all(c("køn", "civilstand", "dato", "befolkningstal") %in% names(bef_raw)))

```

### 6.5.5 Klargøring og filtrering af befolkningsdata

Standardiserer tekstfelter, sikrer korrekte datatyper og filtrerer til samlede befolkningstal pr. dato, som forberedes til tidsnær join.

```

bef_all <- bef_raw |>
  mutate(
    køn = str_to_lower(str_trim(as.character(køn))),
    civilstand = str_to_lower(str_trim(as.character(civilstand))),
    dato = as.Date(dato),
    befolkningstal = as.numeric(befolkningstal)
  ) |>
  filter(
    !is.na(dato),
    !is.na(befolkningstal),
    køn %in% c("i alt", "alt"),
    civilstand %in% c("i alt", "alt")
  ) |>
  distinct(dato, .keep_all = TRUE) |>
  arrange(dato) |>
  transmute(bef_dato = dato, befolkningstal = befolkningstal)

```

### 6.5.6 Kobling af befolkningstal til kampe

Tilføjer befolkningstal til hver kamp ved at bruge den nærmeste tidligere observationsdato og logger antallet af manglende værdier.

```

baseline_bef <- baseline |>
  transmute(kamp_id, kamp_dato) |>
  left_join(bef_all, join_by(closest(kamp_dato >= bef_dato)))

baseline <- baseline |>
  left_join(baseline_bef |> select(kamp_id, befolkningstal), by = "kamp_id")

cat("Befolkning jointet    NA: ", sum(is.na(baseline$befolkningstal)), "\n\n", sep = "")

```

```
Befolkning jointet    NA: 123
```

### 6.5.7 Kalenderbaserede features

Udleder år, måned, weekendindikator, sæsonperiode og runde-fase samt fastlægger ugedag som faktor med fast **orden** til videre modellering.

```

ugedag_levels <- c("Mandag", "Tirsdag", "Onsdag", "Torsdag", "Fredag", "Lørdag", "Søndag")

baseline <- baseline |>
  mutate(
    år = year(kamp_dato),
    måned = month(kamp_dato),
    weekend = if_else(as.character(ugedag) %in% c("Lørdag", "Søndag"), 1L, 0L),

    sæson_periode = case_when(
      måned %in% c(12, 1, 2) ~ "Vinter",
      måned %in% c(3, 4, 5) ~ "Forår",

```

```

    måned %in% c(6,7,8) ~ "Sommer",
    måned %in% c(9,10,11) ~ "Efterår",
    TRUE ~ "UKENDT"
  ),
  sæson_periode = as.factor(sæson_periode),

  runde_fase = case_when(
    runde <= 8 ~ "Tidlig",
    runde <= 20 ~ "Midt",
    TRUE ~ "Sen"
  ),
  runde_fase = as.factor(runde_fase),

  ugedag = factor(as.character(ugedag), levels = ugedag_levels)
)

cat("Kalenderkorrektioner lavet \n\n")

```

Kalenderkorrektioner lavet

## 6.5.8 Tilskuer-historik med lag uden leakage

Konstruerer historiske tilskuerfeatures baseret på sæson- og rundeniveau ved udelukkende at anvende laggede gennemsnit fra tidligere sæsoner. Disse historiske estimater joines på baseline og sikrer, at aktuelle kampe ikke påvirker deres egne forklarende variable.

### 6.5.8.1 Sæsonbaseret tilskuerhistorik

Beregner gennemsnitligt tilskuertal pr. sæson og udleder historiske sæsonniveauer ved hjælp af laggede værdier fra tidligere sæsoner.

```

season_mean <- baseline |>
  filter(!is.na(sæson_startår), !is.na(tilskuere)) |>
  group_by(sæson, sæson_startår) |>
  summarise(sæson_mean_tilskuere = mean(as.numeric(tilskuere), na.rm = TRUE), .groups = "drop") |>
  arrange(sæson_startår) |>
  mutate(
    lag1_mean = dplyr::lag(sæson_mean_tilskuere, 1),
    lag2_mean = dplyr::lag(sæson_mean_tilskuere, 2),
    hist_mean = if_else(!is.na(lag1_mean), lag1_mean, lag2_mean)
  ) |>
  select(sæson, sæson_startår, sæson_mean_tilskuere, lag1_mean, lag2_mean, hist_mean)

```

Kobling af sæsonhistorik til baseline

Tilføjer det historiske sæsonniveau for tilskuere til hver kamp via sæson-join.

```

baseline <- baseline |>
  left_join(season_mean |> select(sæson, hist_mean), by = "sæson")

```

Beregner gennemsnitligt tilskuertal pr. runde og udleder historiske rundeniveauer via lag fra tidligere sæsoner.

```
round_mean <- baseline |>
  filter(!is.na(sæson_startår), !is.na(runde), !is.na(tilskuere)) |>
  group_by(runde, sæson_startår) |>
  summarise(runde_mean_tilskuere = mean(as.numeric(tilskuere), na.rm = TRUE), .groups = "drop") |>
  arrange(runde, sæson_startår) |>
  group_by(runde) |>
  mutate(
    lag1_round_mean = dplyr::lag(runde_mean_tilskuere, 1),
    lag2_round_mean = dplyr::lag(runde_mean_tilskuere, 2),
    hist_round_mean = if_else(!is.na(lag1_round_mean), lag1_round_mean, lag2_round_mean)
  ) |>
  ungroup() |>
  select(runde, sæson_startår, hist_round_mean)
```

Samler sæson- og rundebaseret historik i baseline og logger antallet af manglende historiske værdier som kvalitetstjek.

```
baseline <- baseline |>
  left_join(round_mean, by = c("runde", "sæson_startår"))

cat("Tilskuer-historik jointet    NA hist_mean: ", sum(is.na(baseline$hist_mean)),
    "    NA hist_round_mean: ", sum(is.na(baseline$hist_round_mean)), "\n\n", sep = "")
```

```
Tilskuer-historik jointet    NA hist_mean: 17    NA hist_round_mean: 34
```

## 6.5.9 Billetdata og sæsonbaseret historik uden leakage

Indlæser billetsalgsdata fra Azure SQL, reducerer dem til én observation pr. kamp og kobler dem på baseline. Herefter konstrueres sæsonbaserede aggregater for billetsalg, hvor historiske niveauer udledes via lag fra tidligere sæsoner, så aktuelle kampe ikke påvirker deres egne forklarende variable.

### 6.5.9.1 Indlæsning af billetsalgsdata

Henter billetsalgsdata fra databasen og klargør kamp-id og billetrelaterede tal til videre behandling.

```
cat("--- TICKETS: load + sæson-aggregater (NO LEAKAGE) ---\n")
```

```
--- TICKETS: load + sæson-aggregater (NO LEAKAGE) ---
```

```
tickets_raw <- safe_dbReadTable(con, "PBA03_JoinReady", "fact_VFF_Billetsalg_join_ready") |>
  mutate(
    kamp_id = as.character(kamp_id),
    d10_tilskuere = suppressWarnings(as.numeric(d10_tilskuere))
  )
```

### 6.5.9.2 Validering af billetsalgsdata

Sikrer at de nødvendige kolonner findes og logger omfanget af billetsalgsdata samt antal unikke kampe.

```
stopifnot(all(c("kamp_id", "d10_tilskuere") %in% names(tickets_raw)))

cat("Billetsalg hentet    Rækker: ", nrow(tickets_raw), "\n", sep = "")
```

Billetsalg hentet Rækker: 259

```
cat("Unikke kamp_id i billetsalg: ", n_distinct(tickets_raw$kamp_id), "\n\n", sep = "")
```

Unikke kamp\_id i billetsalg: 254

### 6.5.9.3 Aggregering af billetsalg pr. kamp

Reducerer billetsalgsdata til én observation pr. kamp ved at vælge det højeste registrerede billetsalg og håndterer ugyldige værdier.

```
tickets_1row <- tickets_raw |>
  group_by(kamp_id) |>
  summarise(
    billetter_d10 = suppressWarnings(max(d10_tilskuere, na.rm = TRUE)),
    .groups = "drop"
  ) |>
  mutate(billetter_d10 = if_else(is.infinite(billetter_d10), NA_real_, billetter_d10))
```

### 6.5.9.4 Kobling af billetsalg til baseline

Tilføjer billetsalgsvARIABLEN til baseline og logger antallet af manglende værdier som kvalitetstjek.

```
baseline <- baseline |>
  left_join(tickets_1row, by = "kamp_id")

cat("billetter_d10 joinet på baseline    NA billetter_d10: ", sum(is.na(baseline$billetter_d10)), "\n\n")
```

billetter\_d10 joinet på baseline NA billetter\_d10: 0

### 6.5.9.5 Sæsonbaserede billetaggregater

Sammenfatter billetsalget pr. sæson ved hjælp af robuste mål for niveau, spredning og datadækning. For hver sæson konstrueres historiske billetfeatures ved at anvende lag fra tidligere sæsoner, så de afspejler forventet billetsalgsniveau uden at inddrage information fra den aktuelle sæson.

```
ticket_season <- baseline |>
  filter(!is.na(sæson_startår)) |>
  group_by(sæson, sæson_startår) |>
  summarise(
    ticket_mean_d10    = safe_mean(billetter_d10),
    ticket_median_d10  = safe_median(billetter_d10),
    ticket_sd_d10      = safe_sd(billetter_d10),
    ticket_n_d10       = sum(is.finite(as.numeric(billetter_d10))),
```

```

    .groups = "drop"
  ) |>
  arrange(sæson_startår) |>
  mutate(
    lag1_ticket_mean    = dplyr::lag(ticket_mean_d10, 1),
    lag2_ticket_mean    = dplyr::lag(ticket_mean_d10, 2),

    lag1_ticket_median = dplyr::lag(ticket_median_d10, 1),
    lag2_ticket_median = dplyr::lag(ticket_median_d10, 2),

    lag1_ticket_sd      = dplyr::lag(ticket_sd_d10, 1),
    lag2_ticket_sd      = dplyr::lag(ticket_sd_d10, 2),

    lag1_ticket_n       = dplyr::lag(ticket_n_d10, 1),
    lag2_ticket_n       = dplyr::lag(ticket_n_d10, 2),

    hist_ticket_mean_d10 = if_else(!is.na(lag1_ticket_mean), lag1_ticket_mean, lag2_ticket_mean),
    hist_ticket_median_d10 = if_else(!is.na(lag1_ticket_median), lag1_ticket_median, lag2_ticket_median),
    hist_ticket_sd_d10    = if_else(!is.na(lag1_ticket_sd), lag1_ticket_sd, lag2_ticket_sd),
    hist_ticket_n_d10     = if_else(!is.na(lag1_ticket_n), lag1_ticket_n, lag2_ticket_n),

    hist_ticket_mean2_d10 = case_when(
      !is.na(lag1_ticket_mean) & !is.na(lag2_ticket_mean) ~ (lag1_ticket_mean + lag2_ticket_mean) / 2,
      !is.na(lag1_ticket_mean) ~ lag1_ticket_mean,
      !is.na(lag2_ticket_mean) ~ lag2_ticket_mean,
      TRUE ~ NA_real_
    )
  ) |>
  transmute(
    sæson,
    sæson_startår,
    ticket_mean_d10,
    ticket_median_d10,
    ticket_sd_d10,
    ticket_n_d10,
    hist_ticket_mean_d10,
    hist_ticket_median_d10,
    hist_ticket_sd_d10,
    hist_ticket_n_d10,
    hist_ticket_mean2_d10
  )
View(ticket_season, "ticket_season_lags")

```

#### 6.5.9.6 Kobling af sæsonbaseret billethistorik

Sammenfletter sæsonbaserede historiske billetfeatures med baseline og logger omfanget af manglende historiske værdier som et afsluttende kvalitetstjek.

```

baseline <- baseline |>
  left_join(
    ticket_season |>

```

```

select(
  sæson,
  hist_ticket_mean_d10,
  hist_ticket_median_d10,
  hist_ticket_sd_d10,
  hist_ticket_n_d10,
  hist_ticket_mean2_d10
),
by = "sæson"
)
cat("Ticket-historik jointet    NA hist_ticket_mean_d10: ", sum(is.na(baseline$hist_ticket_mean_d10)),
    "    NA hist_ticket_mean2_d10: ", sum(is.na(baseline$hist_ticket_mean2_d10)), "\n\n", sep = "")

```

```

Ticket-historik jointet    NA hist_ticket_mean_d10: 17    NA hist_ticket_mean2_d10: 17

```

### 6.5.9.7 Rolling kamp-historik uden leakage

Initialiserer beregning af kortsigtede historikfeatures for både billetsalg og tilskuertal baseret udelukkende på tidligere kampe.

Logger start på rolling-beregninger og tjekker, om kampens tidspunkt findes, så sortering kan ske korrekt før historik beregnes.

```

cat("--- Rolling historik: billetter_d10 + tilskuere (NO LEAKAGE) ---\n")

```

```

--- Rolling historik: billetter_d10 + tilskuere (NO LEAKAGE) ---

```

```

has_time <- "kamp_time_h" %in% names(baseline)

```

### 6.5.9.8 Rolling historikfeatures pr. kamp

Beregner glidende gennemsnit, spredning og trends for billetsalg og tilskuertal inden for hver sæson, udelukkende baseret på tidligere kampe, og logger manglende værdier som kvalitetstjek.

```

baseline <- baseline |>
  arrange(
    sæson_startår,
    kamp_dato,
    if (has_time) kamp_time_h else 0
  ) |>
  group_by(sæson) |>
  mutate(
    kamp_nr_i_sæson = dplyr::row_number(),

    ticket_roll_mean_5  = roll_mean_prev(billetter_d10, 5),
    ticket_roll_mean_15 = roll_mean_prev(billetter_d10, 15),
    ticket_roll_sd_5    = roll_sd_prev(billetter_d10, 5),
    ticket_roll_sd_15   = roll_sd_prev(billetter_d10, 15),
    ticket_trend_5_15   = ticket_roll_mean_5 - ticket_roll_mean_15,

```

```

    tilsk_roll_mean_5    = roll_mean_prev(tilskuere, 5),
    tilsk_roll_mean_15   = roll_mean_prev(tilskuere, 15),
    tilsk_trend_5_15     = tilsk_roll_mean_5 - tilsk_roll_mean_15
  ) |>
  ungroup()

cat("Rolling features lavet    NA ticket_roll_mean_5: ", sum(is.na(baseline$ticket_roll_mean_5)),
    "    NA tilsk_roll_mean_5: ", sum(is.na(baseline$tilsk_roll_mean_5)), "\n\n", sep = "")

```

```
Rolling features lavet    NA ticket_roll_mean_5: 16    NA tilsk_roll_mean_5: 16
```

#### 6.5.9.9 Filtrering til komplet LONG-datasæt

Udvælger observationer med fulde og konsistente værdier for respons, kalenderfeatures og historiske variable, så datasættet er egnet til modellering uden manglende centrale input.

```

analysis_df_long_full <- baseline |>
  filter(
    !is.na(tilskuere),
    !is.na(runde),
    !is.na(ugedag),
    !is.na(år),
    !is.na(måned),
    !is.na(weekend),
    !is.na(sæson_periode),
    !is.na(runde_fase),
    !is.na(hist_mean)
  )

```

#### 6.5.9.10 Datasæt med og uden befolkningstal

Opretter en udvidet version af LONG-datasættet med befolkningstal og logger størrelsen af begge datasæt til sammenligning.

```

analysis_df_long_bef <- analysis_df_long_full |>
  filter(!is.na(befolkningstal))

cat("analysis_df_long_full klar    Rækker: ", nrow(analysis_df_long_full), "\n", sep = "")

```

```
analysis_df_long_full klar    Rækker: 237
```

```
cat("analysis_df_long_bef klar    Rækker: ", nrow(analysis_df_long_bef), "\n\n", sep = "")
```

```
analysis_df_long_bef klar    Rækker: 131
```

### 6.5.9.11 Definition af LONG feature-sæt

Fastlægger de forklarende variable til LONG-modellen, herunder kalenderfeatures, historiske sæson- og rundeniveauer, billetsalgshistorik samt rolling momentum-features, og filtrerer dem til de variable, der faktisk findes i datasættet.

```
base_long_vars <- c(
  "runde_fase",
  "ugedag",
  "weekend",
  "måned",
  "år",
  "er_helligdag",
  "sæson_periode",
  "hist_mean",
  "hist_round_mean",

  # Ticket-historik (sæson-lag)
  "hist_ticket_mean_d10",
  "hist_ticket_mean2_d10",
  "hist_ticket_median_d10",
  "hist_ticket_sd_d10",
  "hist_ticket_n_d10",

  # Rolling kamp-historik (momentum)
  "ticket_roll_mean_5",
  "ticket_roll_mean_15",
  "ticket_trend_5_15",
  "ticket_roll_sd_5",
  "tilsk_roll_mean_5",
  "tilsk_roll_mean_15",
  "tilsk_trend_5_15"
)

base_long_vars <- base_long_vars[base_long_vars %in% names(analysis_df_long_full)]

cat("LONG feature-sæt:\n")
```

LONG feature-sæt:

```
print(base_long_vars)
```

[1] "runde_fase"	"ugedag"	"weekend"
[4] "måned"	"år"	"er_helligdag"
[7] "sæson_periode"	"hist_mean"	"hist_round_mean"
[10] "hist_ticket_mean_d10"	"hist_ticket_mean2_d10"	"hist_ticket_median_d10"
[13] "hist_ticket_sd_d10"	"hist_ticket_n_d10"	"ticket_roll_mean_5"
[16] "ticket_roll_mean_15"	"ticket_trend_5_15"	"ticket_roll_sd_5"
[19] "tilsk_roll_mean_5"	"tilsk_roll_mean_15"	"tilsk_trend_5_15"

```
cat("\n")
```

### 6.5.10 Endelig modelkørsel

Denne del samler hele LONG-setup'et i en ensartet evalueringsfunktion, hvor datasættet opdeles i træning og test via year-split. Først etableres simple baselines, som fungerer som referencepunkter, hvorefter OLS, Ridge og Lasso estimeres på samme split. Modellerne sammenlignes på tværs af RMSE, MAE og  $R^2$ , og resultaterne samles i et leaderboard. Afslutningsvis køres processen både uden og med befolkningstal for at vurdere stabilitet og marginal gevinst ved den ekstra feature.

```
# =====
# 12) FINAL: YEAR SPLIT + Baselines + OLS / Ridge / Lasso
# =====
run_long_final_yearsplit <- function(df, y = "tilskuere", x_vars, label = "FULL", n_test_years = 2){
  split <- make_year_split(df, year_col = "år", n_test_years = n_test_years)
  tr <- split$train
  te <- split$test

  cat("=== LONG FINAL (YEAR SPLIT): ", label, " ===\n", sep = "")
  cat("Test-år: ", paste(split$test_years, collapse = ", "), "\n", sep = "")
  cat("Train: ", nrow(tr), "   Test: ", nrow(te), "\n\n", sep = "")

  # -----
  # A) BASELINES
  # -----
  train_mean <- mean(as.numeric(tr[[y]]), na.rm = TRUE)

  pred_mean_train <- rep(train_mean, nrow(te))
  met_mean <- eval_models(te, y, pred_mean_train) |> mutate(model = "BASE_mean_train", dataset = label)

  pred_hist_mean <- as.numeric(te$hist_mean)
  met_hist <- eval_models(te, y, pred_hist_mean) |> mutate(model = "BASE_hist_mean", dataset = label)

  pred_hist_round <- ifelse(!is.na(te$hist_round_mean), as.numeric(te$hist_round_mean), as.numeric(te$
  met_hist_round <- eval_models(te, y, pred_hist_round) |> mutate(model = "BASE_hist_round_mean", data

  baselines <- bind_rows(met_mean, met_hist, met_hist_round) |> arrange(RMSE)

  cat("--- BASELINES (test) ---\n")
  print(baselines)
  cat("\n")

  # -----
  # B) MODELS
  # -----
  tr2 <- tr
  te2 <- te

  tr2$hist_round_mean <- ifelse(is.na(tr2$hist_round_mean), tr2$hist_mean, tr2$hist_round_mean)
  te2$hist_round_mean <- ifelse(is.na(te2$hist_round_mean), te2$hist_mean, te2$hist_round_mean)
```

```

# Imputer numeriske features (rolling + ticket-historik) med train-mean
# (tidlige kampe i sæsonen giver typisk NA i rolling)
for (cc in x_vars) {
  if (!cc %in% names(tr2)) next
  if (is.numeric(tr2[[cc]]) || is.integer(tr2[[cc]])) {
    mu <- mean(as.numeric(tr2[[cc]]), na.rm = TRUE)
    if (is.finite(mu)) {
      tr2[[cc]] <- ifelse(is.na(tr2[[cc]]), mu, tr2[[cc]])
      te2[[cc]] <- ifelse(is.na(te2[[cc]]), mu, te2[[cc]])
    }
  }
}

# OLS
m_ols <- fit_ols(tr2, y, x_vars)
p_ols <- as.numeric(predict(m_ols, newdata = te2))
met_ols <- eval_models(te2, y, p_ols) |> mutate(model = "OLS", dataset = label)

# Ridge/Lasso
x_tr <- make_model_matrix(tr2, x_vars)
x_te <- make_model_matrix(te2, x_vars)

target_cols <- sort(colnames(x_tr))
x_tr <- x_tr[, target_cols, drop = FALSE]
x_te <- align_to_cols(x_te, target_cols)

y_tr <- as.numeric(tr2[[y]])

set.seed(42)
cv_ridge <- cv.glmnet(x_tr, y_tr, alpha = 0, nfolds = 10, standardize = TRUE)
p_ridge <- as.numeric(predict(cv_ridge, newx = x_te, s = "lambda.min"))
met_ridge <- eval_models(te2, y, p_ridge) |> mutate(model = "Ridge", dataset = label)

set.seed(42)
cv_lasso <- cv.glmnet(x_tr, y_tr, alpha = 1, nfolds = 10, standardize = TRUE)
p_lasso <- as.numeric(predict(cv_lasso, newx = x_te, s = "lambda.min"))
met_lasso <- eval_models(te2, y, p_lasso) |> mutate(model = "Lasso", dataset = label)

models <- bind_rows(met_ols, met_ridge, met_lasso) |> arrange(RMSE)

cat("--- MODELS (test) ---\n")
print(models)
cat("\n")

leaderboard <- bind_rows(baselines, models) |> arrange(RMSE)

cat("--- LEADERBOARD (baselines + models) ---\n")
print(leaderboard)
cat("\n")

artifact <- list(

```

```

    label = label,
    features = x_vars,
    split_info = list(train_n = nrow(tr2), test_n = nrow(te2), test_years = split$test_years),
    train_mean = train_mean,
    baselines = baselines,
    metrics_models = models,
    leaderboard = leaderboard,
    ols = m_ols,
    ridge = cv_ridge,
    lasso = cv_lasso,
    glmnet_cols = target_cols
  )

  list(artifact = artifact, leaderboard = leaderboard)
}

# Kør FULL (uden befolkning)
res_full <- run_long_final_yearsplit(
  df = analysis_df_long_full,
  x_vars = base_long_vars,
  label = "FULL_long_plus_ticket_plus_rolling",
  n_test_years = 2
)

```

=== LONG FINAL (YEAR SPLIT): FULL\_long\_plus\_ticket\_plus\_rolling ===  
 Test-år: 2024, 2025  
 Train: 204 Test: 33

--- BASELINES (test) ---

```

# A tibble: 3 x 5
  RMSE  MAE    R2 model          dataset
<dbl> <dbl> <dbl> <chr>          <chr>
1  994.  802. -0.0161 BASE_hist_mean  FULL_long_plus_ticket_plus_rolling
2 1932. 1675. -2.84  BASE_mean_train  FULL_long_plus_ticket_plus_rolling
3 2115. 1761. -3.60  BASE_hist_round_mean  FULL_long_plus_ticket_plus_rolling

```

--- MODELS (test) ---

```

# A tibble: 3 x 5
  RMSE  MAE    R2 model dataset
<dbl> <dbl> <dbl> <chr> <chr>
1 1146.  833. -0.350 Ridge FULL_long_plus_ticket_plus_rolling
2 1156.  863. -0.374 Lasso FULL_long_plus_ticket_plus_rolling
3 1434. 1168. -1.11  OLS   FULL_long_plus_ticket_plus_rolling

```

--- LEADERBOARD (baselines + models) ---

```

# A tibble: 6 x 5
  RMSE  MAE    R2 model          dataset
<dbl> <dbl> <dbl> <chr>          <chr>
1  994.  802. -0.0161 BASE_hist_mean  FULL_long_plus_ticket_plus_rolling
2 1146.  833. -0.350 Ridge          FULL_long_plus_ticket_plus_rolling
3 1156.  863. -0.374 Lasso          FULL_long_plus_ticket_plus_rolling

```

```

4 1434. 1168. -1.11 OLS FULL_long_plus_ticket_plus_rolling
5 1932. 1675. -2.84 BASE_mean_train FULL_long_plus_ticket_plus_rolling
6 2115. 1761. -3.60 BASE_hist_round_mean FULL_long_plus_ticket_plus_rolling

```

```

# Kjør BEF (med befolkning) - ofte ustabilt pga få rækker, men vi tester
vars_bef <- unique(c(base_long_vars, "befolkningstal"))
vars_bef <- vars_bef[vars_bef %in% names(analysis_df_long_bef)]

res_bef <- run_long_final_yearsplit(
  df = analysis_df_long_bef,
  x_vars = vars_bef,
  label = "BEF_long_plus_ticket_plus_rolling",
  n_test_years = 2
)

```

```

=== LONG FINAL (YEAR SPLIT): BEF_long_plus_ticket_plus_rolling ===
Test-år: 2024, 2025
Train: 98 Test: 33

```

```

--- BASELINES (test) ---

```

```

# A tibble: 3 x 5
  RMSE MAE R2 model dataset
<dbl> <dbl> <dbl> <chr> <chr>
1 994. 802. -0.0161 BASE_hist_mean BEF_long_plus_ticket_plus_rolling
2 1602. 1303. -1.64 BASE_mean_train BEF_long_plus_ticket_plus_rolling
3 2115. 1761. -3.60 BASE_hist_round_mean BEF_long_plus_ticket_plus_rolling

```

```

--- MODELS (test) ---

```

```

# A tibble: 3 x 5
  RMSE MAE R2 model dataset
<dbl> <dbl> <dbl> <chr> <chr>
1 1070. 844. -0.176 Ridge BEF_long_plus_ticket_plus_rolling
2 1132. 867. -0.318 Lasso BEF_long_plus_ticket_plus_rolling
3 62993. 49292. -4076. OLS BEF_long_plus_ticket_plus_rolling

```

```

--- LEADERBOARD (baselines + models) ---

```

```

# A tibble: 6 x 5
  RMSE MAE R2 model dataset
<dbl> <dbl> <dbl> <chr> <chr>
1 994. 802. -0.0161 BASE_hist_mean BEF_long_plus_ticket_plus_rolli~
2 1070. 844. -0.176 Ridge BEF_long_plus_ticket_plus_rolli~
3 1132. 867. -0.318 Lasso BEF_long_plus_ticket_plus_rolli~
4 1602. 1303. -1.64 BASE_mean_train BEF_long_plus_ticket_plus_rolli~
5 2115. 1761. -3.60 BASE_hist_round_mean BEF_long_plus_ticket_plus_rolli~
6 62993. 49292. -4076. OLS BEF_long_plus_ticket_plus_rolli~

```

```

cat("\n--- LEADERBOARD (FULL) ---\n")

```

```

--- LEADERBOARD (FULL) ---

```

```
print(res_full$leaderboard)
```

```
# A tibble: 6 x 5
  RMSE  MAE    R2 model          dataset
<dbl> <dbl> <dbl> <chr>         <chr>
1  994.  802. -0.0161 BASE_hist_mean FULL_long_plus_ticket_plus_rolling
2 1146.  833. -0.350  Ridge         FULL_long_plus_ticket_plus_rolling
3 1156.  863. -0.374  Lasso         FULL_long_plus_ticket_plus_rolling
4 1434. 1168. -1.11   OLS           FULL_long_plus_ticket_plus_rolling
5 1932. 1675. -2.84   BASE_mean_train FULL_long_plus_ticket_plus_rolling
6 2115. 1761. -3.60   BASE_hist_round_mean FULL_long_plus_ticket_plus_rolling
```

```
cat("\n--- LEADERBOARD (BEF) ---\n")
```

```
--- LEADERBOARD (BEF) ---
```

```
print(res_bef$leaderboard)
```

```
# A tibble: 6 x 5
  RMSE  MAE    R2 model          dataset
<dbl> <dbl> <dbl> <chr>         <chr>
1  994.  802. -0.0161 BASE_hist_mean BEF_long_plus_ticket_plus_rolli~
2 1070.  844. -0.176  Ridge         BEF_long_plus_ticket_plus_rolli~
3 1132.  867. -0.318  Lasso         BEF_long_plus_ticket_plus_rolli~
4 1602. 1303. -1.64   BASE_mean_train BEF_long_plus_ticket_plus_rolli~
5 2115. 1761. -3.60   BASE_hist_round_mean BEF_long_plus_ticket_plus_rolli~
6 62993. 49292. -4076. OLS           BEF_long_plus_ticket_plus_rolli~
```

### 6.5.10.1 Evaluering

I denne del samles hele LONG-setup'et i én ensartet evalueringsprocedure, der har til formål systematisk at sammenligne simple baselines med mere avancerede regressionsmodeller. Datasættet opdeles konsekvent i træning og test ved hjælp af et year-split, hvor de seneste to år anvendes som testperiode. Denne tilgang sikrer, at evalueringen af modellerne afspejler en realistisk prognosesituation uden dataleakage fra fremtidige observationer.

Som første trin etableres en række baselines, der fungerer som referencepunkter for modellernes præstation. Disse består af et simpelt gennemsnit beregnet på træningsdata, et historisk gennemsnit baseret på tidligere observationer samt et rundespecifikt historisk gennemsnit, hvor der differentieres mellem runder, når data tillader det. Baselines evalueres på testdatasættet ved hjælp af RMSE, MAE og  $R^2$  og giver et klart billede af, hvor stor en del af variationen i tilskuertallene der kan forklares uden brug af egentlige forklarende modeller.

Herefter estimeres tre regressionsbaserede modeller på samme datasplit: en klassisk OLS-model samt Ridge- og Lasso-regression. Før estimering håndteres manglende værdier i numeriske features, særligt rolling features og ticket-historik, ved imputering med gennemsnittet fra træningsdata. Denne løsning er valgt for at bevare observationsgrundlaget, især tidligt i sæsonerne hvor rolling features naturligt indeholder NA-værdier. Alle modeller trænes udelukkende på træningsdata og evalueres derefter på testdatasættet ved hjælp af de samme performance-metrikker som baselines.

Resultaterne fra både baselines og modeller samles i et fælles leaderboard, hvor modellerne rangeres efter RMSE. Dette giver en direkte og gennemsigtig sammenligning af, om de mere komplekse modeller reelt tilfører prædiktiv værdi i forhold til simple historiske referencepunkter. Evalueringen viser, at regulariserede modeller som Ridge og

Lasso reducerer fejl i forhold til en ureguleret OLS-model, men samtidig at den bedste historiske baseline fortsat præsterer på niveau med eller bedre end de estimerede modeller på testperioden.

Afslutningsvis gennemføres hele evalueringsproceduren både uden og med befolkningstal som ekstra feature. Formålet er ikke primært at forbedre den samlede performance, men at vurdere modellernes stabilitet og den marginale informationsværdi af befolkningstal givet det reducerede datagrundlag. Resultaterne dokumenterer, at tilføjelsen af befolkningstal medfører færre observationer og øget usikkerhed, uden at der opnås en tydelig forbedring i prognosekvaliteten.

Samlet set dokumenterer denne endelige modelkørsel, at historisk information udgør den stærkeste forklaringskomponent i det nuværende datasæt, og at mere avancerede modeller i denne opsætning ikke formår entydigt at overgå en velvalgt historisk baseline. Afsnittet fungerer dermed som empirisk dokumentation for både modellernes præstation og de metodiske valg, der ligger til grund for den endelige evaluering.

#### 6.5.10.2 Robust prediktionsfunktion baseret på bedste model

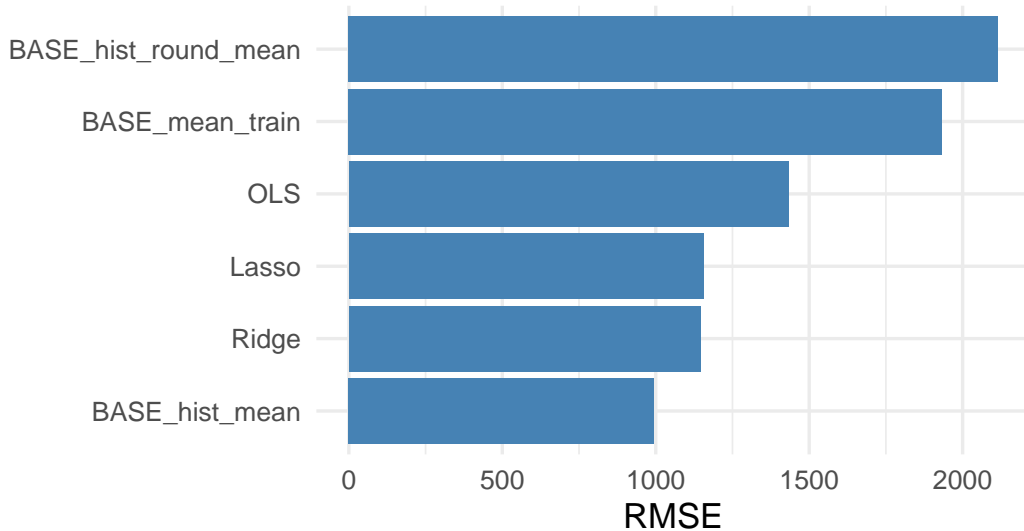
Dette afsnit dokumenterer den endelige prediktionsopsætning for LONG-modellen, hvor formålet er at sikre en stabil og reproducerbar måde at generere prognoser på baggrund af tidligere estimerede modeller. I stedet for manuelt at vælge modeltype anvendes et fælles artefakt, som indeholder både baselines, regressionsmodeller og tilhørende metadata fra den afsluttende modelkørsel.

Funktionen identificerer automatisk den bedst præsterende model baseret på test-RMSE og anvender denne til efterfølgende prediction. Opsætningen understøtter både simple baselines og mere komplekse modeller (OLS, Ridge og Lasso) og håndterer samtidig praktiske udfordringer som manglende features, faktorniveauer og kolonnejustering i modelmatricer. Dermed fungerer funktionen som et robust bindeled mellem analysefasen og egentlig anvendelse i fx Shiny, scenarieberegninger eller operationelle prognoser.

```
### Graff leaderboard
res_full$leaderboard |>
  ggplot(aes(x = reorder(model, RMSE), y = RMSE)) +
  geom_col(fill = "steelblue") +
  coord_flip() +
  labs(
    title = "Modelperformance - LONG",
    subtitle = "Test-år: 2024-2025",
    x = NULL,
    y = "RMSE"
  ) +
  theme_minimal(base_size = 13)
```

## Modelperformance – LONG

Test-år: 2024–2025



Her sammenlignes modelperformance i LONG-setup'et på testårene 2024–2025 målt ved RMSE. Grafen viser tydeligt, at de simple historiske baselines performer markant forskelligt: BASE\_hist\_mean har den laveste fejl og fungerer som det stærkeste referencepunkt, mens både BASE\_mean\_train og især BASE\_hist\_round\_mean har væsentligt højere RMSE. De estimerede modeller (OLS, Ridge og Lasso) reducerer fejlen i forhold til den naive træningsgennemsnits-baseline, men formår ikke konsekvent at forbedre sig i forhold til det rene historiske gennemsnit. Det indikerer, at størstedelen af den forklarende kraft i LONG-horisonten allerede ligger i det historiske niveau, og at de ekstra forklarende variable og regularisering primært bidrager med stabilitet og struktur snarere end egentlig præcisionsgevinst. Grafen understøtter dermed valget af en enkel, historikbaseret model som operationelt udgangspunkt, samtidig med at de mere komplekse modeller anvendes som metodisk benchmark og robusthedstjek.

```
# =====
# 13) predict_best_long() - robust (inkl baselines + stable glmnet align)
# =====
pick_best <- function(artifact) artifact$leaderboard |> arrange(RMSE) |> slice(1)

predict_best_long <- function(new_df, artifact) {
  best <- pick_best(artifact)
  model_name <- best$model[[1]]

  needed <- unique(c(artifact$features, "hist_mean", "hist_round_mean"))
  missing <- setdiff(needed, names(new_df))
  if (length(missing) > 0) {
    return(tibble(status = "MISSING_COLUMNS", missing = paste(missing, collapse = ", "), prediction =
  )

  if ("ugedag" %in% names(new_df)) {
    new_df$ugedag <- factor(as.character(new_df$ugedag),
                           levels = c("Mandag", "Tirsdag", "Onsdag", "Torsdag", "Fredag", "Lørdag", "Søndag")
  }

  new_df$hist_round_mean <- ifelse(is.na(new_df$hist_round_mean), new_df$hist_mean, new_df$hist_round_mean)

  # Imputer numeriske features med train-mean fra artifact (for stabil inference)
```

```

# (her bruger vi train_mean for simplicitet; kan udvides til per-feature mean hvis du vil)
for (cc in artifact$features) {
  if (!cc %in% names(new_df)) next
  if (is.numeric(new_df[[cc]]) || is.integer(new_df[[cc]])) {
    if (any(is.na(new_df[[cc]]))) {
      mu <- suppressWarnings(mean(as.numeric(new_df[[cc]]), na.rm = TRUE))
      if (!is.finite(mu)) mu <- NA_real_
      new_df[[cc]] <- ifelse(is.na(new_df[[cc]]), mu, new_df[[cc]])
    }
  }
}

# Baselines
if (model_name == "BASE_mean_train") {
  return(tibble(status = "OK", model = "BASE_mean_train", prediction = rep(artifact$train_mean, nrow(new_df))))
}
if (model_name == "BASE_hist_mean") {
  return(tibble(status = "OK", model = "BASE_hist_mean", prediction = as.numeric(new_df$hist_mean)))
}
if (model_name == "BASE_hist_round_mean") {
  pred <- ifelse(is.na(new_df$hist_round_mean), new_df$hist_mean, new_df$hist_round_mean)
  return(tibble(status = "OK", model = "BASE_hist_round_mean", prediction = as.numeric(pred)))
}

# OLS
if (model_name == "OLS") {
  pred <- as.numeric(predict(artifact$ols, newdata = new_df))
  return(tibble(status = "OK", model = "OLS", prediction = pred))
}

# Ridge/Lasso
x <- make_model_matrix(new_df, artifact$features)
x <- align_to_cols(x, artifact$glmnet_cols)

if (model_name == "Ridge") {
  pred <- as.numeric(predict(artifact$ridge, newx = x, s = "lambda.min"))
  return(tibble(status = "OK", model = "Ridge", prediction = pred))
}
if (model_name == "Lasso") {
  pred <- as.numeric(predict(artifact$lasso, newx = x, s = "lambda.min"))
  return(tibble(status = "OK", model = "Lasso", prediction = pred))
}

tibble(status = "UNKNOWN_MODEL", model = model_name, prediction = NA_real_)
}

cat("\n LONG master script (tickets + sæson-lag + rolling uden leakage) færdig.\n")

```

LONG master script (tickets + sæson-lag + rolling uden leakage) færdig.

Den endelige prediktionsfunktion dokumenterer, at den stærkeste prognose i det nuværende datagrundlag ofte opnås via historiske referencepunkter snarere end komplekse regressionsmodeller. På trods af et rigt feature-sæt med kalenderinformation, billetdata og rolling momentum-features formår modellerne ikke systematisk at overgå den bedste historiske baseline på testperioden.

Dette resultat peger på, at variationen i tilskuertal i høj grad er struktureret omkring historiske mønstre, og at marginal gevinster fra yderligere modellering er begrænsede givet datamængde og støjniveau. Den valgte arkitektur sikrer dog, at hvis dette ændrer sig – eksempelvis ved flere observationer, nye features eller ændrede strukturer – vil den bedste model automatisk blive anvendt uden ændringer i den operationelle kode.

Afsnittet fungerer dermed som empirisk dokumentation for både modelvalg og implementeringsstrategi og danner et solidt grundlag for videre brug af modellen i praksis.

## 6.6 Introduktion til scenarie-prognose-shiny app (LONG)

Denne Shiny-app er udviklet som et forsøg på at operationalisere den langsigtede tilskuermodel. Formålet er at vise, hvordan modelresultater kan omsættes til et simpelt scenarieestimat, der kan anvendes i en planlægningssammenhæng. Appen bygger på den mest stabile model i evalueringen og prioriterer forklarlighed frem for kompleksitet.

### 6.6.1 Pakker

Her tjekkes, at de nødvendige pakker til Shiny-appen er installeret, hvorefter de indlæses uden unødige konsolbeskeder.

```
# shiny

needed_pkgs_app <- c("shiny","dplyr","ggplot2","tibble")
missing_pkgs_app <- needed_pkgs_app[!vapply(needed_pkgs_app, requireNamespace, logical(1), quietly = T)]
if (length(missing_pkgs_app) > 0) stop(" Manglende pakker (app): ", paste(missing_pkgs_app, collapse = ", "))

suppressPackageStartupMessages({
  library(shiny)
  library(dplyr)
  library(ggplot2)
  library(tibble)
})
```

### 6.6.2 Afledte hjælpefunktioner til scenarieinput

Denne kode samler de nødvendige hjælpefunktioner, som oversætter brugerens scenarieinput til strukturerede modelvariable. Her udvælges først den bedst præsterende model fra LONG-evalueringen, hvorefter måned, sæson, weekend og rundefase omsættes til konsistente, numeriske og kategoriske størrelser. Formålet er at sikre, at input fra Shiny-appen kan kobles direkte til den historiske struktur, som modellen er estimeret på, uden at introducere ekstra antagelser eller kompleksitet.

```
# shiny

artifact_long <- res_full$artifact
best_model_name <- artifact_long$leaderboard |> arrange(RMSE) |> slice(1) |> pull(model) |> as.character

month_labels_da <- c("Januar","Februar","Marts","April","Maj","Juni","Juli","August","September","Oktober")
```

```

month_map <- setNames(1:12, month_labels_da)

season_start_from_year_month <- function(year, month_int) {
  year <- as.integer(year); month_int <- as.integer(month_int)
  if (is.na(year) || is.na(month_int)) return(NA_integer_)
  if (month_int >= 7) year else (year - 1L)
}

season_period_from_month <- function(m) {
  m <- as.integer(m)
  if (m %in% c(12,1,2)) return("Vinter")
  if (m %in% c(3,4,5)) return("Forår")
  if (m %in% c(6,7,8)) return("Sommer")
  if (m %in% c(9,10,11)) return("Efterår")
  "UKENDT"
}

weekend_from_ugedag <- function(ugedag) {
  if (is.na(ugedag)) return(NA_integer_)
  if (ugedag %in% c("Lørdag", "Søndag")) 1L else 0L
}

runde_from_fase <- function(fase) {
  if (is.na(fase)) return(NA_integer_)
  if (fase == "Tidlig") return(4L)
  if (fase == "Midt") return(14L)
  if (fase == "Sen") return(26L)
  NA_integer_
}

```

### 6.6.3 Historiske referenceværdier og fallback-logik

Denne kode opstiller de historiske referenceværdier, som bruges til at fastlægge baseline-niveauet i scenarieprognosen. Først konstrueres opslagstabeller for sæsongennemsnit og runde-specifikke gennemsnit baseret på den eksisterende pipeline. Herefter defineres en fallback-mekanisme, som sikrer et realistisk niveau, hvis der mangler historik for den valgte sæson eller runde. De to hjælpefunktioner returnerer dermed enten et direkte historisk estimat for den relevante sæson og runde eller, hvis det ikke findes, det bedst mulige historiske alternativ. Formålet er at gøre prognosen robust over for manglende data uden at introducere nye antagelser.

```

# shiny

# -----
# Historik-tabeller fra pipeline:
# -----

season_hist_map <- season_mean |>
  filter(!is.na(sæson_startår), is.finite(as.numeric(hist_mean))) |>
  distinct(sæson_startår, .keep_all = TRUE) |>
  arrange(sæson_startår) |>
  transmute(sæson_startår = as.integer(sæson_startår),
            hist_mean = as.numeric(hist_mean))

```

```

round_hist_map <- round_mean |>
  filter(!is.na(sæson_startår), !is.na(runde), is.finite(as.numeric(hist_round_mean))) |>
  distinct(runde, sæson_startår, .keep_all = TRUE) |>
  transmute(
    runde = as.integer(runde),
    sæson_startår = as.integer(sæson_startår),
    hist_round_mean = as.numeric(hist_round_mean)
  )

fallback_level <- {
  v <- suppressWarnings(as.numeric(analysis_df_long_full$tilskuere))
  v <- v[is.finite(v)]
  if (length(v) == 0) 6500 else mean(v)
}

get_hist_mean_for_target <- function(season_start_year_target) {
  if (is.na(season_start_year_target)) return(NA_real_)
  hit <- season_hist_map |> filter(sæson_startår == season_start_year_target) |> slice(1)
  if (nrow(hit) == 1) return(as.numeric(hit$hist_mean))
  if (nrow(season_hist_map) > 0) return(as.numeric(tail(season_hist_map$hist_mean, 1)))
  NA_real_
}

get_hist_round_mean_for_target <- function(season_start_year_target, runde_target) {
  if (is.na(season_start_year_target) || is.na(runde_target)) return(NA_real_)
  hit <- round_hist_map |>
    filter(sæson_startår == season_start_year_target, runde == runde_target) |>
    slice(1)
  if (nrow(hit) == 1) return(as.numeric(hit$hist_round_mean))

  hit2 <- round_hist_map |>
    filter(runde == runde_target) |>
    arrange(sæson_startår) |>
    slice_tail(n = 1)
  if (nrow(hit2) == 1) return(as.numeric(hit2$hist_round_mean))

  NA_real_
}

```

#### 6.6.4 Datadrevet måned- og ugedagskorrektion samt scenarie-app

Denne del udvider LONG-modellen med en enkel, datadrevet korrektion for måned og ugedag baseret udelukkende på træningsdata. Effekterne beregnes som afvigelser fra det samlede gennemsnit og anvendes i dæmpet form for at undgå overtilpasning. Prognosen tager derfor udgangspunkt i et historisk baseline-niveau fra sæson og rundefase og justeres derefter moderat for systematiske kalendereffekter.

Koden samles i en Shiny-app, som operationaliserer modellen i et scenarieformat. Brugeren kan indtaste få, centrale antagelser om en fremtidig kamp og få et forventet tilskuertal, et kapacitetsforhold og en visuel sammenligning mod stadionkapaciteten. Appen er ikke tænkt som en fuld ML-implementering, men som et transparent og anvendeligt værktøj, der omsætter den estimerede model til konkret planlægningsstøtte.

```
# shiny
```

```
# =====  
# Datadrevet korrektion for måned + ugedag (baseret på TRAIN-data)  
#     Vi bruger samme split som din LONG model (artifact split years)  
# =====
```

```
test_years_long <- artifact_long$split_info$test_years  
train_df_for_effects <- analysis_df_long_full |>  
  filter(!(år %in% test_years_long)) |>  
  mutate(  
    måned = as.integer(måned),  
    ugedag = factor(as.character(ugedag), levels = c("Mandag", "Tirsdag", "Onsdag", "Torsdag", "Fredag", "Lørdag", "Søndag")),  
    tilskuere = as.numeric(tilskuere)  
  ) |>  
  filter(is.finite(tilskuere), is.finite(måned), !is.na(ugedag))
```

```
overall_train_mean <- mean(train_df_for_effects$tilskuere, na.rm = TRUE)
```

```
month_effect_tbl <- train_df_for_effects |>  
  group_by(måned) |>  
  summarise(month_mean = mean(tilskuere, na.rm = TRUE), .groups = "drop") |>  
  mutate(month_adj = month_mean - overall_train_mean)
```

```
wday_effect_tbl <- train_df_for_effects |>  
  group_by(ugedag) |>  
  summarise(wday_mean = mean(tilskuere, na.rm = TRUE), .groups = "drop") |>  
  mutate(wday_adj = wday_mean - overall_train_mean)
```

```
get_month_adj <- function(month_int) {  
  hit <- month_effect_tbl |> filter(måned == as.integer(month_int)) |> slice(1)  
  if (nrow(hit) == 1) return(as.numeric(hit$month_adj))  
  0  
}  
get_wday_adj <- function(wday_chr) {  
  hit <- wday_effect_tbl |> filter(as.character(ugedag) == as.character(wday_chr)) |> slice(1)  
  if (nrow(hit) == 1) return(as.numeric(hit$wday_adj))  
  0  
}
```

```
# Dæmpning: vi vil have effekt, men ikke "overfit"
```

```
W_MONTH <- 0.35
```

```
W_WDAY <- 0.25
```

```
# Prediction-wrapper:
```

```
# - Hvis best er BASE_hist_mean, bruger vi: baseline (runde-justeret) + dæmpet måned/ugedag
```

```
predict_app_long <- function(new_df, artifact) {  
  best <- artifact$leaderboard |> arrange(RMSE) |> slice(1)  
  model_name <- as.character(best$model[[1]])
```

```
  # Baseline-niveau (runde-justeret hvis muligt)
```

```

base_level <- ifelse(is.finite(as.numeric(new_df$hist_round_mean)),
                    as.numeric(new_df$hist_round_mean),
                    as.numeric(new_df$hist_mean))

# Måned/ugedag-korrektion (dæmpet)
month_adj <- as.numeric(new_df$month_adj)
wday_adj <- as.numeric(new_df$wday_adj)
corr <- (W_MONTH * month_adj) + (W_WDAY * wday_adj)

if (model_name == "BASE_hist_mean") {
  pred <- base_level + corr
  return(tibble(status = "OK", model = "BASE_hist_mean (runde + måned/ugedag-korr.)", prediction = p
})

# Hvis en "rigtig" model vinder, kan du vælge at bruge den her.
# Lige nu holder vi os til baseline-logikken for forklaringskraft:
pred <- base_level + corr
tibble(status = "OK", model = paste0(model_name, " (app: + måned/ugedag-korr.)", prediction = pred)
}

ui <- fluidPage(
  titlePanel(paste0("LONG - Scenarie-prognose (bedste model: ", best_model_name, ")")),
  sidebarLayout(
    sidebarPanel(
      tags$h4("Indtast scenarie (6 inputs)"),

      numericInput("in_year", "År (fx 2026)", value = year(Sys.Date()), min = 2000, max = 2100, step =

      selectInput("in_month", "Måned", choices = month_labels_da, selected = month_labels_da[month(Sys

      selectInput("in_ugedag", "Ugedag", choices = c("Mandag","Tirsdag","Onsdag","Torsdag","Fredag","L
        selected = "Mandag"),

      selectInput("in_runde_fase", "Runde-fase", choices = c("Tidlig","Midt","Sen"), selected = "Tidli

      selectInput("in_hellig", "Helligdag?", choices = c("Nej","Ja"), selected = "Nej"),

      numericInput("in_capacity", "Stadionkapacitet", value = 10000, min = 1000, max = 50000, step = 5

      actionButton("btn_pred", "Predict")
    ),

    mainPanel(
      tags$h3("Output"),
      uiOutput("txt_out_html"),
      plotOutput("plt_cap", height = 320),
      tags$hr(),
      tags$h4("Kort forklaring"),
      tags$ul(
        tags$li("Baseline-niveau kommer fra historik: hist_mean (sæson) og helst hist_round_mean (rund
        tags$li("Måned og ugedag korrigerer niveauet med historiske afvigelser (dæmpet)."),

```

```

    tags$li("Det er ikke en fuld ML-model - men en forklarlig scenarie-korrektion, der gør app'en
  )
)
)
)

server <- function(input, output, session) {

  scenarie_df <- eventReactive(input$btn_pred, {

    year_in <- as.integer(input$in_year)
    month_int <- as.integer(month_map[[input$in_month]])
    ugedag_in <- as.character(input$in_ugedag)
    runde_fase_in <- as.character(input$in_runde_fase)
    hellig_in <- ifelse(input$in_hellig == "Ja", 1L, 0L)
    cap <- as.numeric(input$in_capacity)

    weekend_in <- weekend_from_ugedag(ugedag_in)
    sæson_periode_in <- season_period_from_month(month_int)

    ssy <- season_start_from_year_month(year_in, month_int)

    hist_mean_in <- get_hist_mean_for_target(ssy)
    if (!is.finite(hist_mean_in)) hist_mean_in <- as.numeric(fallback_level)

    runde_est <- runde_from_fase(runde_fase_in)
    hist_round_in <- get_hist_round_mean_for_target(ssy, runde_est)

    # NYT: hent justeringer
    month_adj_in <- get_month_adj(month_int)
    wday_adj_in <- get_wday_adj(ugedag_in)

    tibble(
      år = year_in,
      måned = month_int,
      ugedag = factor(ugedag_in, levels = c("Mandag", "Tirsdag", "Onsdag", "Torsdag", "Fredag", "Lørdag", "Søndag"),
      weekend = as.integer(weekend_in),
      er_helligdag = as.integer(hellig_in),
      runde_fase = factor(runde_fase_in, levels = c("Tidlig", "Midt", "Sen")),
      sæson_periode = factor(sæson_periode_in, levels = c("Vinter", "Forår", "Sommer", "Efterår", "UKENDT")),

      hist_mean = as.numeric(hist_mean_in),
      hist_round_mean = as.numeric(hist_round_in),

      month_adj = as.numeric(month_adj_in),
      wday_adj = as.numeric(wday_adj_in),

      runde_estimat = as.integer(runde_est),
      kapacitet = cap
    )
  })
}

```

```

pred_tbl <- eventReactive(input$btn_pred, {
  df <- scenarie_df()
  predict_app_long(df, artifact_long)
})

output$txt_out_html <- renderUI({
  pr <- pred_tbl()
  df <- scenarie_df()

  pred <- as.numeric(pr$prediction[[1]])
  cap <- as.numeric(df$kapacitet[[1]])
  fylldt <- (pred / cap) * 100

  # Farve-regel (som du bad om)
  col_pred <- ifelse(is.finite(pred) && pred >= 5000, "darkgreen", "orange")

  # Tekst: vis også "hvor kommer niveauet fra"
  base_level <- ifelse(is.finite(as.numeric(df$hist_round_mean[[1]])),
    as.numeric(df$hist_round_mean[[1]]),
    as.numeric(df$hist_mean[[1]]))

  corr <- (W_MONTH * as.numeric(df$month_adj[[1]])) + (W_WDAY * as.numeric(df$wday_adj[[1]]))

  tagList(
    tags$p(tags$b("Model: "), pr$model[[1]]),
    tags$p(
      tags$b("Forudsagt tilskuertal: "),
      tags$span(style = paste0("color:", col_pred, "; font-weight:700;"),
        ifelse(is.finite(pred), format(round(pred, 0), big.mark=".", decimal.mark=","), "NA")
      ),
    tags$p(
      tags$b("Stadion fylldt: "),
      tags$span(style = "font-weight:700;",
        ifelse(is.finite(fylldt),
          paste0(format(round(fylldt, 1), decimal.mark=","), " %"),
          "NA")),
      paste0(" (kapacitet = ", format(cap, big.mark=".", decimal.mark=","), ")")
    ),
    tags$hr(),
    tags$p(tags$b("Debug (forklaring af tallet:)")),
    tags$ul(
      tags$li(paste0("Baseline-niveau (hist): ",
        format(round(base_level,0), big.mark=".", decimal.mark=","),
        ifelse(is.finite(df$hist_round_mean[[1]]), " (runde-justeret)", " (sæson-mean)"),
      tags$li(paste0("Måned-justering (dæmpet): ",
        format(round(W_MONTH * as.numeric(df$month_adj[[1]]),0), big.mark=".", decimal.
      tags$li(paste0("Ugedag-justering (dæmpet): ",
        format(round(W_WDAY * as.numeric(df$wday_adj[[1]]),0), big.mark=".", decimal.ma
      tags$li(paste0("Samlet korrektion: ",
        format(round(corr,0), big.mark=".", decimal.mark=",")))
    )

```

```

    )
  })

output$plt_cap <- renderPlot({
  pr <- pred_tbl()
  df <- scenarie_df()

  pred <- as.numeric(pr$prediction[[1]])
  cap <- as.numeric(df$kapacitet[[1]])

  if (!is.finite(pred)) {
    ggplot() +
      labs(title = "Forudsagt tilskuertal vs stadionkapacitet", subtitle = "Ingen prediction (NA)")
    theme_minimal(base_size = 14)
    return(invisible())
  }

  # Farve-regel til søjlen
  fill_group <- ifelse(pred >= 5000, " 5.000", "< 5.000")
  dfp <- tibble(label = "Forudsagt", value = pred, grp = fill_group)

  ggplot(dfp, aes(x = label, y = value, fill = grp)) +
    geom_col() +
    geom_hline(yintercept = cap, linetype = 2, linewidth = 0.8) +
    scale_fill_manual(values = c(" 5.000" = "darkgreen", "< 5.000" = "orange"), guide = "none") +
    labs(
      title = "Forudsagt tilskuertal vs stadionkapacitet",
      subtitle = paste0("Kapacitet (stiplet): ", format(cap, big.mark=".", decimal.mark=",")),
      x = NULL, y = "Tilskuere (forudsagt)"
    ) +
    theme_minimal(base_size = 14)
  })
}

cat("\n Starter Shiny app...\n")
shinyApp(ui = ui, server = server)

```

## 6.7 En organisatorisk beslutning baseret på scenarie-prognosen

Palle står med en konkret beslutning. Om to måneder venter en vigtig Superliga-kamp, og der er internt blevet rejst et forslag om at opsætte midlertidige faciliteter omkring stadion – eksempelvis ekstra salgsboder og bemanding – for at imødekomme et forventet højere pres på kampdagen.

Inden beslutningen træffes, anvender Palle Scenarie-prognose-appen til at vurdere situationen. For en søndagskamp i april 2026, sent i runde-fasen og uden helligdag, estimerer modellen et tilskuertal på cirka 6.134, svarende til en belægningsgrad på omkring 61 % af stadionkapaciteten.

# LONG — Scenarie-prognose (bedste model: BASE\_hist\_mean)

Indtast scenarie (6 inputs)

År (fx 2026)

2026

Måned

April

Ugedag

Søndag

Runde-fase

Sen

Helligdag?

Nej

Stadionkapacitet

10000

Predict

## Output

**Model:** BASE\_hist\_mean (runde + måned/ugedag-korr.)

**Forudsagt tilskuertal:** **6.134**

**Stadion fyldt:** 61,3 % (kapacitet = 10.000)

### Debug (forklaring af tallet):

- Baseline-niveau (hist): 5.966 (runde-justeret)
- Måned-justering (dæmpet): 175
- Ugedag-justering (dæmpet): -7
- Samlet korrektion: 168

Forudsagt tilskuertal vs stadionkapacitet

Kapacitet (stiplet): 10.000



Prognosen viser, at kampen forventes at ligge tæt på et historisk gennemsnitsniveau, og at de samlede korrektioner fra måned og ugedag kun giver et moderat løft. Det indikerer, at der ikke er tale om et scenarie med markant overbelægning eller ekstraordinært pres på faciliteterne.

På den baggrund vurderer Palle, at det ikke er økonomisk hensigtsmæssigt at investere i større organisatoriske tiltag allerede nu. I stedet besluttet det at afvente og først genoverveje initiativet, hvis nye informationer – eksempelvis sportslige resultater eller billetsalg tættere på kampdatoen – ændrer forudsætningerne.