

Opgave 1 – Webscrape

Opgave 1.1 – Hente data fra Bilbasen

I skal hente udvalgte biler fra bilbasen.dk. Hent så mange som muligt i en "lukket" serie. I kan f.eks vælge VW, model Up, eldrevne. Koordiner med de andre grupper så I henter forskellige biler. Når I henter data, skal I ud over de oplagte data også huske at gemme linket så man kan hente flere data på hver bil. I skal også hente forhandleren – både hans id samt firmanavn, adresse samt cvrnummer. Det kan involvere manuelt arbejde.

Formålet med opgaven var at scrape data fra Bilbasen.dk i en lukket serie med fokus på Mercedes EQA-klassen. Derudover skulle der indsamlies oplysninger om forhandlerne, herunder forhandler-ID, navn, adresse og CVR-nummer. Opgaven krævede en kombination af automatiseret databehandling og manuel intervention for at sikre fuldstændighed og præcision.

For atindhente data fra Bilbasen.dk blev der anvendt R's HTTR-pakke til at sende HTTP-anmodninger. GET-funktionen blev konfigurereret til at inkludere cookie og user-agent, hvilket simulerer en legitim bruger og omgår serverrestriktioner:

```
response <- GET(startlink,
  add_headers(
    cookie = 'INSERT COOKIE',
    `user-agent` = "Mozilla"
  )
)
```

Ved hjælp af browserens udviklerværktøj blev relevante CSS-selectors identificeret for de ønskede parametre, herunder bilens pris, modeltype og egenskaber. Eksempelvis blev prisen udtrukket fra CSS-klassen ".Listing_price__6B3kE", mens modeltype blev fundet under ".Listing_makeModel__7yqgs".

CSS-selectors anvendt til webscraping lyder som følger:

```
pricetag <- ".Listing_price__6B3kE"
propertiestag <- ".Listing_properties__ptWv"
modeltag <- ".Listing_makeModel__7yqgs"
detail_itemstag <- ".ListingDetails_list__WPBUe"
descriptiontag <- ".Listing_description__scNNM"
locationtag <- ".Listing_location__nKGQz"
```

Data blev herefter struktureret i en dataframe kaldet carframe med følgende kolonner:

Price	Details	Makemodel	Properties	Description	Location	Link	CarID	Scrapdate
No data available in table								

For at sikre komplet indsamling blev et loop implementeret, der itererede gennem alle sider for Mercedes EQA-klassen. HTML-indholdet blev parsed og data ekstraheret fra de relevante HTML-elementer. De udtrukne data blev samlet i en dataframe kaldet scraped_data og tilføjet til carframe ved hjælp af rbind.

Dette ses af nedenstående kode:

```
scraped_data <- data.frame(  
  Price = price,  
  Details = detail_items,  
  Makemodel = model,  
  Properties = property,  
  Description = description,  
  Location = location,  
  Link = link,  
  CarID = carid,  
  Scrapedate = Sys.time()  
)  
  
carframe <- rbind(carframe, scraped_data)
```

Forhandleroplysninger blev struktureret i en separat dataframe, dealerframe, med følgende kolonner:

	CarID	DealerCVR	DealerAddress	DealerID
No data available in table				

For at indsamle disse oplysninger blev et loop oprettet, der gennemgik bilernes individuelle sider, hvorefter forhandlerdata blev ekstraheret ved hjælp af CSS-selectors. Eksempelvis blev CVR-numre udtrukket fra klassen ".bas-MuiSellerInfoComponent-cvr", mens adresser blev fundet under ".bas-MuiSellerInfoComponent-address". Forhandler-ID blev udledt fra mønsteret "idxxx" i bilens URL.

Private annoncer, hvor CVR-numre og andre oplysninger manglede, blev angivet som NA.

Kodning til oprettelse af dealerframe ser således ud:

```
dealerframe <- data.frame(  
  CarID = character(),  
  DealerCVR = character(),  
  DealerAddress = character(),  
  DealerID = character()  
)
```

	CarID	DealerCVR	DealerAddress	DealerID
1	6311333	29399190	Bavnehøjvej 21, 8600 Silkeborg	14540
2	6349172	26172403	Bondovej 13-15, 5250 Odense SV	1989
3	6326546	42664707	Sakskærvej 100, 6800 Varde	20446
4	6310953	25905504	Herredsvej 30, 7100 Vejle	5973
5	6296481	48118712	Viborgvej 5-11, 8600 Silkeborg	2638

På grund af manglende oplysninger i scraping-processen blev firmanavne manuelt koblet til CVR-numre ved at oprette en tabel med sammenhørende værdier. Ved kendskab til CVR-numre var forhandlernavnene tilgængelige på CVR-registret (virk.dk).

Det samlede datasæt carframe indeholder følgende oplysninger:

- Pris, detaljer, modeltype, egenskaber, beskrivelse, lokation, link, bil-ID og scrapedato.

Forhandleroplysninger blev gemt i dealerframe, som inkluderer CVR-numre, adresser og forhandler-ID'er. Firmanavne blev manuelt tilføjet for at supplere de manglende data. De to datasæt blev derefter kombineret for at opnå en fuldstændig oversigt.

	Price	Details	Makemodel	Properties	Description	Location	Link	CarID	Scrapedate	DealerCVR	DealerAddress	DealerID
1	379.900 kr.	9/20229.000 km416 km rækkevidde	Mercedes EQA300AMatic Sd	8/20229.000 km...	AMG Line Aut., 18" alufælge, foldout, tilma. 360° kamera, ad...	Silkeborg, Østjylland	https://www.bilbasen.dk/brugt/bil/mercedes/eqa300/4matic...	6311333	2024-11-24 16:03:38	29199190	Børnehøjvej 21, 8600 Silkeborg	14540
2	254.900 kr.	7/202145.11 km424 km rækkevidde	Mercedes EQA250Progressive Sd	7/202145.11 km...	2023 Mercedes-Benz EQA 250 til salg! Denne el-SUV har kar...	Odense SV, Fyn	https://www.bilbasen.dk/brugt/bil/mercedes/eqa250/progr...	6349712	2024-11-24 16:03:38	26172403	Bondvæj 13-15, 5250 Odense SV	1869
3	339.900 kr.	6/202232.000 km520 km rækkevidde	Mercedes EQA250+AMG Edition Sd	6/202232.000 km...	Svingbart Treck v AMG Line v 250+ Version - Long Rang...	Vejle, Syd- og Sønderjylland	https://www.bilbasen.dk/brugt/bil/mercedes/eqa250/amg-e...	6329546	2024-11-24 16:03:38	42664707	Sæskærvej 100, 6800 Vejle	20446
4	379.900 kr.	11/20232.000 km530 km rækkevidde	Mercedes EQA250+AMG Line Sd	11/20232.000 km...	Komfort/Teknik: Premiumpakk. Fahrerassistenzpakk. Plus, A...	Vejle, Syd- og Sønderjylland	https://www.bilbasen.dk/brugt/bil/mercedes/eqa250/amg-l...	6310093	2024-11-24 16:03:38	25005504	Herrerdovsvej 30, 7100 Vejle	5973
5	339.900 kr.	7/20229.000 km419 km rækkevidde	Mercedes EQA250AMS Line Sd	7/20229.000 km...	BEMÆRK: v AMG Sportpakk. + Sælgbar anhængertræk +	Silkeborg, Østjylland	https://www.bilbasen.dk/brugt/bil/mercedes/eqa250/amg-l...	6295681	2024-11-24 16:03:38	48118712	Viborgvej 5-11, 8600 Silkeborg	2638

Konklusion:

Ved hjælp af R's HTTR-pakke og browserens udviklerværktøj blev data fra Bilbasen.dk succesfuldt scraped og struktureret i et detaljeret datasæt. CSS-selectors gjorde det muligt at udtrække specifikke bil- og forhandleroplysninger. Kombineret med manuel behandling blev det sikret, at datasættene indeholder omfattende og præcise oplysninger, som kan anvendes til videre analyse.

Opgave 1.2 – Rense data

Salgsteksten fra bilen indeholder en masse overflødige tegn. Sørg for at få renset teksten så der kun er almindelige karakterer og tegn (punktum, komma) tilbage. Sørg også for at "newline" erstattes af ". " og at mange mellemrum erstattes med ét mellemrum.

Formålet med denne opgave er at rense datasættet og sikre, at salgsteksterne for biler kun indeholder relevante karakterer og tegn. Dette indebærer fjerneelse af overflødige tegn, formateringsfejl samt konsolidering af mellemrum for at opnå en mere struktureret og læsbar tekstbeskrivelse. Yderligere blev datasættet behandlet for at oprette nye variabler på følgende vis:

1. Rensning af salgstekster Salgsteksterne, der findes under kolonnen Description, blev behandlet for at fjerne overflødige tegn og formateringsfejl:

- Ny linje (\n) blev erstattet med et punktum og mellemrum (".". ")
- Tabulatorer (\t) blev erstattet med ét mellemrum.
- Emojis og ikke-alfanumeriske tegn blev fjernet, undtagen tegn som punktum og komma.
- Flere mellemrum blev reduceret til ét enkelt mellemrum.
- Ledende og afsluttende mellemrum blev fjernet med funktionen trimws().

Eksempel på transformation:

- Original: "Mercedes\nBenz\tEQA250 🚗🚀"
- Renset: "Mercedes. Benz EQA250"

2. Oprettelse af nye variabler ud fra kolonnen Makemodel:

- Brand: De første otte tegn i Makemodel blev isoleret for at oprette en ny kolonne, der angiver bilmærket.
- Doors: Antallet af døre blev udtrukket som en ny kolonne, hvor kun gyldige værdier som "4d" og "5d" blev inkluderet.

- Modeltype: Modellenavnet blev isoleret, og ved hjælp af grep() blev biler af typen "EQA250" blev inkluderet i datasættet.

4. Behandling af andre variabler såsom pris (Price), kilometerstand (Km), og rækkevidde (Range) blev behandlet og renset for overflødige formater som valuta-symboler og tusindtalsseparator, og de blev konverteret til numeriske værdier.

Nedenfor ses et udsnit af koden, der blev brugt til at rense teksten og oprette nye variabler:

```
Mercedes$Description <- Mercedes$Description %>%
  gsub("\n", ". ", .) %>% # Erstat newline-tegn med ". "
  gsub("\t", " ", .) %>% # Erstat tabulatorer med mellemrum
  gsub("[^\x01-\x7F]", "", .) %>% # Fjern emojis og ikke-
alfanumeriske tegn, kun tillad punktum, komma og mellemrum
  gsub("\\s+", " ", .) %>% # Erstat flere mellemrum med ét mellemrum
  trimws() # Fjern eventuelle ledende og afsluttende mellemrum

Mercedes$Brand <- gsub("^(.{1,8}).*", "\\\1", Mercedes$Makemodel)

Mercedes$Doors <- gsub(".*(\\d{1}d)$", "\\\1", Mercedes$Makemodel)
Mercedes$Doors <- ifelse(Mercedes$Doors %in% c("4d", "5d"),
Mercedes$Doors, NA)

Mercedes$Modeltype <- gsub("^.{1,8}\\s(.*)\\s\\d{1}d$", "\\\1",
Mercedes$Makemodel)
Mercedes <- Mercedes[grep("EQA250", Mercedes$Modeltype), ]
```

Konklusion

Rensningen af datasættet har bidraget til en betydelig forbedring af datakvaliteten. Overflødige tegn og formateringsfejl blev fjernet, hvilket resulterede i en mere ensartet og brugervenlig præsentation af data. De oprettede variabler og filtrering af irrelevante observationer bidrager til den senere analyse i opgaven.

Opgave 1.3 – Hente nye data – simuleret

I skal nu lave en simuleret hentning af biler, hvor I skal tage udgangspunkt i de biler I hentede i 1.1. Den simulerede kørsel skal have en scrapedate som ligger én dag senere end 1.1 og den skal have 2 rækker med nye biler, 3 rækker med ændrede priser og så skal der mangle 5 rækker fra den oprindelige testkørsel så I kan simulere, at bilerne er blevet solgt.

Der udføres en simuleret hentning af biler med udgangspunkt i webscraping af bilbasen i opgave 1.1, hvor der er indhentet data på Mercedes EQA250 som der ved udførelsen af webscraping har været til salg på bilbasen. Her tages der udgangspunkt i dataframen navngivet "Mercedes_simulation".

Formålet med simuleringen er, at der foretages ændringer og tilføjelser som afspejler en opdatering af dataene, som kunne opstå i en real-life situation. Simuleringen omfatter følgende elementer:

- **Ændring af scrapedate:** simuleringen skal reflektere en opdatering af datene én dag senere.
- **Tilføjelse af nye biler:** to nye biler tilføjes til datasættet.
- **Ændring af priser:** tre biler får reguleret sine priser.
- **Fjernelse af solgte biler:** fem biler fjernes for at simulere, at de er blevet solgt.

Tilgangen skal forsøge at skabe et virkelighedsscenarie af, hvordan et datasæt kan bevæge sig.

Ændring af scrapedate

I det oprindelige datasæt fra opgave 1.1 er hver bil blevet opdateret med en "scrapedate", som angiver tidspunktet for, hvornår dataene blev hentet. For at simulere, at der er sket en opdatering af dataene én dag senere, blev værdierne i kolonnen Scrapedate justeret ved at tilføje én dag. Dette blev udført med følgende ved brug af nedenstående R Kode:

```
Mercedes_simulation$Scrapedate <- Mercedes_simulation$Scrapedate +  
as.difftime(1, units = "days")
```

Denne justering simulerer, at der er sket en opdatering af dataene én dag senere end den oprindelige dag for webscraping.

Tilføjelse af nye biler

Som en del af simuleringen tilføjes to nye biler til datasættet. De nye biler bliver oprettet ved at generere tilfældige værdier for bilens egenskaber, herunder mærke, model, dørene, årgang, km-stand, rækkevidde, pris, samt forhandlerens detaljer. Som illustreret nedenstående R kode anvendes primært sample() til at udføre denne handling, hvorefter de generet biler tilføjes til dataframen ved brug af rbind():

```
nye_biler <- data.frame(  
  CarID = max(Mercedes_simulation$CarID) + 1:2,  
  Brand = sample(Mercedes_simulation$Brand, 2, replace = TRUE),  
  Modeltype = sample(Mercedes_simulation$Modeltype, 2, replace =  
TRUE),  
  Doors = sample(Mercedes_simulation$Doors, 2, replace = TRUE),  
  Year = sample(Mercedes_simulation$Year, 2, replace = TRUE),  
  Km = sample(Mercedes_simulation$Km, 2, replace = TRUE),  
  Range = sample(Mercedes_simulation$Range, 2, replace = TRUE),  
  Price = sample(Mercedes_simulation$Price, 2, replace = TRUE),  
  Description = "Jeg er ikke rigtig hehe",  
  Location = sample(Mercedes_simulation$Location, 2, replace =  
TRUE),  
  DealerName =  
sample(Mercedes_simulation$DealerName[!is.na(Mercedes_simulation$Dea  
lerName)], 2, replace = TRUE),  
  DealerCVR =  
sample(Mercedes_simulation$DealerCVR[!is.na(Mercedes_simulation$Deal  
erCVR)], 2, replace = TRUE),  
  DealerID = sample(Mercedes_simulation$DealerID, 2, replace =  
TRUE),  
  DealerAddress =  
sample(Mercedes_simulation$DealerAddress[!is.na(Mercedes_simulation$  
DealerAddress)], 2, replace = TRUE),  
  Link = "https://www.bilbasen.dk/Jeg/er/en/simuleret/bil",  
  Scrapedate = as.POSIXct(sample(Mercedes_simulation$Scrapedate, 2,  
replace = TRUE))  
)
```

Ændring af priser

Ved simulering af prisændring af tre tilfældige biler anvendes sample() endnu engang, hvorefter at der med runif() simuleres et tilfældigt prisfald på mellem 10 % og 20 %. Bilernes ID (CarID) gemmes før prisændringen, så disse nemt kan udpeges efter behov. R koden ser ud som følger:

```

set.seed(123) # For reproducibility

price_changes <- sample(1:nrow(Mercedes_simulation), 3)

changed_prices <- Mercedes_simulation$CarID[price_changes]

Mercedes_simulation$Price[price_changes] <-
round(Mercedes_simulation$Price[price_changes] *
runif(length(price_changes), min = 0.8, max = 0.9))

```

Fjernelse af solgte biler

Afslutningsvis simuleres et bilsalg ved at der udpeges fem tilfældige biler, som derefter bliver markeret som værende solgt. Dette gøres i første omgang ved, at der oprettes en ny kolonne kaldet ”solgt”, som ved brug af boolean markere sandt/falsk ift. Om en bil er solgt eller ej. Der anvendes hertil `sample()` til at vælge fem tilfældige biler som herefter indlæses på ny i datasættet som værende solgt.

Konklusion

Den simulerede opdatering af bildataene giver et realistisk billede af, hvordan et datasæt kan ændre sig over tid. Ved at implementere ændringer som opdatering af `scrapedate`, tilføjelse af nye biler, prisændringer og fjernelse af solgte biler, har vi skabt et dynamisk datasæt, der bedre afspejler de realiteter, der findes i bilmarkedet.

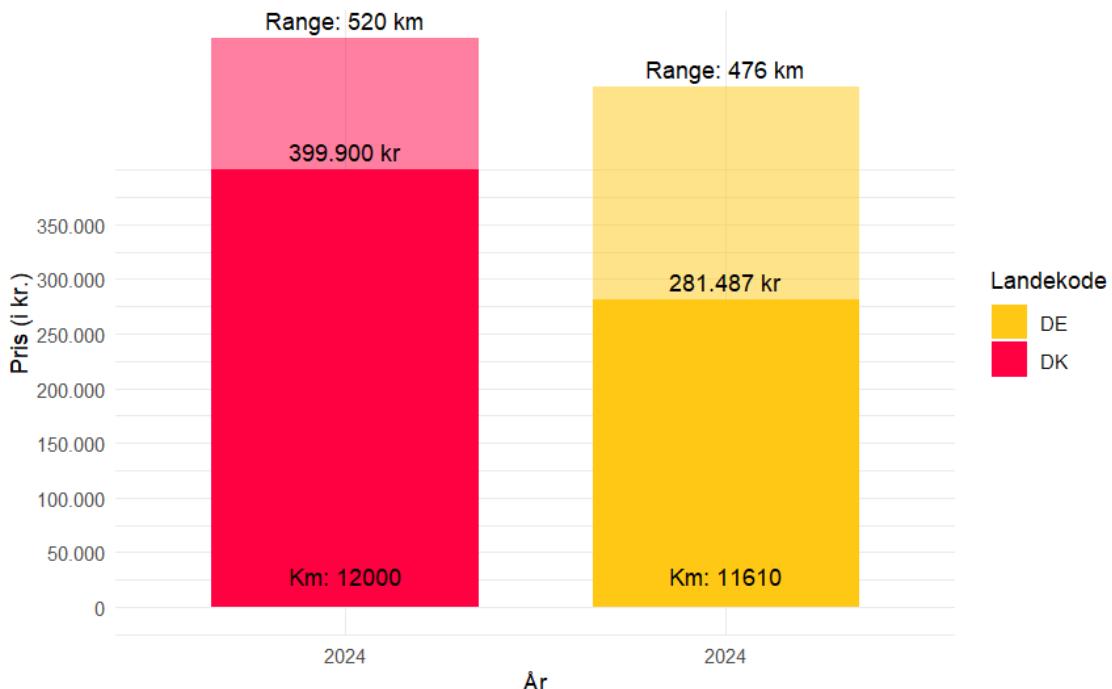
Opgave 1.4 – Hente tyske data

I skal nu hente samme bilsegment fra Tyskland, så man kan afgøre om det kan betale sig at køre til Tyskland og hente den. Jeg har kigget på 12gebrauchtwagen.de men det kan være der findes andre.

I opgave 1.4 skulle vi undersøge, om det kan betale sig at hente en bil fra samme bilsegment i Tyskland i stedet for at købe den i Danmark. Til dette formål har vi taget udgangspunkt i autoscout24.de, en af Europas største online platforme for køb og salg af nye og brugte biler. I opgaven har vi ligeført gjort brug af samme fremgangsmåde, som i tidligere opgaver, hvor vi har anvendt brugen af GET-metoden til at indhente data fra hjemmesiden. For at undgå begrænsninger under datascrapingen anvendte vi cookies og user-agent.

Ud fra siden har vi scrapet på baggrund af flere forskellige parameter, som vi vurderede, var essentielle for at konkludere om det kan betale sig at tage til Tyskland og købe en bil. Disse parametre inkluderede bl.a. ”mærke”, ”modeltype”, ”døre”, ”år”, ”km”, ”rækkevidde”, ”pris”, ”forhandleradresse”, ”forhandlernavn” og ”bil-link”. Efter scrapingen af siden gjorde vi endvidere brug af data-cleaning for senerehen at kunne merge vores to datasæt sammen for at kunne konkludere, om det økonomin kan betale sig at køre til Tyskland for at købe en bil. For at gøre de tyske priser sammenlignelige med de danske, oprettede vi en ny kolonne i datasættet. Denne kolonne omregner priserne fra euro til danske kroner, hvilket giver et mere retvisende grundlag for sammenligningen

Mercedes EQA250+AMG i Tyskland er billigere end Danmark!



Ud fra grafen er der blevet taget udgangspunkt i to tilfældige Mercedes EQA250+AMG fra Danmark og Tyskland, som begge er fra årgang 2024. Den overordnede pointe, som fremgår tydeligt af grafen, er, at bilen i Tyskland er væsentligt billigere end den tilsvarende bil i Danmark, på trods af små forskelle i rækkevidde og kilometerstand.

Hertil kan det aflæses, at den danske model har en pris på 399.900 kr, mens den tyske model kun koster 281.487 kr. Denne forskel på 118.413 kr viser, at bilen i Tyskland er betydeligt billigere, selv før eventuelle afgifter eller transportomkostninger tilføjes. Dette skyldes bl.a. også at der i Tyskland kun er en moms på 19%, som er lavere end den danske på 25%, hvilket bidrager til lavere salgspriser. Hertil tilbyder den tyske stat også et BAFA-tilskud på op til 9.000 euro (cirka 67.000 kr), som gives til købere af fabriksnye elbiler i Tyskland¹. Desuden er det tyske marked også mere konkurrencepræget end det danske, hvilket er med til at presse priserne ned. Ser man på afgiften for en Mercedes EQA250, ligger den mellem 0 og 20.000 kr., hvilket yderligere understreger, at det kan betale sig at tage til Tyskland for at købe bilen.

EQA 250

0 kr. – 20.000 kr.

Afgift på Mercedes EQA 250 - <https://quickimport.dk/hvad-er-afgiften-paa-mercedes-benz-eqa/>

Ser man på rækkevidden, kan det ud fra grafen aflæses, at den tyske model har en rækkevidde på 476 km, mens den danske model tilbyder lidt mere med 520 km. Denne forskel på 44 km i rækkevidde er en teknisk fordel for den danske bil, men den økonomiske besparelse på over 100.000 kr, hvilket gør det mindre

¹ <https://quickimport.dk/hvad-er-afgiften-paa-mercedes-benz-eqa/>

fordelagtigt at købe bilen i Danmark. Kilometerstanden for de to biler er næsten identisk, med 11.610 km i Tyskland kontra 12.000 km i Danmark, hvilket ikke spiller en væsentlig rolle.

Konklusion:

Det kan ud ovenstående konkluderes, at det kan betale sig at tage til Tyskland for at købe en Mercedes EQA250+AMG frem for i Danmark, da der er en prisforskel på 118.413 kr mellem den danske og tyske model på netop denne bil. Denne besparelse skyldes primært lavere moms i Tyskland, det tyske statstilskud og et mere konkurrencepræget marked i Tyskland kontra Danmark. Selv når man medregner den danske registreringsafgift, som for denne bilmodel ligger mellem 0 og 20.000 kr, er der stadig betydelige besparelser ved at hente bilen i Tyskland frem for at købe den i Danmark.

Opgave 2 – SQL

Opgave 2.1 – Oprette skemaet for bilbasen

For at kunne gemme bilerne i databasen er det vigtigt at oprette et skema, hvor man har invariante entiteter organiseret i tabeller med fremmed-nøgler så de kan linkes sammen, samt variende data i tidsserieagtige tabeller, hvor man linker til den sidst-forekommende observation. Jeres design skal beskrives i et ER-diagram og jeres DDL-statements skal ligge på github med fileextension .sql.

Se under opgave 2.2

Opgave 2.2 – Gemme bilerne i database

I skal nu gemme jeres første scrape-resultat i jeres database. I kan lade R-driveren gøre arbejdet med at oprette skemaet men I skal sørge for at der er plads til det, som R-driveren sætter på datatyperne. I skal desuden definere carid som primær nøgle. Jeres INSERT-statements skal også ligge på github.

I besvarelsen af opgaverne 2.1 og 2.2 er disse behandlet samlet, da de er udført i én sammenhængende proces. Som led i opbygningen af en database i MySQL Workbench blev der først oprettet et skema med navngivet *bilbasen*. Efterfølgende blev data indlæst fra RStudio ved hjælp af pakkerne **DBI** og **RMariaDB**. Datagrundlaget stammer fra vores webscraping-aktivitet i opgave 1.

I overensstemmelse med data science-modellen blev der foretaget datarensning og forberedelse af datasættet (jf. opgave 1.2). De rensede data blev gemt som en RDS-fil under navnet *Mercedes*. For at tilpasse databaseopbygningen i MySQL blev datasættet opdelt i tre tabeller: **car_history**, **car**, og **dealer**, som hver især har følgende struktur og funktionalitet:

- **car_history**: Indholder *CarID* samt variabedata såsom pris, scrapedato og status for solgt/ikke-solgt.
- **car**: Indholder stamdata (ikke-variabel data) samt referencer til *DealerID* og *CarID*.
- **dealer**: Indholder *DealerID* samt information om forhandler og lokation.

Følgende R-kode blev anvendt til at indlæse data fra RStudio til MySQL Workbench:

```
#1. Indlæs RDS fil med webscraping data
Mercedes <- readRDS(file = "Mercedes.rds")

#2. Opret tabeller til MySQL
car_history <- Mercedes[,c(1,8,16)]
car_history$Solgt <- FALSE
car <- Mercedes[,c(1:7,9,13)]
dealer <- Mercedes[,c(10:15)]
## Fjern duplikater da DealerID kun skal indlæses unikt i databasen
i MySQL
dealer <- dealer[!duplicated(dealer$DealerID), ]

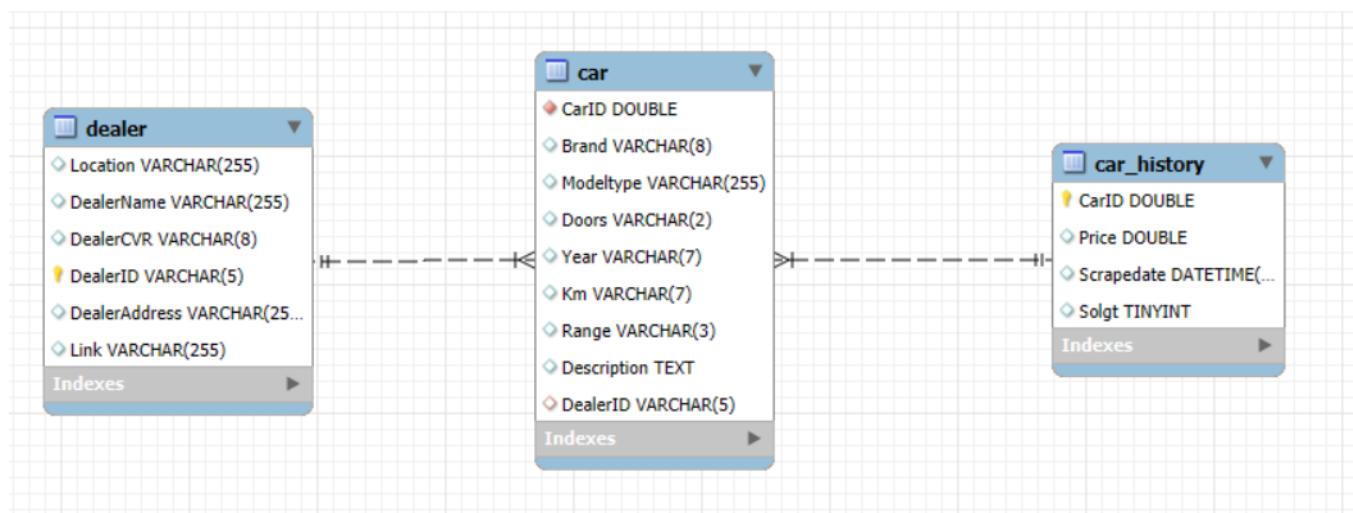
#3. Opret forbindelse til MySQL database
connection <- dbConnect(MariaDB(),
                        db = "bilbasen",
                        host = "localhost",
                        port = 3306,
                        user = "root",
                        password = "")
```

```
#4. Indlæs tabeller i MySQL databasen navngivet "bilbasen"
dbWriteTable(connection,"car_history",car_history)
dbWriteTable(connection,"car",car)
dbWriteTable(connection,"dealer",dealer)
```

Ved at indlæse dataframes direkte som tabeller i MySQL kunne vi undgå manuel oprettelse af tabeller og individuelle *INSERT*-statements i MySQL. Tabellerne blev forbundet ved hjælp af primær- og fremmednøgler:

- *CarID* fungerer som primær nøgle i **car_history** og er fremmednøgle i **car**.
- *DealerID* fungerer som primær nøgle i **dealer** og er fremmednøgle i **car**.

Denne relationelle struktur er illustreret i det nedenstående EER-diagram.



Konklusion: Skemaet er designet med invariant data organiseret i separate tabeller, hvor relationer mellem tabellerne sikres via primær- og fremmednøgler. Variabedata er opbevaret i tidsserie-strukturer, og designet er dokumenteret i et EER-diagram. DDL-statements er tilgængelige på Github i formatet .sql. Det første scrape-resultat blev gemt i databasen ved hjælp af R-driveren, der sikrede korrekt oprettelse af tabeller og datatyper. CarID/DealerID blev defineret som primær nøgle. Alle nødvendige *INSERT*-statements samt scripts til MySQL er uploadet på Github.

Opgave 2.3 – Opdatere databasen ud fra den simulerede kørsel

Hvis I har fået lavet et korrekt skema – altså at I kan versionere prisen vha en pris-tabel – burde I kunne opdatere databasen med jeres simulerede nye scraping, hvor I altså opretter/ændrer en record med pris, dato og carid så man joine de to tabeller på carid. Det gøres bedst ved at I laver et script eller en funktion som henter den forrige kørsel fra databasen (altså jeres data fra opgave 2.2) og sammenligner med den simulerede kørsel (altså kørsel-II). I skal finde a) Nye records b) Ændrede records (på prisen) c) Missing records (solgte biler) Og opdatere databasen efterfølgende. De solgte biler skal ikke slettes men markeres som TRUE i sold.

I denne opgave blev databasen opdateret baseret på en simuleret scraping-kørsel (opgave 1.3).

Fremgangsmåden følger en systematisk tilgang, der bygger videre på tidligere opgaver (2.1 og 2.2). Formålet

var at versionere opdateringer i data ved at identificere og håndtere nye, ændrede og manglende records, samt at markere solgte biler.

Den simulerede scraping blev gemt som en dataframe kaldet `Mercedes_simulation`. Denne dataframe blev opdelt i tre tabeller, i overensstemmelse med det tidligere oprettede databaseskema:

1. **`car_history_simulation`**: Indholder variabel data såsom pris, scrapedato og status (solgt/ikke solgt).
2. **`car_simulation`**: Indholder stamdata som mærke, model og andre faste oplysninger.
3. **`dealer_simulation`**: Indholder oplysninger om bilforhandlere og deres lokationer.

Tabellerne blev midlertidigt indlæst i MySQL ved hjælp af R-pakkerne DBI og RMariaDB. Koden for indlæsning og forberedelse af data ses nedenfor:

```
Mercedes_simulation <- readRDS("Mercedes_simulation.rds")  
  
# Tilføj simuleret data til database i MySQL for at versionere  
# opdatering/ændring i data.  
car_history_simulation <- Mercedes_simulation[,c(1,8,16:17)]  
car_simulation <- Mercedes_simulation[,c(1:7,9,13)]  
dealer_simulation <- Mercedes_simulation[,c(10:15)]  
  
dealer_simulation <-  
  dealer_simulation[!duplicated(dealer_simulation$DealerID), ]  
  
dbWriteTable(connection, "car_history_simulation", car_history_simulation)  
dbWriteTable(connection, "car_simulation", car_simulation)  
dbWriteTable(connection, "dealer_simulation", dealer_simulation)  
  
saveRDS(car_history, file = "car_history.rds")  
saveRDS(car, file = "car.rds")  
saveRDS(dealer, file = "dealer.rds")  
  
saveRDS(car_history_simulation, file = "car_history_simulation.rds")  
saveRDS(car_simulation, file = "car_simulation.rds")  
saveRDS(dealer_simulation, file = "dealer_simulation.rds")
```

Efter den midlertidige indlæsning blev data fra de simulerede tabeller integreret i de oprindelige tabeller ved brug af `INSERT INTO` med `ON DUPLICATE KEY UPDATE`. Denne fremgangsmåde sikrer, at:

- **Nye records** bliver tilføjet.
- **Ændrede records** bliver opdateret (f.eks. prisændringer).
- **Manglende records** (solgte biler) markeres som solgt (`Solgt = TRUE`) i databasen.

Eksempel på SQL-kommando til opdatering af tabellen `car_history`:

```
-- Opdatér tabel car_history med simuleret data  
INSERT INTO car_history (CarID, Price, Scrapedate, Solgt)  
SELECT CarID, Price, Scrapedate, Solgt  
FROM car_history_simulation  
ON DUPLICATE KEY UPDATE
```

```

Price = car_history_simulation.Price,
Scrapedate = car_history_simulation.Scrapedate,
Solgt = car_history_simulation.Solgt;

```

Efter opdateringen blev de midlertidige tabeller fjernet for at optimere databasen, og opdateringerne blev valideret ved at sammenligne data mellem tabellerne ved hjælp af JOIN-operationer. Testen viste, at opdateringerne var korrekt implementeret og kunne kobles til den oprindelige database. Se nedenfor:

```

56 •   SELECT
57      *
58  FROM
59      car_history
60  ORDER BY solgt DESC;

```

	CarID	Price	Scrapedate	Solgt
▶	6242779	394800	2024-11-25 15:03:54.921717	1
	6305396	318900	2024-11-25 15:03:55.077368	1
	6324306	299900	2024-11-25 15:03:47.431451	1
	6333539	289800	2024-11-25 15:03:58.478576	1
	6359958	326999	2024-11-25 15:03:38.904325	1
	5895865	289900	2024-11-25 15:04:12.879365	0
	5930520	329900	2024-11-25 15:04:14.527993	0
	5994974	319900	2024-11-25 15:04:14.521015	0
	6032354	289900	2024-11-25 15:03:38.826961	0

car_histov 2 ×

```

56 •   SELECT
57      car_history.price,
58      car.year
59  FROM
60      car_history
61      JOIN
62      car ON car_history.carid = car.carid
63  WHERE
64      SUBSTRING_INDEX(car.year, '/', - 1) = '2022';

```

	price	year
▶	289900	1/2022
	289900	1/2022
	359900	12/2022
	359900	12/2022
	319900	7/2022
	319900	7/2022
	364900	12/2022
	364900	12/2022

Result 7 ×

Konklusion: Med denne fremgangsmåde er databasen opdateret med nye og ændrede records samt korrekt markeret solgte biler. Ved at anvende en struktureret tilgang og systematisk versionering sikres datakvalitet og integritet i databasen. Dette skaber et robust fundament for fremtidige analyser og forespørgsler.

Opgave 2.4 – Scrape & SQL med Miljødata.

I skal kigge på <https://envs.au.dk/om-instituttet-1/faglige-omraader/luftforurening-udledninger-og-effekter/data-om-luftkvalitet/aktuelle-maalinger/tabeller> og hente links fra H.C.Andersens Boulevard, Anholt, Banegårdsgade i Århus og Risø. I skal lave et R-script som kan hente data fra de fire lokationer og gemme dem i fire tabeller. I skal derpå vente en dag og så hente data igen og opdatere tabellerne med nye data. I forbindelse med næste OLA skal scriptet kunne køre på en linux-server én gang i døgnet. Så jeres script skal helst kunne eksekveres i terminalen uafhængigt af Rstudio med målestationen som argument.

Vi havde til opgave at skulle lave et R-script, som kan hente data fra 4 lokationer, herunder H.C.Andersens Boulevard, Anholt, Banegårdsgade i Århus og Risø fra ENVS, som er institut for miljøvidenskab på Århus universitet. Det omhandler data om luftkvalitet, der skal gemmes i fire tabeller, hvor dataen igen skal hentes 24 timer efter.

Ud fra tabellerne bliver det opdelt i hhv. starttid og de forskellige gasser og partikler, der udledes. Ud fra gasserne er der: NO_2 : Kvælstofdioxid $\mu\text{g}/\text{m}^3$, NO_x : Kvælstofoxider $\mu\text{g}/\text{m}^3$, CO : Kulmonoxid mg/m^3 , O_3 : Ozon $\mu\text{g}/\text{m}^3$, SO_2 : Svooldioxid $\mu\text{g}/\text{m}^3$. Ud fra partikler er der: PM_{10} (Teom): $\mu\text{g}/\text{m}^3$. Partikler mindre end 10 μm , $PM_{2.5}$ (Bam): $\mu\text{g}/\text{m}^3$. Partikler mindre end 2,5 μm^2 .

Gasser: NO_2 : Kvælstofdioxid $\mu\text{g}/\text{m}^3$, NO_x : Kvælstofoxider $\mu\text{g}/\text{m}^3$, CO : Kulmonoxid mg/m^3 , O_3 : Ozon $\mu\text{g}/\text{m}^3$, SO_2 : Svooldioxid $\mu\text{g}/\text{m}^3$
Partikler: PM_{10} (Teom): $\mu\text{g}/\text{m}^3$. Partikler mindre end 10 μm , $PM_{2.5}$ (Bam): $\mu\text{g}/\text{m}^3$. Partikler mindre end 2,5 μm^2

Gasser og partikler, der indgår i tabellerne:

Endvidere blev der i første omgang indsamlet data fra perioden 27/10-2024 kl. 17:00 til 27/11-2024 kl. 15:00.

I forhold til scraping af data fra ENVS tog vi udgangspunkt i en GET-request, hvor vi havde vores baseurl samt gjorde vi brug af user-agent, som beskriver hvilken type browser, der sender forespørgslen. Derudover gjorde vi brug af cookie for ikke at blive begrænset, der i første omgang blev gemt i rawres. For at hente data fra de fire lokationer udtrak vi HTML-indholdet. Hertil blev vi også nødt til at bruge en CSRF-token til at autentificere vores requests, da hjemmesiden krævede en sikkerhedstoken kaldet ”__RequestVerificationToken”, som kan findes ved at åbne developer tools. For at kunne foretage POST-requests mod MainTable-endpointet, hvor målingsdata kan findes, var det nødvendigt at bruge en

² <https://envs2.au.dk/Luftdata/Presentation/table/Copenhagen/HCAB>

sikkerhedstoken ellers fik man en statuskode 500.

```
... ▼<script type="text/javascript" async="async"> == $0
    var data = {};
    data.__RequestVerificationToken = $('#__AjaxAntiForgeryForm input[name=__RequestVerificationToken]').val();
    $.ajax({
        url: "/Luftdata/Presentation/table/MainTable/Copenhagen/HCAB",
        type: "POST",
        data: data,
        success: function (response) {
            $("#mainTable").html(response);
        }
    });
    $.ajax({
        url: "/Luftdata/Presentation/table/SideTable/Copenhagen/HCAB",
        type: "POST",
        data: data,
        success: function (response) {
            $("#sideTable").html(response);
        }
    });
}
```

JavaScript-kode, der bruger en RequestVerificationToken:

For at finde tabellen, der indeholder måledataene, brugte vi en CSS-selektor, der refererer til tabelens data/placering. Dette blev efterfølgende konverteret til dataframes og gemt som .rds filer.

Resultaterne er ikke kvalitetskontrollerede.

Tidsstemplerne vises i tidszonen UTC+1. Klokken nu: 16:46 (UTC+1)

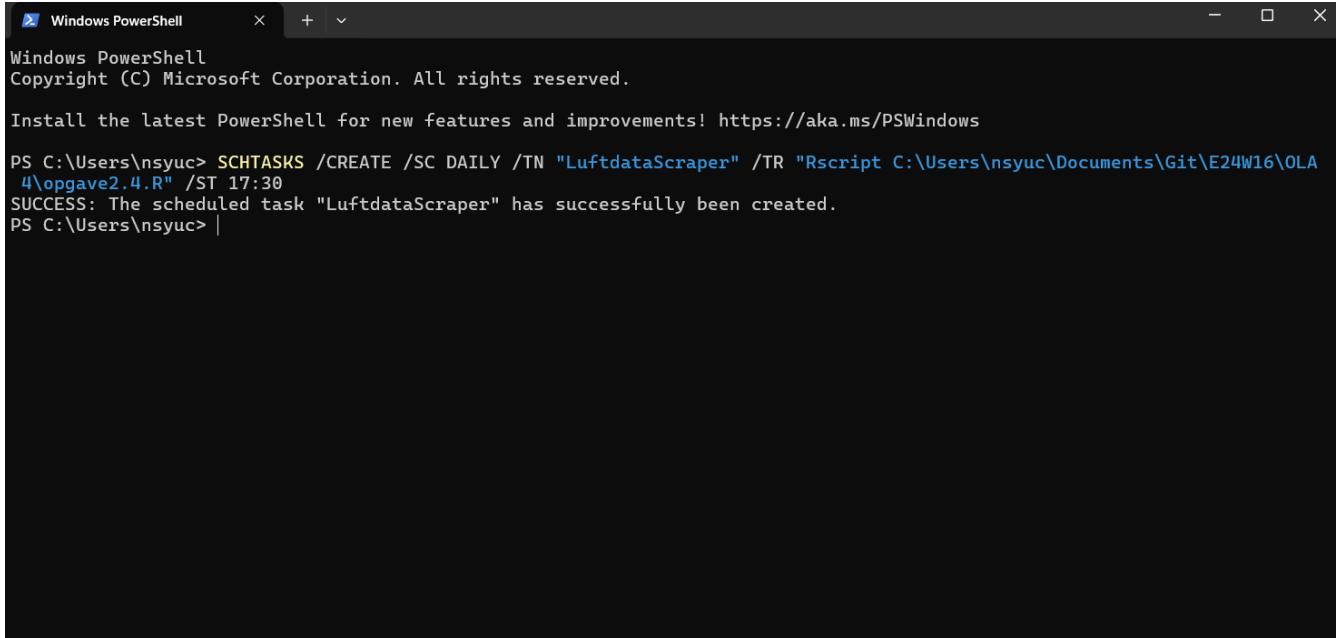
København - H. C. Andersens Boulevard							
Målt (starttid)	CO	NO ₂	NO _x	SO ₂	O ₃	PM ₁₀	PM _{2,5}
27-11-2024 15:00	0,28	42,6	77,5	1,7	25,9	22,0	12,5
27-11-2024 14:00	0,25	35,9	66,6	1,6	33,1	25,1	9,6
27-11-2024 13:00	0,21	29,0	52,7	1,5	38,4	30,6	5,8
27-11-2024 12:00	0,24	33,7	66,7	1,6	34,4	33,6	8,2
27-11-2024 11:00	0,24	34,6	70,9	0,93	29,9	21,0	13,7
27-11-2024 10:00	0,25	34,3	64,3	1,0	31,1	22,2	10,0
27-11-2024 09:00	0,28	44,7	91,5	2,4	21,2	14,1	9,0
27-11-2024 08:00	0,31	56,4	121,7	1,7	11,7	16,1	10,2
27-11-2024 07:00	0,27	49,1	109,5	1,7	18,0	24,0	5,8
27-11-2024 06:00	0,22	33,1	68,1	2,1	29,2	11,1	7,9
27-11-2024 05:00	0,19	18,3	31,8	1,1	42,5	10,9	9,7
27-11-2024 04:00	0,16	9,6	13,2		50,3	17,4	8,2
27-11-2024 03:00	0,16	8,2	10,4	0,31	53,6	9,6	8,8

© 2024 - Aarhus Universitet, Institut for Miljøvidenskab

```
body style="padding-bottom: 40px;">
  > div class="navbar navbar-inverse navbar-fixed-top">
    > div class="container body-content">
      :before
      <p class="text-danger col-lg-6 text-center">Resultaterne er ikke kvalitetskontrollerede.</p>
      <p class="text-danger col-lg-6 text-center">Tidsstemplerne vises i tidszonen UTC+1. Klokken nu: 16:46 (UTC+1)</p>
      > form id="__AjaxAntiForgeryForm" action="#" method="post">
      > h1 class="text-center">København - H. C. Andersens Boulevard
      > div class="col-lg-12 text-center">
        > div id="mainTable" class="col-lg-8">
          > div class="col-lg-12" == $0
        > /div
        > div id="sideTable" class="col-lg-4">
        > /div
      > /div
      > footer class="navbar-fixed-bottom" style="background-color: white">
        > /footer
        <script src="/Luftdata/Presentation/bundles/jquery?v=DiIze7uJxdh05fc_30wsB4VF0hiPM73uYggakL8l"></script>
        <script src="/Luftdata/Presentation/bundles/bootstrap?v=g7cxIWV5ve_i8yKtg1loBytOlgl_wBzTNealBc41"></script>
        ><script type="text/javascript" async="async"> == </script>
      </body>
    </html>
  <!-- inesvg.smil.svgclippaths body div.container.body-content div#mainTable.col-lg-8 div.col-lg-12 -->
```

Afslutningsvis blev ovenstående proces gentaget for de tre øvrige lokationer, Anholt, Banegårdsgade i Århus og Risø.

I forhold til at skulle vente en dag og så hente data igen og opdatere tabellerne med nye data gjorde vi brug af både crontab (mac-user) og windows task scheduler, som blev anvendt i windows powershell.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\nsyuc> SCHEDTASKS /CREATE /SC DAILY /TN "LuftdataScraper" /TR "Rscript C:\Users\nsyuc\Documents\Git\E24W16\0LA
4\opgave2.4.R" /ST 17:30
SUCCESS: The scheduled task "LuftdataScraper" has successfully been created.
PS C:\Users\nsyuc> |
```

Automatisering af datasættene for at opdatere tallene via Windows Powershell.

I forhold til at skulle få dataen ind i SQL gjorde vi brug af r-pakken MariaDB, hvor vi oprettede forbindelse til databasen "luftdata". Forbindelsen blev herunder konfigureret med de nødvendige parametre som localhost, root og password.

Da dataformatet fra webscraping ikke matchede SQL's foretrukne format, blev vi nødt til at få det konverteret fra det oprindelige format "DD-MM-YY HH:MM" til SQL's format "YYYY-MM-DD HH:MM" ved hjælp af strptime og format. Dette blev gjort for alle datasæt fra både d. 27/11-2024 og d. 28/11-24 for alle fire målestationer.

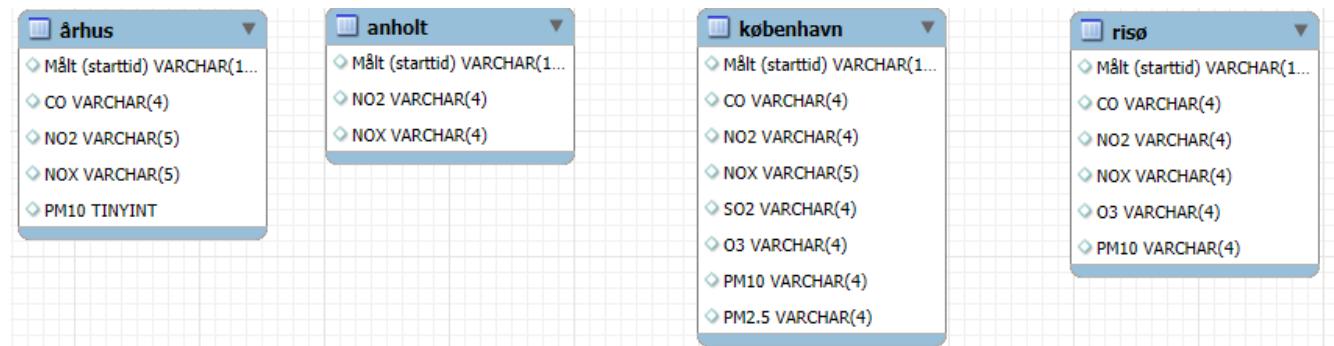
Efter formateringen blev datasættende indlæst i SQL-databasen ved hjælp af dbWriteTable funktionen. For hver målestation blev der oprettet to tabeller, én fra den første scraping som var d. 27/11-24 og én for det nye scraping som var d. 27/11-24. Herefter havde vi til opgave at skulle tilføje de nye observationer fra andet scrape til vores første scraping uden at slette data i SQL.

For at samle begge dages scraping gjorde vi brug af INSERT INTO og select kommandoer med ON DUPLICATE KEY UPDATE.

For hver målestation, herunder Århus, Anholt, København og Risø blev der gjort brug af INSERT-kommandoer, der tager hjde for de forskellige kolonner hver station måler. Fx mäter Anholt kun NO2 og NOX, mens København mäter flere parametre inkl. PM2.5 OG SO2 osv.

Hertil gjorde vi også brug af DUPLICATE KEY UPDATE, da det ville sikre, at hvis der allerede eksisterede en data på et givent tidspunkt, så ville værdierne blive opdateret med de nye data i stedet for at skabe dubletter.

Nedenstående EER diagram viser strukturen af vores database fra luftdata.



EER Diagram fra luftdata:

Opgave 3 – Analyse af logfiler

Opgave 3.1 – Rapport fra en webserver

I skal åbne logfilerne og lave en rapport over aktiviteten på webserveren. Rapporten skal indeholde en optælling af aktive ip-adresser pr døgn. Plot med de mest aktive. whois-info på den mest aktive. Gruppering på 404, herunder en beskrivelse af "mistænksomme" requests. Det værste der kan ske, er at en mistænksom request returnerer 200. Overvej hvordan man kan fange dem.

Indlæsning af log-filen (terminal)

For at kunne analysere webserveraktiviteten som beskrevet i opgaven, er det nødvendigt at downloade og placere logfilerne korrekt, så de er klar til videre behandling. Dette kræver, at vi organiserer vores arbejdsfiler (hvilket vi gør via terminalen) og gør dem let tilgængelige for senere brug i R.

Først navigeres til den mappe, hvor arbejdet med logfilerne skal foregå. I dette tilfælde er det mappen OLA4, som findes i strukturen:

```
cd /Users/sevimkilinc/Documents/Dataprojekter/OLA/OLA4
```

cd (Change Directory) bruges til at skifte til den specifikke mappe, hvor logfilerne skal placeres. Dette sikrer, at det videre arbejde foregår i den korrekte kontekst.

For at holde filstrukturen organiseret, oprettes en separat mappe under OLA4 med navnet logfiler. Dette gøres med følgende kommando:

```
mkdir logfiler
```

mkdir (Make Directory) bruges til at oprette en ny mappe. Navnet logfiler blev valgt for at tydeliggøre, at denne mappe udelukkende skal indeholde logfiler.

Flytning af logfilen til den nyoprettede mappe: Den downloadede fil log.tar, som ligger i standardmappen Downloads, skal flyttes til den nyoprettede mappe logfiler. Flytningen udføres med følgende kommando:

```
mv ~/Downloads/log.tar /Users/sevimkilinc/Documents/Dataprojekter/OLA/OLA4/logfiler
```

mv (Move) bruges til at flytte eller omdøbe en fil. Her flyttes log.tar fra Downloads til den specifikke sti, hvor logfilerne skal opbevares.

Udpakning af logfilen: Når filen er placeret i mappen logfiler, navigeres der til mappen for at udpakke log.tar. Dette gøres med:

```
cd /Users/sevimkilinc/Documents/Dataprojekter/OLA/OLA4/logfiler
```

Udpakning udføres derefter med:

```
tar -xvf log.tar
```

- tar er en kommando til at håndtere arkivfiler. Flagene -xvf står for:
 - -x: Udpakning (extract).
 - -v: Vis detaljerede oplysninger under processen (verbose).
 - -f: Specifierer filen, der skal behandles (her log.tar).

Slutvis kontrollerer vi zipfilen er udpakket korrekt vha. af "ls" (list) funktionen.

```
access.log  
access.log.1  
access.log.2.gz  
access.log.3.gz  
...  
log.tar
```

Vi kan nu se, at log.tar er udpakket ud fra overstående output.

Databehandling af logfilerne (Datascience modellen):

1. Dataindsamling

Formål: Hente logdata fra en webserver og samle dem til én datakilde.

- **setwd():** Bruges til at angive stien til mappen med logfiler
- **list.files():** Identificerer alle filer, der matcher mønsteret "**access***", dvs. filer relateret til adgangslogfiler.
- **lapply(Logfiles, readLines):** Læser indholdet af hver logfil linje for linje og gemmer dem som en liste i variablen **log_content**.
- **do.call(c, log_content):** Samler alle linjer fra alle logfiler til én samlet variabel **all_logs**.

```
setwd("/Users/sevimkilinc/Documents/Dataprojekter/OLA/OLA4/logfiler")
file.exists("/Users/sevimkilinc/Documents/Dataprojekter/OLA/OLA4/logfiler") |
Logfiles <- list.files(pattern = "access*", path = ".")
log_content <- lapply(Logfiles, readLines)

# Kombiner indholdet af alle logfiler
all_logs <- do.call(c, log_content)
```

2. Datarensning:

Formål: Uddrage de nødvendige informationer fra de rå logfiler.

- **extract_log_data()**: En "selv defineret" funktion, der udtrækker de centrale elementer fra hver linje i logfilen:
 - **IP-adresse**: Den første del af linjen.
 - **Statuskode**: Den HTTP-statuskode, der angiver succes eller fejl (fx 200, 404).
 - **Path**: URL-stien, der blev forespurgt (fx /login, /wp-admin).
 - **Tidspunkt**: Hvornår forespørgslen blev foretaget.
- **t(sapply(all_logs, extract_log_data))**: Anvender funktionen på hver linje i logfilen og gemmer resultatet i en matrix.

```


extract_log_data <- function(raw_log) {
  ip <- str_extract(raw_log, "\\\\S+") # Første ord: IP-adresse
  status <- str_extract(raw_log, "\\s\\\\d{3}\\\\s") # HTTP-statuskode
  path <- str_extract(raw_log, "\\\\""(GET|POST|HEAD)\\\\s(.*)\\\\sHTTP") # Path fra forespørgsel
  time <- str_extract(raw_log, "\\\\[(.*)\\\\]") # Tidsstempel
  return(c(ip, time, path, status))
}

parsed_logs <- t(sapply(all_logs, extract_log_data))
colnames(parsed_logs) <- c("IP", "Time", "Path", "Status")
logs_prep <- as.data.frame(parsed_logs, stringsAsFactors = FALSE)


```

3. Dataforberedelse

Formål: Klargøre data til analyse ved at formaterne og tilføje nye variable.

- **Dato og tid:**
 - **sub(":(.*)", "", logs_prep\$Time)**: Fjerner tidsoplysninger (hh:mm:ss) fra dato'en.
 - **gsub("\\[", "", logs_prep\$date)**: Fjerner specialtegn som venstre parenteser.
 - **as.Date(logs_prep\$date, format = "%d/%b/%Y")**: Konverterer datoer til en standardiseret Date-type.
- **Nøjagtig tid:**
 - **sub("^.*:(\\d{2}:\\d{2}:\\d{2}).*", "\\1", logs_prep\$Time)**: Uddrager kun tidskomponenten (hh:mm:ss) fra loglinjen.

4. Dataanalyse

Formål: Oprette en struktureret og analyserbar version af logdata.

- **structured_logs <- logs_prep**: Den færdige data gemmes i en ny variabel structured_logs, som nu kan bruges til videre analyser (fx optælling af IP'er, mistænksomme requests, analyse af statuskoder osv.).

Aktive IP adresser pr. døgn.

Vi har analyseret logfilerne for at identificere det daglige antal unikke IP-adresser. En **unik IP-adresse** repræsenterer en enkelt enhed (f.eks. en bruger, computer eller anden enhed), der har sendt forespørgsler til serveren. Da vi skal analysere IP-aktiviteten af logfilerne, har vi valgt at tage følgende strategiske tilgang:

1. Optælling af aktive IP-adresser pr. dag

- Vi har grupperet forespørgslerne efter dato og talt antallet af unikke IP-adresser pr. dag. Dette giver et overblik over, hvordan aktiviteten varierer fra dag til dag, og hvilke datoer der har højeste aktivitet.

2. Identificering af de mest aktive IP-adresser

- Ved at tælle antallet af forespørgsler for hver IP-adresse har vi identificeret de mest aktive IP'er. Disse IP'er kan stå for en betydelig del af trafikken og kan være relevante for videre undersøgelse.

Optælling af aktive IP-adresser pr. dag:

Denne fremgangsmåde giver et klart overblik over, hvor mange forskellige enheder der har kommunikeret med serveren i løbet af en given dag. Det er vigtigt at bemærke, at hver unik IP kun tælles én gang pr. dag, uanset hvor mange gange den samme IP har sendt forespørgsler til serveren. Med andre ord dækker optællingen kun over unikke IP-adresser pr. server og ikke antallet af forsøg fra den samme IP-adresse.

Koden anvender funktionen `aggregate()`, som sikrer, at hver IP kun tælles én gang pr. dato.

```
active_ips_per_day <- aggregate(IP ~ Date, data = structured_logs, FUN = function(x) length(unique(x)))
colnames(active_ips_per_day) <- c("Date", "UniqueIPs")
```

Nedenstående viser resultatet dataframen "active_ips_per_day" som viser, hvordan aktiviteten varierer dagligt, og hvilke datoer der har højest IP-aktivitet:

	Date	UniqueIPs
1	2022-11-08	167
2	2023-10-26	143
3	2023-10-27	165
4	2023-10-28	163
5	2023-10-29	139
6	2023-10-30	157
7	2023-10-31	137
8	2023-11-01	198
9	2023-11-02	246
10	2023-11-03	179
11	2023-11-04	161
12	2023-11-05	74
13	2023-11-06	86
14	2023-11-07	93
15	2023-11-08	91
16	2023-11-09	34

Identificering af de mest aktive IP-adresser:

Vi har sorteret de samlede forekomster af hver IP-adresse i logfilen, hvilket giver et klart billede af de IP'er, der genererer størstedelen af trafikken

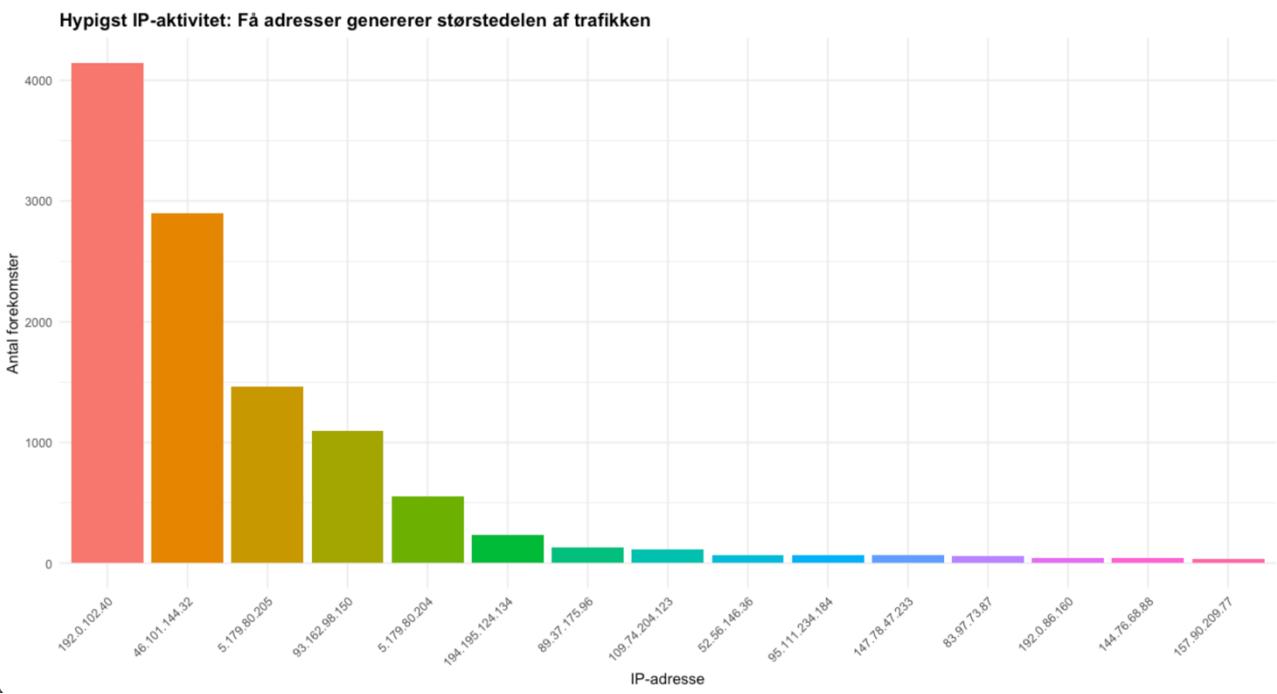
Ved at bruge funktionen **table()** har vi talt antallet af forespørgsler for hver IP, og de mest aktive IP'er er identificeret ved at sortere i faldende rækkefølge.

```
# Optælling af forekomster for hver IP-adresse
IP_antal <- table(structured_logs$IP)
sorted_IP_antal <- sort(IP_antal, decreasing = TRUE)
Antal_forekomster_IP <- as.data.frame(sorted_IP_antal)
colnames(Antal_forekomster_IP) <- c("IP", "Antal")
```

Overstående resulteret i nedenstående dataframe "Antal_forekomster_IP":

	IP	Antal
1	192.0.102.40	4145
2	46.101.144.32	2902
3	5.179.80.205	1464
4	93.162.98.150	1094
5	5.179.80.204	555
6	194.195.124.134	233
7	89.37.175.96	128
8	109.74.204.123	115
9	52.56.146.36	70
10	95.111.234.184	66
11	147.78.47.233	64
12	83.97.73.87	58
13	192.0.86.160	44
14	144.76.68.88	42
15	157.90.209.77	39
16	159.65.197.63	38
17	162.240.239.98	36
18	64.227.108.223	30
19	103.237.86.234	28
20	157.245.40.165	28
21	192.0.86.85	28

For at visualiserer aktive IP- adresser pr. døgn, har vi udvalgt op 15 trafikerede IP-adresser:



PLOT 1: GRAFEN VISER, AT EN LILLE GRUPPE IP-ADRESSER STÅR FOR LANGT STØRSTEDELEN AF TRAFIKKEN, HVOR DEN MEST AKTIVE IP ADRESSE ALENE HAR OVER 4000 REQUEST.

Plot af mest aktive IP adresse

Nedenstående kode illustrerer, hvordan vi ved hjælp af `GET()`-funktionen fra `httr`-pakken sender en GET-forespørgsel til den konstruerede URL "<http://ipinfo.io/>". Denne forespørgsel anmoder om data fra `ipinfo.io` for den specifikke IP-adresse. Funktionen `fromJSON(rawToChar(response$content))` konverterer de modtagne rådata fra JSON-format til en liste "response", hvilket gør det nemmere at arbejde med dataene i R.

```
Top_1_active_IP <- "192.0.102.40"

# Hent oplysninger for IP-adressen
response <- GET(paste0("https://ipinfo.io/", Top_1_active_IP, "/json"))
data <- fromJSON(rawToChar(response$content))
cat(paste("IP:", data$ip, "\nCITY:", data$city, "\nREGION:",
          data$region, "\nCOUNTRY:", data$country, "\nLOCATION:", data$loc, "\n"))
```

Efter vores API-kald via **IPinfo** kunne vi indhente detaljerede oplysninger om den mest aktive IP-adresse: **192.0.102.40**. Disse data inkluderer blandt andet geografisk placering (latitude og longitude) samt by, region og land (bliver printet i consolen ud af overstående loop vha. `cat` funktionen, fx koordinaterne som er med til at definere latitude og longitude)

IP Location	United States Marina Del Rey Early Registration Addresses
ASN	AS2635 AUTOMATTIC, US (registered Oct 01, 2012)
Whois Server	whois.arin.net
IP Address	192.0.102.40

Denne server, tilknyttet IP-adressen **192.0.102.40**, er registreret under **AUTOMATTIC**, en velkendt amerikansk hostingudbyder, og er placeret i Marina Del Rey, USA. Det høje aktivitetsniveau gør serveren til

den mest aktive IP-adresse i vores data, hvilket potentielt kan skyldes legitime forespørgsler, hvis serveren f.eks. bruges til en populær tjeneste som WordPress.

Dog kan det også indikere mistænkelig aktivitet, hvis serveren er blevet kompromitteret eller misbrugt til at generere unormal trafik. Dette aspekt vil blive analyseret og gennemgået nærmere i en senere del af opgaven. Her fokuserer vi alene på at identificere, at **192.0.102.40** er den mest aktive IP-adresse i vores analyse.

Mistænksomme request:

Vi har valgt at fokusere på antallet af forespørgsler ud fra statuskode 404 (da gruppering af denne statuskode både kræves i opgaven og kan være indikativ for mistænkelig adfærd) som den primære indikator for mistænkelig aktivitet. Statuskode 404 afslører hurtigt mønstre, der ofte forbindes med bot- og hackerangreb. Højfrekvente forespørgsler mod paths som *wp-admin*, *.env*, og *git/config* indikerer typisk automatiserede angreb eller forsøg på at udnytte sårbarheder. En alternativ tilgang kunne have været at analysere forespørgsler baseret på nøgleord som *wp*, *php*, og *git* ved at ekstrahere disse oplysninger fra *path*-kolonnen.

Metodisk tilgang til analyse af mistænkelige aktiviteter

For at identificere potentielt mistænkelig aktivitet i logfilerne har vi anvendt en struktureret tilgang, der fokuserer på HTTP-forespørgsler med statuskode 404. Denne metode gør det muligt at fremhæve mønstre, der kan indikere automatiserede angreb eller forsøg på at finde sårbare filer. Den overordnede fremgangsmåde kan beskrives som følger:

Filtrering af statuskode 404

Logfilerne filtreres, så kun forespørgsler med statuskode 404 inkluderes. Dette isolerer de relevante data:

```
logs_404 <- structured_logs[structured_logs$status == 404, ]
```

Gruppering og optælling

Dataene grapperes efter kombinationen af Path og IP, hvorefter antallet af 404-fejl (Count404) tælles for hver kombination. Denne tilgang giver mulighed for at identificere de specifikke IP-adresser og paths, der oftest er forbundet med fejl. Det høje antal forespørgsler mod ressourcer som *.env*, *wp-admin*, og *.git/config* kan indikere automatiserede scanninger eller angrebsmønstre.

```
Path_404_summary <- aggregate(Status ~ Path + IP, data = logs_404, FUN = length)
colnames(Path_404_summary) <- c("Path", "IP", "Count404")
```

Resultatet sorteres efter antallet af fejl for at fremhæve de mest mistænksomme kombinationer, hvilket udgør tabellen sorted_path_404:

		IP	Count404
109.237.97.180	1 "POST /wp-admin/admin-ajax.php HTTP	93.162.98.150	331
	2 "POST /wp-admin/admin-ajax.php HTTP	5.179.80.204	41
	3 "GET /actuator/gateway/routes HTTP	83.97.73.87	25
	4 "GET /.git/config HTTP	170.64.220.120	15
	5 "GET /robots.txt HTTP	157.90.209.77	14
	6 "GET /docker-compose.yml HTTP	159.100.22.187	14
	7 "GET /sitemap HTTP	157.90.209.77	12
	8 "GET /sitemap.txt HTTP	157.90.209.77	12
	9 "POST /core/.env HTTP	162.240.239.98	12
	10 "GET /favicon.ico HTTP	5.179.80.205	12

Ved at inkludere tællingerne (Count404) prioriteres de mest mistænksomme kombinationer, hvilket hjælper med at fokusere på de IP-adresser og paths, der er mest sandsynlige kilder til scanninger eller angreb.

Udvælgelse af mistænksomme IP'er:

- De 10 IP-adresser med flest 404-fejl vælges fra den tidligere grupperede og sorterede tabel (sorted_path_404), da disse vurderes til at være dem med højeste frekvens statuskode 404.
- Listen over unikke IP'er genereres for at sikre, at hver IP kun inkluderes én gang.

```
Top_10_suspicious <- head(sorted_path_404, 10)
suspicious_ips <- unique(Top_10_suspicious$IP)
```

Nedestående funktion muliggør, at vi kan indhente information på vores top 10 mistænkelige IP-adresser. Find forklaring i tidligere opgave om aktivitet.

```
# Funktion til at hente IP-data
fetch_ip_details <- function(ip) {
  response <- GET(paste0("https://ipinfo.io/", ip, "/json"))
  if (status_code(response) == 200) {
    data <- fromJSON(rawToChar(response$content))
    loc <- strsplit(data$loc, ",")[[1]]
    return(data.frame(
      IP = ip,
      City = data$city,
      Region = data$region,
      Country = data$country,
      Latitude = as.numeric(loc[1]),
      Longitude = as.numeric(loc[2]),
      stringsAsFactors = FALSE
    ))
  }
  return(NULL)
}

# Hent data for de 10 mistænksomme IP'er
top_10_suspicious_IP_details <- do.call(rbind, lapply(top_10_suspicious_IPs, fetch_ip_details))
```

Vi kører nedenstående kode for at behandle data fra IPinfo. Under analysen blev det tydeligt, at visse IP-adresser rapporteres med samme geografiske koordinater (longitude og latitude). Disse duplikater kan skyldes delte servere eller netværk. For at sikre en præcis analyse fjernes disse dubletter, så hver lokation kun fremgår én gang i datasættet. Efter denne behandling står vi tilbage med 7 unikke IP-adresser, som repræsenterer de mest mistænsomme lokationer.

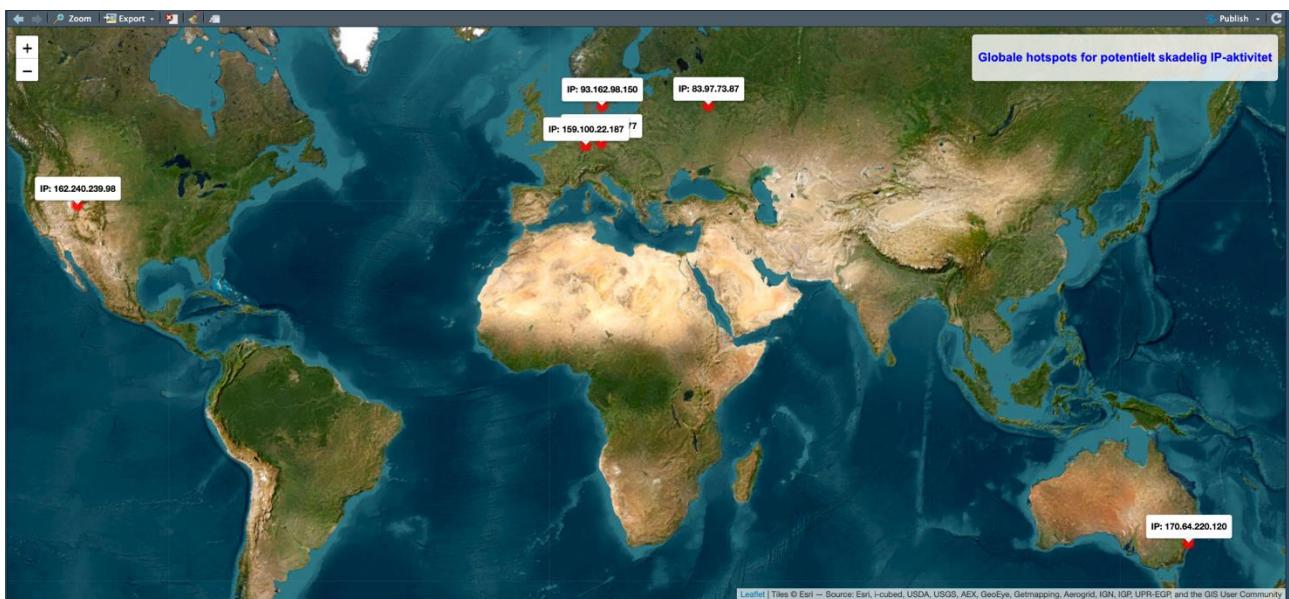
```
# Hent data for de mistænsomme IP'er
top_suspicious_IP_details <- do.call(rbind, lapply(top_10_suspicious_IPs, fetch_ip_details))

# Fjern dubletter baseret på Latitude og Longitude
top_suspicious_IP_details <- top_suspicious_IP_details[!duplicated(top_suspicious_IP_details[, c("Latitude", "Longitude")]), ]
```

VI ENDER DERMED MED FØLGENDE DATAFRAME:

	IP	City	Region	Country	Latitude	Longitude
1	93.162.98.150	Copenhagen	Capital Region	DK	55.6759	12.5655
3	83.97.73.87	Moscow	Moscow	RU	55.7522	37.6156
4	170.64.220.120	Sydney	New South Wales	AU	-33.9092	151.1940
5	157.90.209.77	Falkenstein	Saxony	DE	50.4779	12.3713
6	159.100.22.187	Frankfurt am Main	Hesse	DE	50.1155	8.6842
7	162.240.239.98	Provo	Utah	US	40.2338	-111.6585

UD FRA OVERSTÅENDE DATAFRAME, KAN VI NU LAVE EN INTIATIV GRAF VHA. LEAFLET PAKKEN I R.



Figur 2: Kortet viser globale hotspots for potentieligt skadelig IP-aktivitet baseret på de 7 mest mistænsomme IP-adresser. Hver marker repræsenterer en unik lokation, hvorfra der er registreret et højt antal 404-fejl på specifikke ressourcer. Aktivitet er identificeret i følgende lande: USA, Danmark, Tyskland, Rusland, og Australien.

Andre mønstre mistænkelige request

Som tidligere nævnt ville vi undersøge følgende IP-adresse **192.0.102.40**. Da denne IP-adresse ikke havde en eneste statuskode 404, men derimod kun 200, blev den ikke medtaget i den tidligere analyse af mistænsomme forespørgsler. Den inddrages her for at demonstrere, hvordan mistænkelig aktivitet kan variere i form og mønster.

Den mistænkelige aktivitet ved IP-adressen **192.0.102.40** kan sammenfattes således:

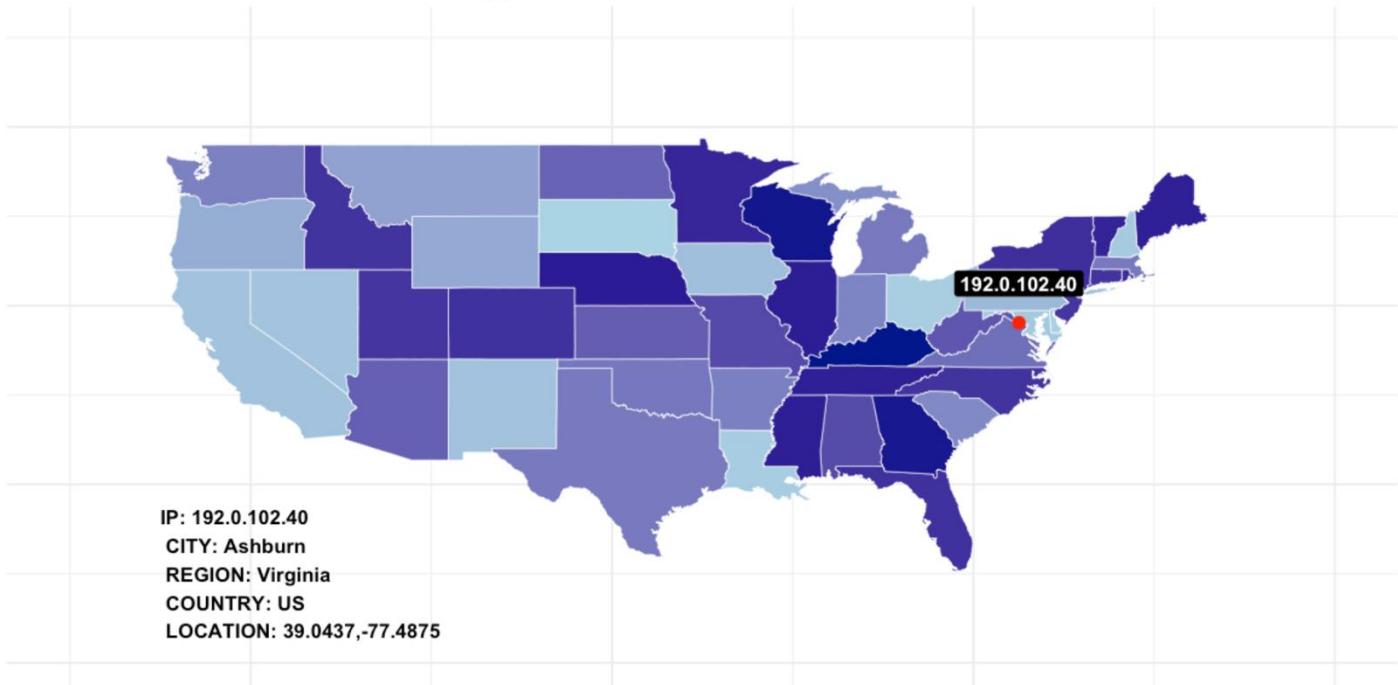
- **Ekstrem høj aktivitet:** IP-adressen genererede **4115 forespørgsler**, hvilket er en betydelig del af de samlede forespørgsler i logfilen. Blandt de **15.000 samlede forespørgsler** var **12.289** markeret som statuskode **200**, svarende til **81,9%** af alle forespørgsler. At én IP-adresse bidrager så markant til dette antal er i sig selv mistænkeligt.

IP	Time	Path	Status	Date	Exacttime
1 192.0.102.40	[09/Nov/2023:00:04:38 +0000]	"HEAD / HTTP	200	2023-11-09	00:04:38
2 192.0.102.40	[09/Nov/2023:00:09:38 +0000]	"HEAD / HTTP	200	2023-11-09	00:09:38
3 192.0.102.40	[09/Nov/2023:00:14:39 +0000]	"HEAD / HTTP	200	2023-11-09	00:14:39
4 192.0.102.40	[09/Nov/2023:00:19:38 +0000]	"HEAD / HTTP	200	2023-11-09	00:19:38
5 192.0.102.40	[09/Nov/2023:00:24:38 +0000]	"HEAD / HTTP	200	2023-11-09	00:24:38
6 192.0.102.40	[09/Nov/2023:00:29:38 +0000]	"HEAD / HTTP	200	2023-11-09	00:29:38
7 192.0.102.40	[09/Nov/2023:00:34:38 +0000]	"HEAD / HTTP	200	2023-11-09	00:34:38
8 192.0.102.40	[09/Nov/2023:00:39:38 +0000]	"HEAD / HTTP	200	2023-11-09	00:39:38
9 192.0.102.40	[09/Nov/2023:00:44:38 +0000]	"HEAD / HTTP	200	2023-11-09	00:44:38
10 192.0.102.40	[09/Nov/2023:00:49:38 +0000]	"HEAD / HTTP	200	2023-11-09	00:49:38
11 192.0.102.40	[09/Nov/2023:00:54:37 +0000]	"HEAD / HTTP	200	2023-11-09	00:54:37
12 192.0.102.40	[09/Nov/2023:00:59:38 +0000]	"HEAD / HTTP	200	2023-11-09	00:59:38
13 192.0.102.40	[09/Nov/2023:01:04:38 +0000]	"HEAD / HTTP	200	2023-11-09	01:04:38
15 192.0.102.40	[09/Nov/2023:01:09:37 +0000]	"HEAD / HTTP	200	2023-11-09	01:09:37
16 192.0.102.40	[09/Nov/2023:01:14:38 +0000]	"HEAD / HTTP	200	2023-11-09	01:14:38
17 192.0.102.40	[09/Nov/2023:01:19:38 +0000]	"HEAD / HTTP	200	2023-11-09	01:19:38
18 192.0.102.40	[09/Nov/2023:01:24:38 +0000]	"HEAD / HTTP	200	2023-11-09	01:24:38
19 192.0.102.40	[09/Nov/2023:01:29:39 +0000]	"HEAD / HTTP	200	2023-11-09	01:29:39
20 192.0.102.40	[09/Nov/2023:01:34:36 +0000]	"HEAD / HTTP	200	2023-11-09	01:34:36
21 192.0.102.40	[09/Nov/2023:01:39:37 +0000]	"HEAD / HTTP	200	2023-11-09	01:39:37
22 192.0.102.40	[09/Nov/2023:01:44:38 +0000]	"HEAD / HTTP	200	2023-11-09	01:44:38
23 192.0.102.40	[09/Nov/2023:01:49:37 +0000]	"HEAD / HTTP	200	2023-11-09	01:49:37
24 192.0.102.40	[09/Nov/2023:01:54:37 +0000]	"HEAD / HTTP	200	2023-11-09	01:54:37
26 192.0.102.40	[09/Nov/2023:01:59:38 +0000]	"HEAD / HTTP	200	2023-11-09	01:59:38

- **Forespørgselstype HEAD:** Alle forespørgsler fra denne IP var af typen **HEAD**, som kun henter metadata fra serveren uden at indlæse selve indholdet (tag også de korte tidsintervaller i betragtning). Dette er usædvanligt for normal trafik, da typiske forespørgsler ofte kombinerer HEAD med GET. En sådan ensidig forespørgselstype er ofte karakteristisk for automatiserede scripts eller scanninger.

- **Indikation på automatiseret aktivitet:** Kombinationen af et stort antal forespørgsler (alle med HEAD) og fraværet af fejlstatusser (f.eks. 404) kan indikere, at denne IP er en del af et script eller en bot, der systematisk scanner serveren.
- Denne analyse demonstrerer, hvordan mistænksom aktivitet ikke nødvendigvis kræver fejlstatusser, men kan identificeres gennem unormale forespørgselsmønstre og en markant overvægt af forespørgselstypen HEAD.

USA toppler listen med den mest aktive IP-adresse:



GRAF 3: GRAFEN VISER, AT USA, REPRÆSENTERET VED IP-ADRESSEN 192.0.102.40, TOPPER LISTEN SOM DEN MEST AKTIVE IP-ADRESSE MED LOKATION I ASHBURN, VIRGINIA. DENNE IP STÅR FOR EN BETYDELIG ANDEL AF TRAFIKKEN OG VIL BLIVE UNDERSØGT NÆRMERE SENERE I OPGAVEN FOR POTENTIELT MISTÆNKELIG AKTIVITET.

Denne specifikke IP-adresse nævnes som en del af en senere vurdering af mistænkelig aktivitet, hvor dens trafik vil blive diskuteret i forhold til, om den kan betragtes som mistænkelig eller legitim. Dette vil blive behandlet i en efterfølgende del af opgaven.

Ekstra noter til mistænksomme request

I en normal kontekst betyder statuskoden **200**, at serveren har behandlet en forespørgsel korrekt og returnerer den ønskede ressource, f.eks. adgang til en hjemmeside. Dette indikerer, at alt fungerer som forventet.

En udfordring opstår dog, hvis en **mistænksom request** (f.eks. forespørgsler til administrative endpoints som /wp-admin, /login, eller .php-filer) returnerer statuskoden **200**. Dette kan indikere, at hackere eller bots potentielt har fået uautoriseret adgang til ressourcer, som de ikke burde have adgang til.

Mistænkelige mønstre og IP-aktivitet

1. Gentagne forespørgsler fra en IP:

- Hvis en IP-adresse gentagne gange sender forespørgsler til forskellige paths og modtager statuskoden **200**, kan det være mistænkeligt. Dette gælder især, hvis paths er kendt for at være administrative eller sensitive.

2. Eksempler på mistænkelige paths:

- Paths, der normalt kun er tilgængelige for autoriserede brugere:
 - /wp-admin
 - /login
 - /config.php

3. Tidsmønstre:

- Mistænkelige requests forekommer ofte:
 - Uden for normal arbejdstid.
 - Med meget høj frevens.
- Brug af tidsstempler kan hjælpe med at identificere og analysere normale vs. unormale mønstre.

Anbefalinger for at identificere mistænksomme requests

- **Analyser paths:** Overvåg requests til administrative eller følsomme paths og tjek, om de returnerer statuskoden **200** uden at være autoriseret.
- **Overvåg IP-aktivitet:** Identificér IP'er med gentagne requests til mistænkelige paths.

Brug tidsmønstre: Sammenligne tidspunkter for requests med normale arbejdstider for at finde uregelmæssigheder.