

2021

Big Data: lagring og bearbeiding(ITAL25019-1 21H)

AV HENRIETTE MATHISEN OG KHAYAM NAMI
GRUPPE 343

Innhold:

- [Datasett](#)
 - [Milepål 1](#)
 - [Milepål 2 - Design av nøkkel-verdidatabase](#)
 - [Milepål 3 - Implementasjon av nøkkel-verdi databasen og design av dokumentdatabasen](#)
 - [Milepål 4 - Implementasjon av dokumentdatabase design av kolonnefamilie og grafdatabase](#)
 - [Milepål 5 - Spark og HDFS](#)
 - [Milepål 6 - Cassandra og Joins](#)
-

Datasett

[**Violent Crime Rates by US State:**](#)

Forfatter: [Mehmet Akturk](#)

Kilden til datasettet: *World Almanac and Book of facts 1975. (Crime rates), Statistical Abstracts of the United States 1975. (Urban rates)*.

Periode: 1975

Beskrivelse(engelsk):

This data set contains statistics, in arrests per 100,000 residents for assault, murder, and rape in each of the 50 US states in 1973. Also given is the percent of the population living in urban areas.

A data frame with 50 observations on 4 variables.

Murder is numeric and Murder arrests (per 100,000)
Assault is numeric and Assault arrests (per 100,000)
UrbanPop is numeric and UrbanPop arrests (per 100,000)
Rape is numeric and Rape arrests (per 100,000)

[**US Police Shootings**](#)

Forfatter: [Ahsen Nazir](#)

Kilden til datasettet: [Washington Post](#)

Periode: 1. Januar 2015 -

Beskrivelse(engelsk):

It contains basic data about people like their name, age, gender and race. Along with it, is the shooting/killing information, like date of event, where it happened? how they were shot? did they attack? Were they holding weapons? Did they show any mental illness? Was the policeman wearing a camera/was the incident recorded? Did the suspect flee? Apart from that, a category column holds type of weapon used by the suspect

US Estimated Crimes

Forfatter: [Bojan Tunguz](#)

Kilden til datasettet: [FBI CRIME STATISTICS](#)

Periode: 1979 - 2019

Beskrivelse(engelsk):

This dataset contains estimated data at the state and national level and was derived from the Summary Reporting System (SRS). These data reflect the estimates the FBI has traditionally included in its annual publications. Download this dataset to see the FBI's estimated crime totals for the nation and all 50 states.

Drug Use By Age

Forfatter: [Bojan Tunguz](#)

Kilden til datasettet: [National Survey on Drug Use and Health from the Substance Abuse and Mental Health Data Archive](#)

Periode: 2015

Beskrivelse(engelsk):

This directory contains data behind the story [How Baby Boomers Get High](#). It covers 13 drugs across 17 age groups.

Source: [National Survey on Drug Use and Health from the Substance Abuse and Mental Health Data Archive](#).

alcohol-use

Percentage of those in an age group who used alcohol in the past 12 months

alcohol-frequency

Median number of times a user in an age group used alcohol in the past 12 months

marijuana-use

Percentage of those in an age group who used marijuana in the past 12 months

marijuana-frequency

Median number of times a user in an age group used marijuana in the past 12 months

cocaine-use

Percentage of those in an age group who used cocaine in the past 12 months

cocaine-frequency

Median number of times a user in an age group used cocaine in the past 12 months

crack-use

Percentage of those in an age group who used crack in the past 12 months

crack-frequency

Median number of times a user in an age group used crack in the past 12 months

heroin-use

Percentage of those in an age group who used heroin in the past 12 months

heroin-frequency

Median number of times a user in an age group used heroin in the past 12 months

hallucinogen-use

Percentage of those in an age group who used hallucinogens in the past 12 months

hallucinogen-frequency

Median number of times a user in an age group used hallucinogens in the past 12 months

halucinogen-use

Percentage of those in an age group who used inhalants in the past 12 months

inhalant-frequency

Median number of times a user in an age group used inhalants in the past 12 months

inhalant-use

Percentage of those in an age group who used pain relievers in the past 12 months

pain-releiver-frequency

Median number of times a user in an age group used pain relievers in the past 12 months

pain-releiver-use

Percentage of those in an age group who used oxycontin in the past 12 months

oxycontin-frequency

Median number of times a user in an age group used oxycontin in the past 12 months

oxycontin-use

Percentage of those in an age group who used tranquilizer in the past 12 months

tranquilizer-frequency

Median number of times a user in an age group used tranquilizer in the past 12 months

tranquilizer-use

Percentage of those in an age group who used stimulants in the past 12 months

stimulant-frequency

Median number of times a user in an age group used stimulants in the past 12 months

meth-use

Percentage of those in an age group who used meth in the past 12 months

meth-frequency

Median number of times a user in an age group used meth in the past 12 months

sedative-use

Percentage of those in an age group who used sedatives in the past 12 months

sedative-frequency

Median number of times a user in an age group used sedatives in the past 12 months

Milepæl 1

Oppgavetekst:

1. Finn datasettene dere skal jobbe med.

Finn 4 forskjellige datasett hvor dere ser for dere at hvert datasett individuelt passer godt til hver av de fire NO-SQL databasemotortypene vi gjennomgår i dette emne. Men hvert datasett skal kunne kobles med de andre på en naturlig måte. F.eks. Kan et datasett som inneholder faktiske reisetider med fly mellom flyplasser kobles med et datasett som inneholder været på de flyplassene.

Beskriv datasettene i rapporten og hvordan de kan passe sammen.

Dere kan nesten helt fritt velge datasett, men her er noen forslag til kilder:

- <https://www.kaggle.com/datasets> (Lenker til en ekstern side.)
- <https://archive.ics.uci.edu/ml/datasets.php> (Lenker til en ekstern side.)
- Jobb (merk at dere vil måtte legge ved datasettet i endelig innlevering)
- Andre kilder

Tips:

- Det kan være enklere å gjøre dette ved å først velge et datasett som inneholder data fra et domene dere kjenner, som en idrett, og deretter prøve å målrettet finne 3 datasett til som dere kan koble til det datasettet.
- Prøv å være kreativ på hvordan datasettene kan kobles.
- Hvilken databasemotor et datasett passer til er i stor grad påvirket av hvordan det skal brukes. Det er ikke utenkelig at for en anvendelse av et datasett vil det være best med en grafdatabase men for en annen så vil det passe best med en nøkkel-verdi-database.
- Dere kan velge de fleste tekst-baserte filformater i dette emnet, men om dere ikke ønsker å skape mye unødvendig arbeid for dere selv, så kan det være smart å velge csv-filer. Husk også at dere nå finner ekte datasett som det ikke er sikkert at er like pene som de dere er vant til å få utlevert i andre emner. Pass derfor på å undersøke datasettene dere velger nøyne for å oppdage mulige problemer med filene som kan føre til at de er vanskeligere å bruke enn dere først tror.

2. Tegn skisse for en nettside

Skisser en enkel nettside som lar en bruker kunne utforske koblingene mellom datasettene dere har valgt. Her skal det vises beregnede verdier som utnytter data flere datasett, samt lar en bruker legge inn nye poster i datasettene.

Dere skal i rapporten legge ved skissene samt beskrivelse av hvordan nettsiden skal fungere. Her skal dere også beskrive hvordan verdiene dere har tenkt til å vise kan beregnes (ikke tenk på implementasjon, men istedenfor hvordan dere ville forklart hvordan det til en kollega som ikke har en teknisk bakgrunn).

Svar

Del 1

Vi har valgt "Kriminalitet i USA" som vårt hovedtema. Datasettene som er valgt gir en oversikt over kriminalitetsnivåene i USA og gir oss muligheten til å se om det er noen

koblinger som ikke er så synlig. Er det noen former for kriminalitet som er koblet mere med visse typer rusmidler f. eks.

Hoved datasettet vårt har vi valgt å være "[Violent Crime rates by US State](#)". Dette datasettet passer å jobbe fra for å finne ut av følgende med hjelp fra andre datasett:

- Kriminalitetsrater i forskjellige statene. Har dette noe korrelasjon med visse typer rusmidler?
- Er det flere "police shootings" av en type "rase"? Og hvilken aldersgruppe er de fleste offrene i?
- Hvordan har utviklingen av kriminalitet påvirket kriminalitet i de forskjellige statene gjennom årene?

Hvilke databasemotorer passer til de forskjellige datasettene?

Når vi ser på datasettene så er det ganske tydelig at flere av dem kan passe til flere typer databasemotorer av NoSQL typen.

Key-value

En type nøkkel-verdi basert database som faller under NoSQL paraplyen av ikke-relasjonsbaserte databaser. Som navn tilsier så er denne typen database basert på en samling av nøkkel-verdig par hvor nøkkelen er en unik identifikator for verdiene i paret. Disse nøklene kan enten være definert på forhånd eller være tilfeldig generert. Videre så kan verdiene være alt fra simple ting som integers, floats, chars og lignende. Men det kan også være mer komplekse objekter som for eksempel JSON objekter.

Dette gjør lesing og skriving til denne typen database motor

Et eksempel på forhåndsvalgt nøkkel

Key	Verdi
"Delaware-arrests"	5.9

Et eksempel på tilfeldig generert nøkkel

Key	Verdi
964297dfy9	5.9

Fordeler

- Enkle/simple data formater gjør utføring av skrive/lese operasjoner veldig fort.

- Som sagt så kan verdien være så og si hva som helst av type data. Dette åpner vei for å ha en mer fleksibel schema struktur.

Ulempor

- Optimalisert kun for en nøkkel og verdi. En parser er nødvendig om man har ønske om å lagre verdier på hver nøkkel.

Hvilke datasett passer?

"[Violent Crime rates by US State](#)" er et datasett hvor Key-Value databaser vil være passende å bruke på grunn av typen innhold. I innholdet så ser vi at det baserer seg på "crime rates" i de forskjellige statene i USA. På grunn av at det er liten sansynlighet for at disse statene endrer navn så kan vi bruke navnet til staten som nøkkelen og de forskjellige crime ratene som verdier. Disse verdiene kan f. eks være i en array eller som et objekt.

Dette gjør at vi har en veldig simpel struktur i databasen hvor det vil være lett å lese/skrive av en høy antall ganger.

Andre datasett i listen som passer til key-value databaser er "[Drug Use By Age](#)" som er også ganske lignende i hvordan vi kunne strukturert den i en key-value database. Her kan aldersgruppene være nøkkelen og de forskjellige rusmiddlene kan være verdiene i enten et objekt eller array.

Document

Dokument-baserte er en NoSQL database som lagrer dataene i form av JSON-lignende dokumenter istedenfor kolonner og rader. Disse dokumentene er lagret i "collections" og former databasen. Skjente dokumentdatabaser er f. eks MongoDB og Amazon DyanomoDB

Hvert dokument består en eller flere nøkkel-verdi par. Et eksempel på et dokument som inneholder flere par.

```
{  
  "ID": "001",  
  "Art": "Fisk",  
  "Navn": "Torsk",  
  "Type": "Saltvannsfisk",  
  "Spisbarhet": "Veldig",  
}
```

Dette gjør dokument databaser veldig fleksibel i form av hva slags data det er man kan lagre og gir i tillegg muligheten til å lagre flere relaterte data sammen.

Fordeler

- Høy fleksibilitet i lagring av data og åpenhet for andre lagringer av data i fremtiden.
- Lagrer data på en veldig lesbar måte for mennesker.
- Stor fleksibilitet med bruk av JSON

Ulempor

- Ikke like strukturert som andre type databaser. Dette kommer av at det er en stor fleksibilitet om hva slags data som kan være lagret i dokumentene, videre så kan dette f. eks svekke referanse integritet i et større system.

Hvilke datasett passer?

Grunnet dokumentdatabaser sin måte å strukturere data på så passer flere datasett med denne type database motoren.

[Violent Crime rates by US State](#)" passer veldig bra sammen med dokumentdatabaser som MongoDB. Ettersom at et dokument er en enhet som inneholder flere nøkkel-verdi par så kan vi sette opp noe lignende som i "key-value".

Et eksempel på et dokument for dette datasettet kan for eksempel være:

```
{  
  ID : "Alabama",  
  Murder_arrests_pr100k: "13.2",  
  Assault_arrests_pr100k: "236",  
  Rape_arrests_pr100k: "58",  
  Ubranpop_arrests_pr100k: "21.2",  
}
```

I denne situasjonen så vil nøkkelen være → ("ID" : "Alabama") for dokumentet

Andre datasett som vi kan bruke i denne motoren vil være [US Estimated Crimes](#). Dette datasettet vil ha veldig godt nytte av en av funksjonene som eksisterer i MongoDB som heter "[Embedded documents](#)". Dette åpner muligheter for å ha dataen fra 1979-2019 innenfor samme dokument ved å hver av årene som et "embedded document" istedenfor separate dokumenter.

Et eksempel

```
{  
state_name: "Alabama"  
year: [  
{  
year: 1979  
population: "3769000",  
violent_crime: "15578",  
homicide: "496",  
rape_legacy: "1037",  
robbery: "4127",  
aggravated_assault: "9918"  
}  
  
{  
year: 1980  
population: "3861466",  
violent_crime: "17320",  
homicide: "509",  
rape_legacy: "1158",  
robbery: "5102",  
aggravated_assault: "10551"  
}  
]  
}
```

Ved å sette det opp som "embedded documents" så kutter man ned på kompleksiteten når man kjører queries på databasen.

Column

En kolonne database er hva navnet tilsier. Den lagrer data i kolonner istedenfor rader som er den tradisjonelle måten å gjøre det på.

I rad baserte relasjon databaser så er dataene lagret rad etter rad. F. eks:

```
1, Fisk, Torsk, Saltvannsfisk, Veldig; 2, Fisk, Laks, Saltvannsfisk, Veldig; 3,  
Fisk, Ørret, Ferskvannsfisk, Veldig;
```

I en kolonne basert database så er dataene lagret litt annerledes i form av:

1, 2, 3; Torsk, Laks, Ørret; Saltvannsfisk, Saltvannsfisk, Ferskvannsfisk;
Veldig, Veldig, Veldig

Selv om de er lagret på forskjellige måter så vil begge ende opp med å se sånn her ut:

ID	Art	Navn	Type	Spisbarhet
1	Fisk	Torsk	Saltvannsfisk	Veldig
2	Fisk	Laks	Saltvannsfisk	Veldig
3	Fisk	Ørret	Ferskvannsfisk	Veldig

Forskjellen i lagringen i data er at kolonne for kolonne blir lagret om gangen i databasen, mens i en tradisjonell rad basert system så er det rad for rad som er lagret.

Fordeler

- Visse typer querier are mye raskere. F. eks kjøre en aggregering av den gjennomsnittlige alderen i en kunde data base. Dette er fordi motoren kan direkte gå til der kolonnen er lagret og lese av hele kolonnen i et sett. I en tradisjonell rad struktur så ville det tatt mye lengre tid. Dette er fordi i en sånn data base så må det leses av rad for rad og legge til den riktige dataen i hver radi aggregeringen.

Ulempor

- Fordelen til denne type data base er også dens store ulempe. Dette i praksis betyr at enkelte queries ikke er godt egnet for kolonne data baser. Dette kan for eksempel være verdier i flere kolonner fra samme rad mange ganger i løpet av en dag. F. eks en kunde database hvor du ønsker å søke opp informasjon om en spesifikk kunde. For å få til dette så må motoren lese gjennom hver kolonne om gangen.

Hvilke datasett passer?

[US Estimated Crimes](#) er et datasett hvor kolonnebasert databasemotor vil hjelpe oss med å forstå mer av dataene som er i settet. På grunn av hvordan kolonnebaserte databaser leser av data så vil dette kunne hjelpe oss veldig bra i aggregeringer av data. F. eks hvor mye befolkningen til en stat har grodd i løpet av 1979-2019. Eller se etter ting som gjennomsnitt, medianer, kvartiler, avvik, osv. Til dette kan vi bruke f. eks Cassandra som vår utvalgte kolonnedbatabase fort gjøre disse aggregeringene.

Graph

En graf database lagrer data i form av noder og forhold istedenfor tabeller eller dokumenter. Dataene er lagret på en måte som kan minne om et tankekart.

Graf databaser har som formål å lagre og navigere forhold mellom flere noder. Mellom disse nodene så har du forhold(edge) som knytter disse to nodene. Selve forholdet er kan man argumentere for å være det viktigste i en graf database.

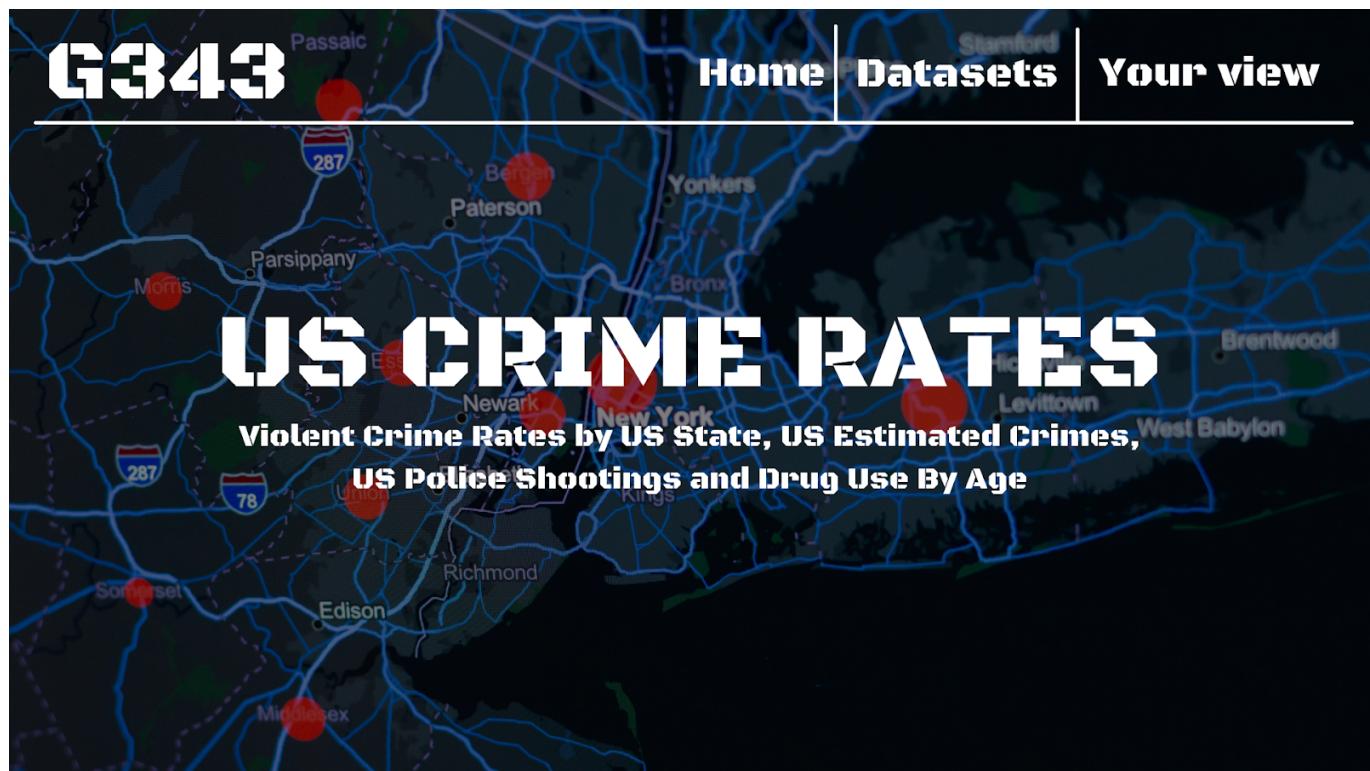
Nå til dags så er vi mer tilknyttet enn før via digitale nettverk. For å forstå mye av tingene som skjer der ute så er det nødvendig å nok koblinger(forhold) mellom disse 2 nodene.

Graf databaser bruker topografiske daa modeller til å lagre data. Disse databasene kobler

Hvilke datasett passer sammen med denne?

[US Police Shootings](#) er et datasett som passer veldig bra til dette. Dette er på grunn av at det kan være mange årsaker til hvorfor noe sånt her skjer og da ønsker man å se på koblingene mellom forskjellige skytinger.

Del 2



Første siden av siden vår, altså da "main page".

DATASETS



[Download CSV file](#)



[Download CSV file](#)



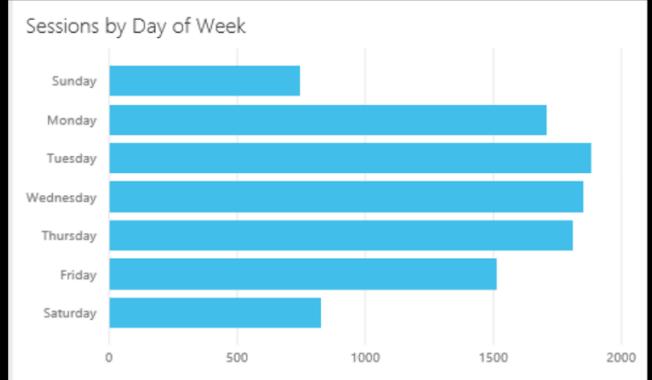
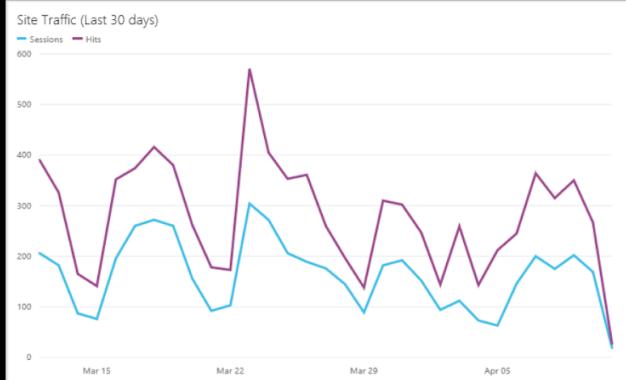
[Download CSV file](#)



[Download CSV file](#)

Siden for datasets, hvor du kan laste ned hver csv fil hvis du ønsker det.

Stats and graph for chosen CSV file



Her kan du se grafen og statistikken for valgt csv fil.

Milepål 2 - Design av nøkkel-verdidatabase

Oppgavetekst:

1. Design dataobjekter og aggregeringer

1. Beskriv prosessene som produserer ny data applikasjonen deres, både via nettsiden og fra andre kilder (slik som oppdatering ved kildene deres).
 2. Design dataobjekter og aggregeringer som dere trenger for nettsiden og velg mellom Riak og etcd. Her må dere tenke på både hvilken informasjon er det som faktisk skal vises og hvordan eksisterende data skal kunne oppdateres med ny, samt krav til konsistens/tilgjengelighet og ytelse.
 3. 1. Begrunn alle valgene for deres design mtp. konsistens, tilgjengelig, og ytelse.
 4. Beskriv hvordan eksisterende data i databasen vil bli oppdatert når dere mottar ny. Bruk en punktliste med detaljert beskrivelse av hvert steg for hver av input kildene deres.
 5. Diskuter om noen av valgene deres ville vært annerledes om dere var dataeier, istedenfor slik det er nå med lett tilgjengelig data fra Internett.
-

Svar:

Del 1

Oppgave A

Oppdatering av data er veldig avhengig av hvilken datasett det er som skal oppdateres og hva det er som skal oppdateres i disse datasettene. Videre så er det viktigere at våre datasett har høy nøyaktighet enn tilgjengelighet. Dette er fordi datasettene vi har valgt er av den typen som er viktig at dataene som blir vist fram er riktige

Som nevnt tidligere så avhenger det litt av hvilket datasett det er som skal oppdateres. Enkelte av datasettene kan bli oppdatert fortløpende når nye data kommer inn. Et

eksempel på dette kan være [US Police Shootings](#) som må muligens oppdateres fortløpende når hendelser av denne naturen oppstår

Andre datasett som [US Estimated Crimes](#) trengs bare å oppdateres når nye data blir publisert årlig av FBI(Federal Bureau of Investigation) i U.S.A i sin årlige rapport.

Videre så må enkelte type data punkter følge en forhåndsgodkjent liste. I "US Police Shootings" så kan dette for eksempel gjelde:

- Gender of person
- Location (City) of event
- Location (State) of event
- race of shot person

Andre former for data trenger ikke å følge forhåndsgodkjente lister. Disse kan f. eks være:

- name
- manner_of_death
- armed

Filbehandling:

Vi har gjort et valg at brukeren ikke skal ha tilgang til å legge til eller slette dataene som er i applikasjonen.

Oppgave B

For vår applikasjon så har vi valgt etcd. Dette er grunnet hva slags data det er vi har valgt å jobbe med. Eksempel på dette er at i dataene våres så har vi navnet til folk som har blitt skutt eller skadet. etcd garanterer atomisk konsistens og passer bedre dataene vi jobber med. Riak derimot er designet for høy tilgjengelighet, men i gjengjeld så ofrer den konsistens. I tillegg så passer den bedre med clusters av maskiner.

Våre datasett er enkle, ikke store og vi vil derfor ikke oppleve at ytelsen i etcd kommer til å være et problem.

Eksempel på et dataobjekt

[DATAOBJEKT BILDET]

Del 2

Oppgave A

Det er mange måter vi kan legge til nye data i databasen på:

- Legge til ny csv fil/oppdatere innlagt csv fil
- Oppdatere dataene på selve siden
- Ta i bruk APlet til siden

Når vi legger til denne nye dataen skal databasen oppdatere seg automatisk. Det skal også sjekkes om ny data, ikke er 100% lik den dataen som allerede ligger inne. For eks. I [US Police Shootings](#) ser vi en oversikt over personer som er drept av politiet, under en arrestasjon. Her ligger det informasjon om enkeltindivider, og dødsårsak. Hvis en ønsker å legge til ny informasjon her, må databasen sjekke at det ikke blir registrert samme person flere ganger. Samt må vi følge strukturen til databasen vi skal legge den oppdaterte/nye dataen i.

Oppgave B:

Om vi hadde vært eier av dataene så hadde vi ikke gjort noe annerledes. Dataene som vi har tilgang til er oversiktig og simple, samt lesbare for brukeren. Dette gjør dem ideelt for applikasjonen vi utvikler.

Milepål 3 - Implementasjon av nøkkel-verdi databasen og design av dokumentdatabasen

Oppgavetekst:

1. Implementer nøkkel-verdi databasen

Merk: Selv om dere valgte Riak i Milepål 2, skal dere implementere mot etcd.

1. Implementer kode for å laste inn data i kv-databasen dere designet i Milepål 2 fra datakildene deres slik dere planla i Milepål 2.
2. Implementer kode for å hente ut aggregeringene og dataobjektene dere trenger for visning i nettsiden fra kv-databasen og skriv de til konsollet.
3. Implementer et program som simulerer at en bruker legger inn ny data via skjema på nettsiden ved at en taster inn verdier i konsollet til Scala programmet dere skriver.

4. Bonus: Implementer en prosedyre som kan oppdatere databasen ved utgivelse av ny data fra kilden på en måte som er helst er mer effektiv enn å bare slette og sette inn data på nytt.

2. Design av dokumentdatabasen

1. Design aggregeringer og dataobjekter for de elementene deres av nettsiden som er basert på data fra datasettet dere har tiltenkt dokumentdatabasen. Legg til passende funksjonalitet i nettsiden om dere ikke har denne funksjonaliteten fra før. Begrunn alle valgene mtp. konsistens og ytelse. (Husk at en dokumentdatabase har mange flere muligheter for spørninger enn en kv-database.)
 2. Beskriv hvordan eksisterende data i databasen vil bli oppdatert når dere mottar ny via skjema på nettsiden og ved mottak av oppdatert data fra kilden. Bruk en punktliste med detaljert beskrivelse av hvert steg for hver av input kildene deres.
-

Svar:

Del 1

A, B, C)

Denne koden laster inn data i key value databasen. Samt så har den et par aggregeringer og mulighet til å lese av databasen ettersom hva slags input det er.

```
case class NodeResponse(key : String, value: String)
case class Node(key : String, value: String, nodes: Array[NodeResponse])
case class Message(action: String, node: Node)
case class stateData( murder: Double, assault: Double, urbanpop: Double,
rape: Double )

object HttpGetPostTest extends App {

  def retrieveKeyValue (key: String): Unit = {

    val url = "http://127.0.0.1:2379/v2/keys/" + key

    val result = scala.io.Source.fromURL(url).mkString

    println(result);

    val messageParsed = new Gson().fromJson( result, classOf[Message] );
```

```
val valueParsed = new Gson().fromJson(
messageParsed.node.nodes(0).value, classOf[stateData] );

println(valueParsed);

println("State: " + key);

println("Murder: " + valueParsed.murder);

println("Assault: " + valueParsed.assault);

println("Urban Pop: " + valueParsed.urbanpop);

println("Rape: " + valueParsed.rape);

}

val file = "US_violent_crime.csv"

val source = io.Source.fromFile(file)

//// Lesing og pushing av key values i KV store

source.getLines.drop(1).foreach {line =>

val row = line.split(",(?=(\"|\"")|=\"$")").map(_.trim)

// Sletting av gåseøyne

val country_0 = row(0).toString.replaceAll("""", "")

val country = country0.toString.replaceAll(" ", "")

println(country);

println("Inserting{ state: " + country + ", murder: " + row(1).toDouble + ", assault: " + row(2).toDouble + ", urbanPop: " + row(3).toDouble + ", rape: " + row(4).toDouble )

val stateObject = new stateData(row(1).toDouble, row(2).toDouble, row(3).toDouble, row(4).toDouble)

// val key = row(0).toString

val stateObject_asJson = new Gson().toJson(stateObject)

println(stateObject_asJson)
```

```

val url = "http://127.0.0.1:2379/v2/keys/" + country

// add name value pairs to a post object

val post = new HttpPost(url)

val nameValuePairs = new ArrayListNameValuePair

nameValuePairs.add(new BasicNameValuePair("value",
stateObject_asJson))

post.setEntity(new UrlEncodedFormEntity(nameValuePairs))

// send the post request

val client = new DefaultHttpClient

val response = client.execute(post)

println("--- HEADERS ---")

response.getAllHeaders.foreach(arg => println(arg))

}

// Eksempel på uthenting

retrieveKeyValue("New_Hampshire")

retrieveKeyValue("Nebraska")

}

```

Del 2

A)

Vi bruker denne for å lage colelctione våres

```

db.createCollection("murder_arrests_per_100k"),
db.createCollection("assault_arrests_per_100k"),
db.createCollection("rape_arrests_per_100k")

```

Et eksempel på en mengde med dataobjekter:

```
db.murder_arrest_per_100k.insertmany([
{states: "Delaware", arrests: 5.9},
{states: "Pennsylvania", arrests: 6.3},
{states: "Rhose Island", arrests: 3.4},
{states: "Utah", arrests: 3.2}
]);
```

```
db.assault_arrest_per_100k.insertmany([
{states: "Delaware", arrests: 238},
{states: "Pennsylvania", arrests: 106},
{states: "Rhose Island", arrests: 174},
{states: "Utah", arrests: 201}
]);
```

```
db.rape_arrest_per_100k.insertmany([
{states: "Delaware", arrests: 15.7},
{states: "Pennsylvania", arrests: 14.9},
{states: "Rhose Island", arrests: 8.3},
{states: "Utah", arrests: 22.9}
]);
```

B)

Vi har ingen input kilder, på vår nettside. Grunnen er fordi datasettene er årsbaserte, og informasjonen blir lagt inn hvert år av FBI (Federal Bureau of Investigation).

Utenforstående (altså brukere) kan kun se informasjonen på siden, men ikke oppdatere den.

Ett av datasettene ([US Police Shootings](#)) inneholder personinformasjon, noe kun FBI har tilgang til i denne sammenhengen. De andre datasettene (også [US Police Shootings](#)), er stats basert. Hver stat har en statistikk på kriminalitets antallet, en samlet informasjonskilde som viser både FBI og brukere hvor “kriminell/voldelig” en stat er. Samt viser denne statistikken hvor mange av de som begår noe kriminelt, faktisk blir tatt av politiet og dømt for sine handlinger.

Ikke alle av hendelsene som skjedde i hver stat, blir dokumentert. Antallet viser kun det som er blitt “fanget” av politiet og/eller kriminal organisasjoner som FBI og CIA. Det viser også hvor bra politidistrikt som var i området, samt hvor mange hendelser politiet klare å stoppe.

Milepål 4 - Implementasjon av dokumentdatabase, design av kolonnefamilie og grafdatabase

Oppgavetekst:

1. Implementer dokumentdatabasen

1. Implementer kode for å laste inn data i mongodb-databasen dere designet i Milepål 3 fra datakildene deres slik dere planla i Milepål 3.
2. Implementer kode for å kjøre spørninger dere trenger mot mongodb-databasen for visning i nettsiden fra databasen og skriv de til konsollet.
3. Implementer et program som simulerer at en bruker legger inn ny data via skjema på nettsiden ved at en taster inn verdier i konsollet til Scala programmet dere skriver.

2. Design av kolonnefamiliedatabasen

1. Design aggregeringer og dataobjekter for de elementene deres av nettsiden som er basert på data fra datasettet dere har tiltenkt kolonnefamiliedatabasen. Legg til passende funksjonalitet i nettsiden om dere ikke har denne funksjonaliteten fra før. Begrunn alle valgene mtp. konsistens og ytelse.
2. Skriv spørninger for opprettelse av databasen, hente ut data, og sette inn data.
3. Beskriv hvordan eksisterende data i databasen vil bli oppdatert når dere mottar ny via skjema på nettsiden og ved mottak av oppdatert data fra kilden.

3. Design av grafdatabasen

1. Design noder og relasjonene som dere trenger for å kunne kjøre spørninger for de elementene deres av nettsiden som er basert på data fra datasettet dere har tiltenkt grafdatabasen. Legg til passende funksjonalitet i nettsiden om dere ikke har denne funksjonaliteten fra før.
2. Skriv spørninger for å hente ut data og sette inn data.
3. Beskriv hvordan eksisterende data i databasen vil bli oppdatert når dere mottar ny via skjema på nettsiden og ved mottak av oppdatert data fra kilden.

Svar:

Milepæl 4

Del 1

A)

Koden

```
object postMongoObj extends App {  
  
    implicit class DocumentObservable[C](val observable:  
    Observable[Document]) extends ImplicitObservable[Document] {  
        override val converter: (Document) ⇒ String = (doc) ⇒ doc.toJson  
    }  
  
    implicit class GenericObservable[C](val observable: Observable[C])  
    extends ImplicitObservable[C] {  
        override val converter: (C) ⇒ String = (doc) ⇒ doc.toString  
    }  
  
    trait ImplicitObservable[C] {  
        val observable: Observable[C]  
        val converter: (C) ⇒ String  
  
        def results(): Seq[C] = Await.result(observable.toFuture(), Duration(10,  
        TimeUnit.SECONDS))  
        def headResult() = Await.result(observable.head(), Duration(10,  
        TimeUnit.SECONDS))  
        def printResults(initial: String = ""): Unit = {  
            if (initial.length > 0) print(initial)  
            results().foreach(res ⇒ println(converter(res)))  
        }  
        def printHeadResult(initial: String = ""): Unit = println(s"initial  
        {converter(headResult())}")  
    }  
  
    val mongoClient: MongoClient = MongoClient();  
    val database: MongoDatabase = mongoClient.getDatabase("uscrime");
```

```

val collection: MongoCollection[Document] =
database.getCollection("arrest_per_100k");

val document = Document(
"State" → "Texas",
"Murder" → 5.9,
"Assault" → 4.9,
"Rape" → 3.4,
"UrbanPop" → 6.9

);

collection.insertOne(document).results();

collection.find().printResults();
}

```

I koden ovenfor så implementeres kode for å laste data inn i mongoDB databasen. Dette gjøres via et objekt som leser inn dokumenter inn i databasen.

Dette objektet henter mongo klienten og velger en spesifikk database, om den ikke eksisterer så vil den lage denne databasen. Videre så velger den en collection hvor dokumentet i slutten av objektet vil bli skrevet inn i ved hjelp av ".insertOne()". Hadde det vært flere dokumenter som skulle bli skrevet inn så hadde det istedet vært en ".insertMany()".

C)

```

object insertMongoObj extends App {
implicit class DocumentObservable[C](val observable:
Observable[Document]) extends ImplicitObservable[Document] {
override val converter: (Document) ⇒ String = (doc) ⇒ doc.toJson
}

implicit class GenericObservable[C](val observable: Observable[C])
extends ImplicitObservable[C] {
override val converter: (C) ⇒ String = (doc) ⇒ doc.toString
}

trait ImplicitObservable[C] {
val observable: Observable[C]
val converter: (C) ⇒ String
}

```

```

def results(): Seq[C] = Await.result(observable.toFuture(), Duration(10,
TimeUnit.SECONDS))
def headResult() = Await.result(observable.head(), Duration(10,
TimeUnit.SECONDS))
def printResults(initial: String = ""): Unit = {
if (initial.length > 0) print(initial)
results().foreach(res => println(converter(res)))
}
def printHeadResult(initial: String = ""): Unit = println(s"$initial
{converter(headResult())}")
}

val mongoClient: MongoClient = MongoClient();
val database: MongoDatabase = mongoClient.getDatabase("uscrime");
val collection: MongoCollection[Document] =
database.getCollection("arrest_per_100k");

val source = io.Source.fromFile("US_violent_crime.csv")

source.getLines.drop(1).foreach { line =>
val row = line.split(",(?=(\"|'')|\"$)").map(_.trim)
println(row.mkString(","))

def parseDouble(value : String) : Option[Double] = if (value == "") None else
Some(value.toDouble)

println("Success")

val document = Document(
"State" -> row(0),
"Murder" -> parseDouble(row(1)),
"Assault" -> parseDouble(row(2)),
"Urbanpop" -> parseDouble(row(3)),
"Rape" -> parseDouble(row(4))
);

collection.insertOne(document).results();
}

println("All documents inserted")
}

```

C)

I vår side så hadde vi ikke planer om gi brukere tillatelse til å modifisere dataene i webapplikasjonen. Derfor har vi ikke implementert en slik funksjon i vår kode, men vi gjort det sånn her.

Pseudokode:

```
Objekt{  
    // Tomme variabler som fylles gjennom at brukeren taster  
    var state = ""  
    var murder = ""  
    var assault = ""  
    var rape = ""  
  
    while loop() {  
        println("Please type state name")  
        var state = readLine("")  
        println("Murder rate:")  
        var murder = readline("")  
        println("assault:")  
        var assault = readline("")  
        println("rape:")  
        var rape = readline("")}  
  
    val document = Document(  
        "State" → row(0),  
        "Murder" → parseDouble(row(1)),  
        "Assault" → parseDouble(row(2)),  
        "Urbanpop" → parseDouble(row(3)),  
        "Rape" → parseDouble(row(4))  
    );  
  
    collection.insertOne(document).results();  
    println("Your submission has been accepted")  
}
```

Del 2

A)

```
Murder_assault_rape_in_1973_in_US:{
```

```
"Hawaii" = {  
    "murder": 5.3,  
    "assault": 46,  
    "rape": 20.2  
},  
  
"Alaska" = {  
    "murder": 10,  
    "assault": 263,  
    "rape": 44.5  
},  
  
"Nevadai" = {  
    "murder": 12.2,  
    "assault": 252,  
    "rape": 46  
},  
  
"Minnesota" = {  
    "murder": 2.7,  
    "assault": 72,  
    "rape": 14.9  
},  
  
"California" = {  
    "murder": 9,  
    "assault": 276,  
    "rape": 40.6  
},}
```

Her har vi valgt å bruke [Violent Crime Rates by US State](#). For å designe aggregeringer og dataobjekter. Vi har videre implementert dette i Cassandra:

```
/* Creating a workspace */
CREATE KEYSPACE uscrimedb WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};

use uscrimedb;

/* Creating tables */
CREATE TABLE USstate1973(
id TEXT PRIMARY KEY,
name TEXT,
murder INT,
assault INT,
rape INT
);

CREATE TABLE crime(
id TEXT,
typeOfCrime TEXT,
PRIMARY KEY(id)
);

/* Inserting info into the tables */
INSERT INTO USstate1973 (id, name, murder, assault, rape)
VALUES (
    'Hawaii',
    'Hawaii',
    5.3,
    46,
    20.2
)
USING TTL 86400;
```

B)

Opprettelse av databasen:

```
CREATE KEYSPACE uscrime
```

Eksempel på å lage en table:

```
CREATE TABLE uscrime(
state text,
murder double,
assault int,
rape double
PRIMARY KEY (state))
```

Eksempel på å sette inn data:

```
INSERT uscrime (state, murder, assault, rape)
```

```
VALUES (California, 9, 276, 40.6)
```

Eksempel på å hente ut stater hvor "murder" er >7:

```
SELECT * FROM uscrime WHERE murder >7
```

C)

NULL

Del 3

A)

```
// 1 - Connecticut
CREATE (Connecticut:State {murder:3.3, assault:110, rape:11.1})

// 2 - Idaho
CREATE (Idaho:State {murder:2.6, assault:120, rape:14.2})

// 3 - Maryland
CREATE (Maryland:State {murder:11.3, assault:300, rape:27.8})

// 4 - New Hampshire
CREATE (NewHampshire:State {murder:2.1, assault:57, rape:9.5})

// 5 - North Carolina
CREATE (NorthCarolina:State {murder:13, assault:337, rape:16.1})

// 6 - Oklahoma
CREATE (Oklahoma:State {murder:6.6, assault:151, rape:20})

// 7 - Oregon
CREATE (Oregon:State {murder:4.9, assault:159, rape:29.3})

// 8 - Tennessee
CREATE (Tennessee:State {murder:13.2, assault:188, rape:26.9})

// 9 - Wisconsin
CREATE (Wisconsin:State {murder:2.6, assault:53, rape:10.8})

// 10 - Wyoming
CREATE (Wyoming:State {murder:6.8, assault:161, rape:15.6})
```

```
// All the states into neo4j format
CREATE
  (Connecticut) - [:ARRESTS] -> (State),
  (Idaho) - [:ARRESTS] -> (State),
  (Maryland) - [:ARRESTS] -> (State),
  (NewHampshire) - [:ARRESTS] -> (State),
  (NorthCarolina) - [:ARRESTS] -> (State),
  (Oklahoma) - [:ARRESTS] -> (State),
  (Oregon) - [:ARRESTS] -> (State)
  (Tennessee) - [:ARRESTS] -> (State),
  (Wisconsin) - [:ARRESTS] -> (State),
  (Wyoming) - [:ARRESTS] -> (State)
;
```

B)

Query hvor det matches de skytingene hvor en person bor i en by i staten Iowa

```
MATCH
  (p:name)-[:LIVES_IN]->(c:Burlington)-[:CITY_IN]->(s:Iowa)<[:SHOT_IN]->(p)
RETURN
  p1
```

Legge inn data:

```
CREATE (:Person { name: "Autumn Steele"}) - [:LIVES_IN] -> (:city {name: Burlington})
```

C)

NULL

Milepæl 5 - Spark og HDFS

Oppgavetekst:

1. **Spark it up**

1. Skriv Spark-programmer for å laste inn csv-filene deres, før å så skrive de til disk igjen som PARQUET filer. Dere kan, om det er praktisk, gjøre eventuelle tilpassninger av dataframes før skriving til PARQUET om dere ønsker.
2. Skriv Spark-programmer som leser PARQUET filene dere lagde i (A) og bruk de til å produsere alle aggregeringer, og dataobjekter dere trenger for alle 4-databasemotorene. Spark-programmet skal skrive aggregeringene og dataobjektene til .csv filer. For eksempel. Hvis dere har en Cassandra Insert 'INSERT INTO cycling.cyclist_name (id, lastname, firstname) VALUES (...)'. Skal dere produsere en .csv-fil som har kolonnene {id, lastname, firstname}. (Merk: Dere trenger ikke å utføre spørringene mot databasemotorene fra Spark, bare lage csv. filene som beskrevet)
3. Legg til litt mer funksjonalitet i nettsiden som viser dybden i hva dere får til med Spark på deres datasett.

2. **HDFS**

1. Skriv kommandoer for å overføre PARQUET filene deres til HDFS.
 2. Modifisert Spark-programmene deres fra (1.B) til å lese fra HDFS istedenfor lokalt filsystem.
 3. Beskriv nøyne hva som skjer bak kulissene når dere kjører programmene deres fra (2.B). Det holder med nøyne beskrivelse av et program. Velg det programmet som gjør dere istic til å skrive en diskusjon som viser hvor godt dere forstår hvordan HDFS fungerer, og hvordan Spark og HDFS jobber sammen.
-

Svar:

Del 1 - Spark it up

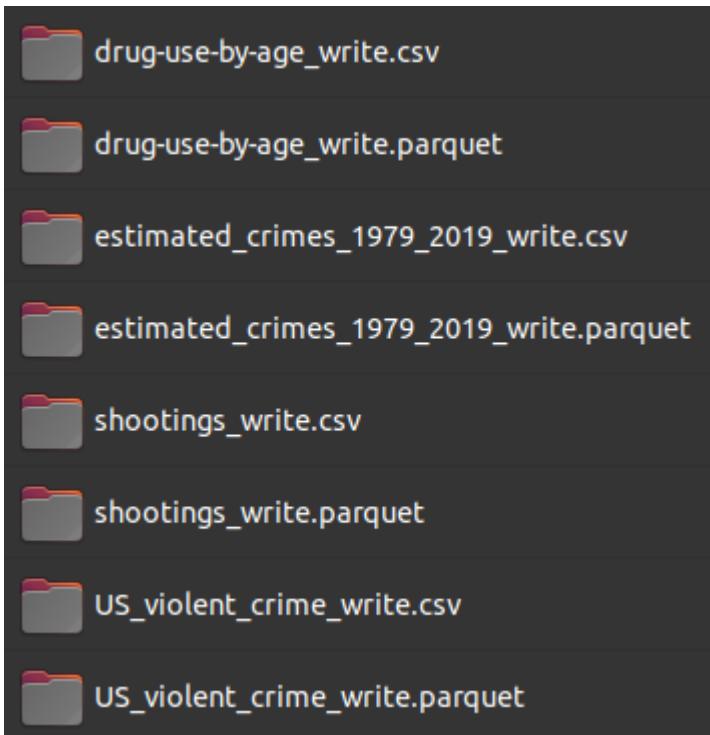
Oppgave A

Oppgavetekst:

Skriv Spark-programmer for å laste inn csv-filene deres, før å så skrive de til disk igjen som PARQUET filer. Dere kan, om det er praktisk, gjøre eventuelle tilpassninger av dataframes før skriving til PARQUET om dere ønsker.

Svar:

Lagt csv-filene om til PARQUET filer.



Oppgave B

Oppgavetekst:

Skriv Spark-programmer som leser PARQUET filene dere lagde i (A) og bruk de til å produsere alle aggregeringer, og dataobjekter dere trenger for alle 4-databasemotorene. Spark-programmet skal skrive aggregeringene og dataobjektene til .csv filer. For eksempel. Hvis dere har en Cassandra Insert 'INSERT INTO cycling.cyclist_name (id, lastname, firstname) VALUES (...)''. Skal dere produsere en .csv-fil som har kolonnene {id, lastname, firstname}. (Merk: Dere trenger ikke å utføre spørringene mot databasemotorene fra Spark, bare lage csv. filene som beskrevet)

Svar:

[Violent Crime Rates by US State](#) - Lese PARQUET format

```
spark.read.format("parquet").load("US_violent_crime_write.parquet").show()
```

State	Murder	Assault	UrbanPop	Rape
Alabama	13.2	236	58	21.2
Alaska	10.0	263	48	44.5
Arizona	8.1	294	80	31.0
Arkansas	8.8	190	50	19.5
California	9.0	276	91	40.6
Colorado	7.9	204	78	38.7
Connecticut	3.3	110	77	11.1
Delaware	5.9	238	72	15.8
Florida	15.4	335	80	31.9
Georgia	17.4	211	60	25.8
Hawaii	5.3	46	83	20.2
Idaho	2.6	120	54	14.2
Illinois	10.4	249	83	24.0
Indiana	7.2	113	65	21.0
Iowa	2.2	56	57	11.3
Kansas	6.0	115	66	18.0
Kentucky	9.7	109	52	16.3
Louisiana	15.4	249	66	22.2
Maine	2.1	83	51	7.8
Maryland	11.3	300	67	27.8

US Police Shootings - Lese PARQUET format

```
spark.read.format("parquet").load("shootings_write.parquet").show()
```

id	name	date	manner_of_death	armed	age	gender	race	city	state	signs_of_mental_illness	threat_level	flee	body_camera	arms_category
3	Tim Elliot	2015-01-02	shot	gun 53.0	M	Asian	Shelton	WA		true	attack Not fleeing	false		Guns
4	Lewis Lee Lemcke	2015-01-02	shot	gun 47.0	M	White	Aloha	OR		false	attack Not fleeing	false		Guns
5	John Paul Quintero	2015-01-03	shot and Tasered	unarmed	23.0	M Hispanic	Wichita	KS		false	other Not fleeing	false		Unarmed
8	Matthew Hoffman	2015-01-04	shot	toy weapon 32.0	M	White	San Francisco	CA		true	attack Not fleeing	false	Other unusual obj...	
9	Michael Rodriguez	2015-01-04	shot	nail gun 39.0	M Hispanic		Evans	CO		false	attack Not fleeing	false	Piercing objects	
11	Kenneth Joe Brown	2015-01-04	shot	gun 18.0	M	White	Guthrie	OK		false	attack Not fleeing	false		Guns
13	Kenneth Arnold Buck	2015-01-05	shot	gun 22.0	M Hispanic		Chandler	AZ		false	attack Car	false		Guns
15	Brock Nichols	2015-01-06	shot	gun 35.0	M	White	Assaria	KS		false	attack Not fleeing	false		Guns
16	Autumn Steele	2015-01-06	shot	unarmed 34.0	F	White	Burlington	IA		false	other Not fleeing	true		Unarmed
17	Leslie Sapp III	2015-01-06	shot	toy weapon 47.0	M	Black	Knoxville	PA		false	attack Not fleeing	false	Other unusual obj...	
19	Patrick Wetter	2015-01-06	shot and Tasered	knife 25.0	M	White	Stockton	CA		false	attack Not fleeing	false	Sharp objects	
21	Ron Sneed	2015-01-07	shot	gun 31.0	M	Black	Freeport	TX		false	attack Not fleeing	false		Guns
22	Hashim Hanif Ibn ...	2015-01-07	shot	knife 41.0	M	Black	Columbus	OH		true	other Not fleeing	false	Sharp objects	
25	Nicholas Ryan Bri...	2015-01-07	shot	gun 30.0	M	White	Des Moines	IA		false	attack Car	false		Guns
27	Omar Julian Maxt...	2015-01-07	shot	gun 37.0	M	Black	New Orleans	LA		false	attack Foot	true		Guns
29	Loren Simpson	2015-01-08	shot	unknown 28.0	M	White	Huntley	MT		false	undetermined Not fleeing	false		Unknown
32	James Dudley Barker	2015-01-08	shot	shovel 42.0	M	White	Salt Lake City	UT		false	attack Not fleeing	true	Blunt instruments	
36	Artago Damon Howard	2015-01-08	shot	unarmed 36.0	M	Black	Strong	AR		false	attack Not fleeing	false	Unarmed	
37	Thomas Hamby	2015-01-08	shot	gun 49.0	M	White	Syracuse	UT		false	attack Not fleeing	true		Guns
38	Jimmy Foreman	2015-01-09	shot	gun 71.0	M	White	England	AR		false	attack Not fleeing	false		Guns

US Estimated Crimes - Lese PARQUET format

```
spark.read.format("parquet").load("estimated_crimes_1979_2019_write.parquet").show()
```

year	state_abbr	state_name	population	violent_crime	homicide	rape_legacy	rape_revised	robbery	aggravated_assault	property_crime	burglary	larceny	motor_vehicle_theft	cavates
1979	null	null	220099000	1208030	21460	76390	null	480700	629480	11041500	3327700	6601000	1112800	null
1979	AK	Alaska	406000	1994	54	292	null	445	1203	23193	5616	15076	2501	null
1979	AL	Alabama	3769000	15578	496	1037	null	4127	9918	144372	48517	83791	12064	null
1979	AR	Arkansas	2180000	7984	198	595	null	1626	5565	70949	21457	45267	4225	null
1979	AZ	Arizona	2450000	14528	219	1120	null	4305	8884	177977	48916	116976	12085	null
1979	CA	California	22696000	184087	2952	12239	null	75767	93129	1511621	496310	847148	167563	null
1979	CO	Colorado	2772000	14472	161	1472	null	4353	8486	189984	49741	117898	13345	null
1979	CT	Connecticut	3115000	12902	131	752	null	6021	5998	167131	48229	96997	21905	null
1979	DC	District of Columbia	656000	10553	180	489	null	6920	2964	45877	13452	28819	3606	null
1979	DE	Delaware	582000	3127	33	162	null	753	2179	34853	8899	23881	2882	null
1979	FL	Florida	8860000	73881	1084	4576	null	22897	46124	607281	198884	378999	38298	null
1979	GA	Georgia	5118000	28594	877	2216	null	10939	14562	248641	81579	145758	21304	null
1979	HI	Hawaii	915000	2651	66	296	null	1688	601	63664	16538	40580	6546	null
1979	IA	Iowa	2993000	5259	65	320	null	1457	3417	119620	26768	85023	7829	null
1979	ID	Idaho	955000	2613	49	186	null	392	1986	35766	9729	23577	2460	null
1979	IL	Illinois	11230000	83540	1203	3702	null	36056	42579	593750	161776	356062	75912	null
1979	IN	Indiana	5400000	18254	448	1681	null	7167	8958	230223	63176	143666	23381	null
1979	KS	Kansas	2369000	8376	130	626	null	2423	5197	107605	31504	69622	6479	null
1979	KY	Kentucky	3527000	8748	335	719	null	3247	4447	103548	32082	62431	9035	null
1979	LA	Louisiana	4026000	27229	682	1554	null	8832	16161	188514	56237	115856	16421	null

Drug Use By Age - Lese PARQUET format

```
spark.read.format("parquet").load("drug-use-by-age_write.parquet").show()
```

age	n	alcohol-use	alcohol-frequency	marijuana-use	marijuana-frequency	cocaine-use	cocaine-frequency	crack-use	crack-frequency	heroin-use	heroin-frequency	hallucinogen-use	hallucinogen-frequency	inhalant-use	inhalant-frequency
12 [2798]	3.9	3.0	1.1	4.8	0.1	5.8	0.0	-	0.1	35.5	0.2	52.0	1.6	19.0	
13 [2575]	8.5	6.0	3.4	15.0	0.1	1.0	0.0	3.0	0.0	-	0.6	6.0	2.5	12.0	
14 [2792]	18.1	5.0	8.7	24.0	0.1	5.5	0.0	-	0.1	2.0	1.6	3.0	2.6	5.0	
15 [2956]	29.2	6.0	14.3	25.0	0.1	4.8	0.1	9.5	0.2	1.0	2.1	4.0	2.5	5.5	
16 [3058]	40.1	10.0	22.5	30.0	1.0	7.0	0.0	1.0	0.1	66.5	3.4	3.0	5.0	3.0	
17 [3038]	49.3	13.0	28.0	36.0	2.0	5.8	0.1	21.0	0.1	64.0	4.8	5.0	2.0	4.0	
18 [2469]	58.7	24.0	33.7	52.0	3.2	5.8	0.4	10.0	0.4	46.0	7.0	4.0	1.8	4.0	
19 [2223]	64.6	36.0	33.4	60.0	4.1	5.5	0.5	2.0	0.5	180.0	8.6	3.0	1.4	3.0	
20 [2271]	69.7	48.0	34.0	60.0	4.9	8.0	0.6	5.0	0.9	45.0	7.4	2.0	1.5	4.0	
21 [2354]	83.2	52.0	33.0	52.0	4.8	5.8	0.5	17.0	0.6	30.0	6.3	4.0	1.4	2.0	
22-23 [4707]	84.2	52.0	28.4	52.0	4.5	5.8	0.5	5.0	1.1	57.5	5.2	3.0	1.0	4.0	
24-25 [4591]	83.1	52.0	24.9	60.0	4.0	6.0	0.5	6.0	0.7	88.0	4.5	2.0	0.8	2.0	
26-29 [2628]	80.7	52.0	26.8	52.0	3.2	5.8	0.4	6.0	0.6	50.0	3.2	3.0	0.6	4.0	
30-34 [2864]	77.5	52.0	16.4	72.0	2.1	8.0	0.5	15.0	0.4	66.0	1.8	2.0	0.4	3.5	
35-49 [3911]	75.0	52.0	10.4	40.0	1.5	15.0	0.5	48.0	0.1	280.0	8.6	3.0	0.3	10.0	
50-64 [1533]	63.2	52.0	7.3	52.0	0.9	36.0	0.4	62.0	0.1	41.0	3.3	4.0	0.2	13.5	
65+ [2448]	49.3	52.0	1.2	36.0	0.0	-	0.0	-1	0.0	120.0	0.1	2.0	0.0	0.0	

pain-releiver-use	pain-releiver-frequency	oxycontin-use	oxycontin-frequency	tranquilizer-use	tranquilizer-frequency	stimulant-use	stimulant-frequency	meth-use	meth-frequency	sedative-use	sedative-frequency
2.0	36.0	0.1	24.5	0.2	52.0	0.2	2.0	0.0	-	0.2	13.0
2.4	14.0	0.1	41.0	0.3	25.5	0.3	4.0	0.1	5.0	0.1	10.0
3.9	12.0	0.4	4.5	0.9	5.8	0.8	12.0	0.1	24.0	0.2	16.5
5.5	10.0	0.8	3.0	2.0	4.5	1.5	6.0	0.3	10.5	0.4	30.0
6.2	7.0	1.1	4.0	2.4	11.0	1.8	9.5	0.3	36.0	0.2	3.0
8.5	9.0	1.4	6.0	3.5	7.0	2.8	9.0	0.6	48.0	0.5	6.5
9.2	12.0	1.7	7.0	4.9	12.0	3.0	8.0	0.5	12.0	0.4	10.0
9.4	12.0	1.5	7.5	4.2	4.5	3.3	6.0	0.4	105.0	0.3	6.0
10.0	10.0	1.7	12.0	5.4	10.0	4.0	12.0	0.9	12.0	0.5	4.0
9.8	15.0	1.3	13.5	3.9	7.0	4.1	10.0	0.6	2.0	0.3	9.0
10.0	15.0	1.7	17.5	4.4	12.0	3.6	10.0	0.6	46.0	0.2	52.0
9.0	15.0	1.3	20.0	4.3	10.0	2.6	10.0	0.7	21.0	0.2	17.5
8.3	13.0	1.2	13.5	4.2	10.0	2.3	7.0	0.6	30.0	0.4	4.0
5.9	22.0	0.9	46.0	3.6	8.0	1.4	12.0	0.4	54.0	0.4	10.0
4.2	12.0	0.3	12.0	1.9	6.0	0.6	24.0	0.2	104.0	0.3	10.0
2.5	12.0	0.4	5.0	1.4	10.0	0.3	24.0	0.2	30.0	0.2	104.0
0.6	24.0	0.0	-	0.2	5.0	0.0	364.0	0.0	-	0.0	15.0

Aggregeringer og dataobjekter:

```

import org.apache.spark.sql.{SaveMode, SparkSession}

import org.apache.spark.sql.functions._

object aggregeringer extends App {

    val spark: SparkSession = SparkSession.builder()
        .master("local[1]")
        .appName("gruppe343")

    .getOrCreate()

    spark.sparkContext.setLogLevel("ERROR 404")

    val dataframe = spark.read.format("parquet").load("")

    val =
        dataframe.sort(col("").desc).limit(10).write.format("csv").option("header",
    true).save("parquet/")

    // Shootings done by police

    val shootingsByPolice =
        dataframe.sort(col("").desc).limit(10).write.format("csv").option("header",
    true).save("aggCsvFiles/shootings.csv")

    // US violent crime rates
  
```

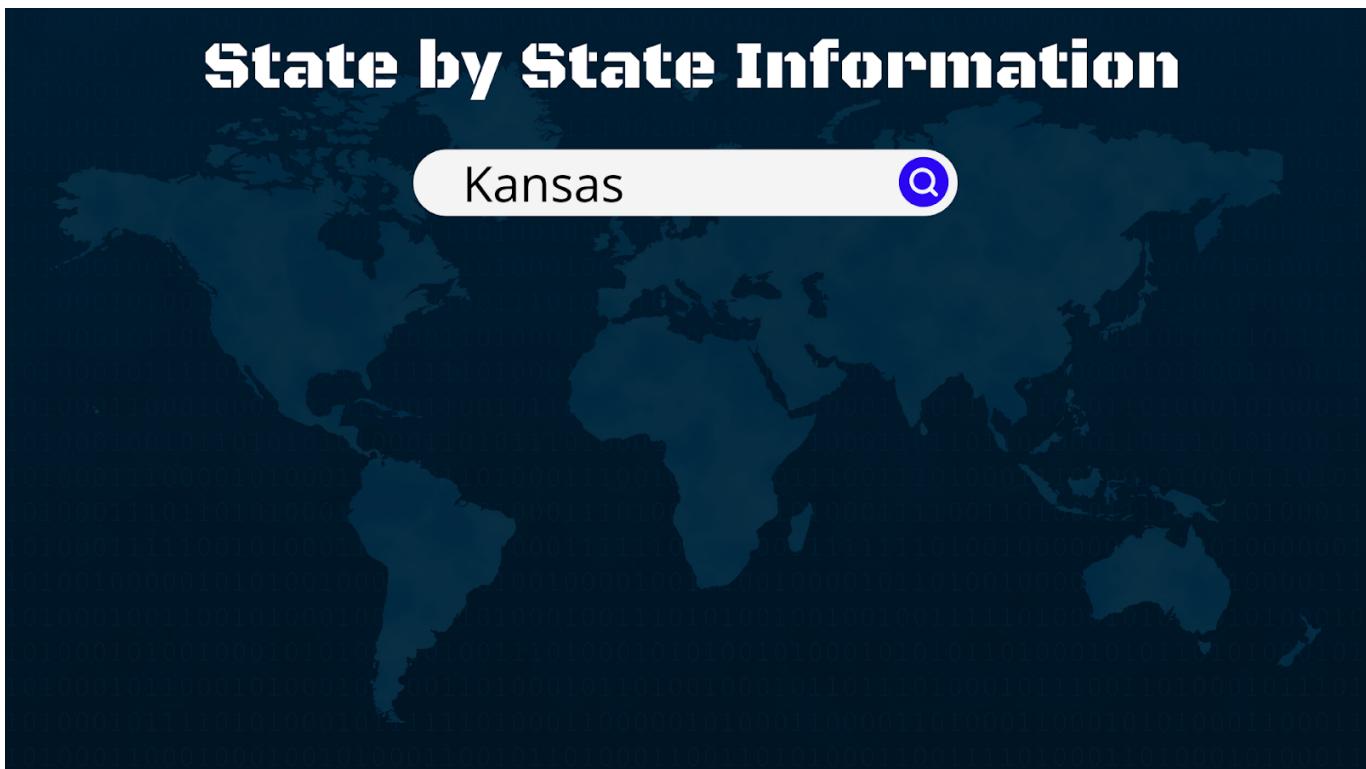
```
val violentCrime =  
    dataframe.sort(col("").desc).limit(10).write.format("csv").option("header",  
true).save("aggCsvFiles/US_violent_crime.csv")  
}
```

Oppgave C

Oppgavetekst:

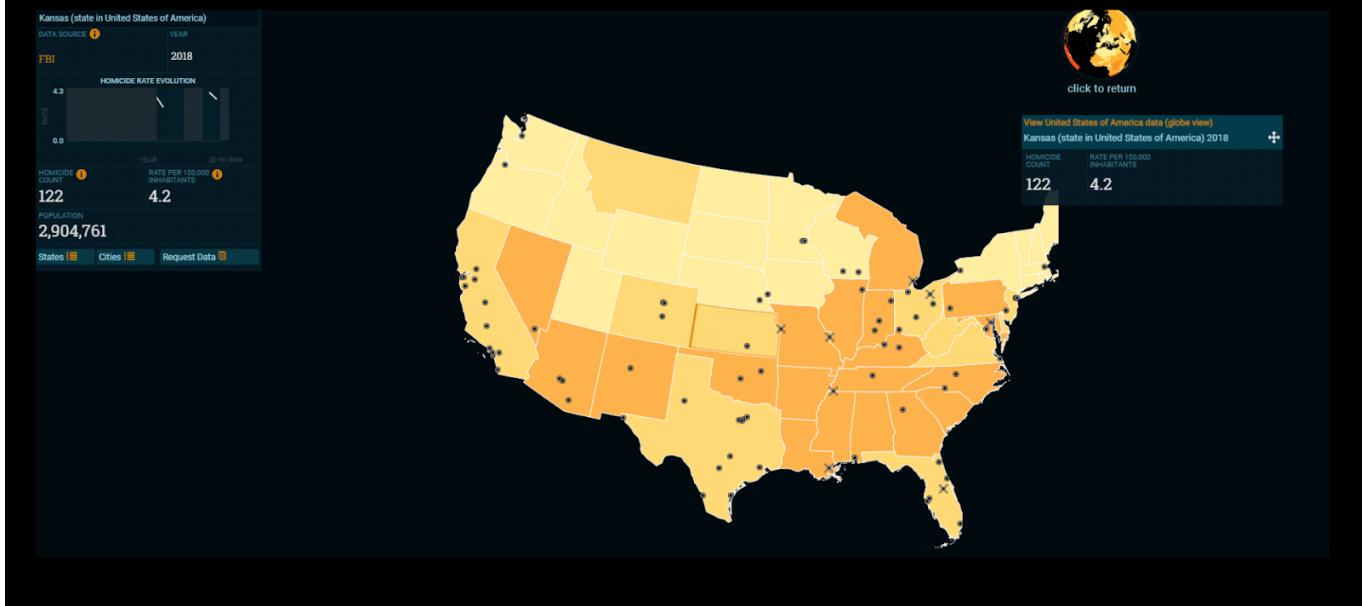
Legg til litt mer funksjonalitet i nettsiden som viser dybden i hva dere får til med Spark på deres datasett.

Svar:



Vi har valgt å implementere en funksjon som gjør det mulig å velge en bestemt stat i en søke bar, for å kunne se all informasjonen om valgt stat samlet.

Kansas



Eksempel på formatet informasjonen om søkt/valgt stat.

Vår side (hvis siden hadde vært ekte) ville hatt samme oppsett som denne siden - link til siden. Den vil komme opp når du søker på en stat.

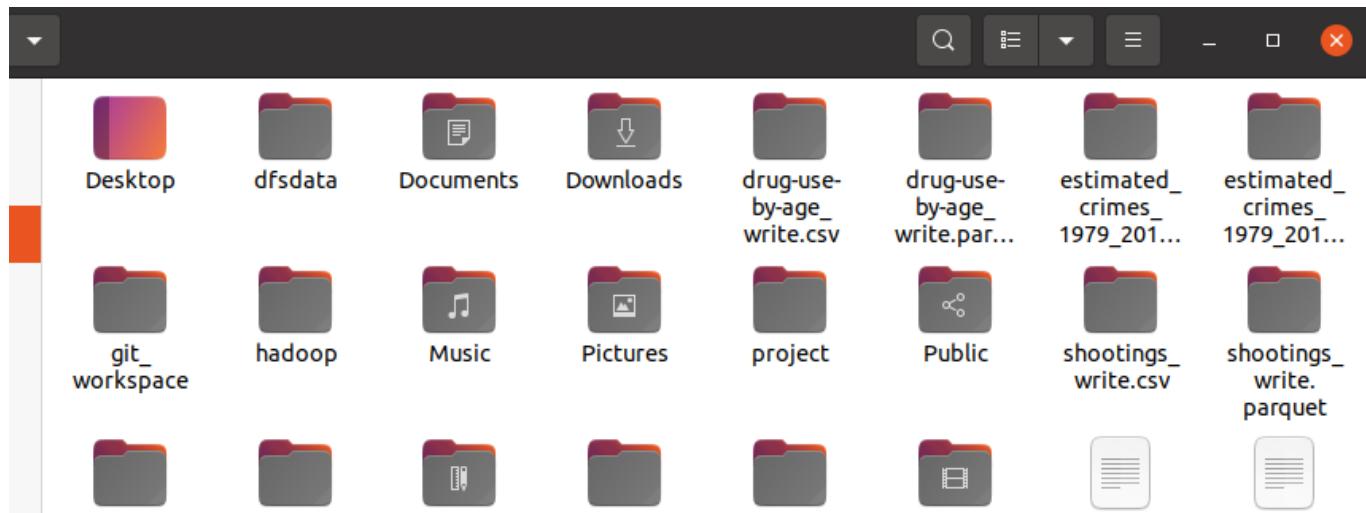
Del 2 - HDFS

Oppgave A

Oppgavetekst:

Skriv kommandoer for å overføre PARQUET filene deres til HDFS.

Svar:



snap	target	Templates	US_violent_crime_write.csv	US_violent_crime_write.par...	Videos	build.sdt	drug-use-by-age.csv

estimated_crimes_1979_2019.csv

shootings.copy.csv

shootings.csv

US_violent_crime.csv

"estimated_crimes_1979_2019.csv" selected (196.0 kB)

```

kms      run KMS, the Key Management Server
registrydns  run the registry DNS server

SUBCOMMAND may print help when invoked w/o parameters or with -h.
student@raptor:~$ hadoop fs -copyFromLocal shootings.csv/shootings.csv
copyFromLocal: `.' : No such file or directory: 'hdfs://0.0.0.0:9000/user/student'
student@raptor:~$ hadoop fs -ls hdfs:///
student@raptor:~$ hadoop fs -copyFromLocal shootings.csv /shootings.csv
student@raptor:~$ hadoop fs -ls hdfs:///
Found 1 items
-rw-r--r--  1 student supergroup      523013 2021-11-20 23:52 hdfs:///shootings.csv
student@raptor:~$ hadoop fs -copyToLocal /shootings.csv shootings.copy.csv
student@raptor:~$ hadoop fs -copyFromLocal druge-use-by-age.csv /drug-use-by-age.csv
copyFromLocal: 'druge-use-by-age.csv' : No such file or directory
student@raptor:~$ hadoop fs -copyFromLocal drug-use-by-age.csv /drug-use-by-age.csv
student@raptor:~$ hadoop fs -copyFromLocal US_violent_crime.csv /US_violent_crime.csv
student@raptor:~$ hadoop fs -copyFromLocal estimated_crimes_1979_2019.csv /estimated_crimes_1979_2019.csv
student@raptor:~$ hadoop fs -ls hdfs:///
Found 4 items
-rw-r--r--  1 student supergroup      1392 2021-11-20 23:59 hdfs:///US_violent_crime.csv
-rw-r--r--  1 student supergroup      2495 2021-11-20 23:58 hdfs:///drug-use-by-age.csv
-rw-r--r--  1 student supergroup    196049 2021-11-21 00:00 hdfs:///estimated_crimes_1979_2019.csv
-rw-r--r--  1 student supergroup      523013 2021-11-20 23:52 hdfs:///shootings.csv
student@raptor:~$ 
```

Oppgave B

Oppgavetekst:

Modifisert Spark-programmene deres fra (1.B) til å lese fra HDFS istedenfor lokalt filsystem.

Svar:

```
scala> val myDF = spark.read.format("csv").option("inferSchema", "true").option("inferSchema", "true").load("hdfs://localhost:9000/s
hootings.csv")
myDF: org.apache.spark.sql.DataFrame = [_c0: string, _c1: string ... 13 more fields]

scala> myDF.show()
<console>:24: error: not found: value myDF
      myDF.show()
           ^

scala> myDF.show()
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| _c0 |      _c1 |      _c2 |      _c3 |      _c4 |      _c5 |      _c6 |      _c7 |      _c8 |      _c9 |      _c10 |
| _c11|      _c12|      _c13|      _c14|      _c15|      _c16|      _c17|      _c18|      _c19|      _c20|      _c21|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id|      name|      date| manner_of_death|      armed|      age|gender|      race|      city|state|signs_of_mental_i...|thre
at_level|      flee|body_camera|      arms_category|
| 3| Tim Elliot|2015-01-02|      shot|      gun|53.0|      M| Asian|      Shelton| WA|      True| |
| attack|Not fleeing| False|      Guns|      shot|      gun|47.0|      M| White|      Aloha| OR|      False|
| 4| Lewis Lee Lembke|2015-01-02|      shot|      gun|47.0|      M| White|      Aloha| OR|      False|
| attack|Not fleeing| False|      Guns|      shot|      gun|23.0|      M| Hispanic|      Wichita| KS|      False|
| 5| John Paul Quintero|2015-01-03| shot and Tasered| unarmed|      23.0|      M| Hispanic|      Wichita| KS|      False|
| other|Not fleeing| False|      Unarmed|      shot|      toy weapon|32.0|      M| White| San Francisco| CA|      True|
| 8| Matthew Hoffman|2015-01-04|      shot|      toy weapon|32.0|      M| White| San Francisco| CA|      True|
| attack|Not fleeing| False|Other unusual obj...|      shot|      nail gun|39.0|      M| Hispanic|      Evans| CO|      False|
| 9| Michael Rodriguez|2015-01-04|      shot|      nail gun|39.0|      M| Hispanic|      Evans| CO|      False|
| attack|Not fleeing| False| Piercing objects|      shot|      gun|18.0|      M| White|      Guthrie| OK|      False|
| 11| Kenneth Joe Brown|2015-01-04|      shot|      gun|18.0|      M| White|      Guthrie| OK|      False|
| attack|Not fleeing| False|      Guns|      shot|      gun|22.0|      M| Hispanic|      Chandler| AZ|      False|
| 13| Kenneth Arnold Buck|2015-01-05|      shot|      gun|22.0|      M| Hispanic|      Chandler| AZ|      False|
| attack|Car| False|      Guns|      shot|      gun|35.0|      M| White|      Assaria| KS|      False|
| 15| Brock Nichols|2015-01-06|      shot|      gun|35.0|      M| White|      Assaria| KS|      False|
| attack|Not fleeing| False|      Guns|      shot|      gun|34.0|      F| White|      Burlington| IA|      False|
| 16| Autumn Steele|2015-01-06|      shot|      unarmed|34.0|      F| White|      Burlington| IA|      False|
| other|Not fleeing| True|      Unarmed|      shot|      toy weapon|47.0|      M| Black|      Knoxville| PA|      False|
| 17| Leslie Sapp III|2015-01-06|      shot|      toy weapon|47.0|      M| Black|      Knoxville| PA|      False|
| attack|Not fleeing| False|Other unusual obj...|      shot|      knife|25.0|      M| White|      Stockton| CA|      False|
| 19| Patrick Wetter|2015-01-06| shot and Tasered|      knife|25.0|      M| White|      Stockton| CA|      False|
| attack|Not fleeing| False| Sharp objects|      shot|      gun|31.0|      M| Black|      Freeport| TX|      False|
| 21| Ron Sneed|2015-01-07|      shot|      gun|31.0|      M| Black|      Freeport| TX|      False|
| attack|Not fleeing| False|      Guns|      shot|      knife|41.0|      M| Black|      Columbus| OH|      True|
| 22| Hashim Hanif Ibn ...|2015-01-07|      shot|      knife|41.0|      M| Black|      Columbus| OH|      True|
| other|Not fleeing| False| Sharp objects|      shot|      gun|30.0|      M| White| Des Moines| IA|      False|
| 25| Nicholas Ryan Bri...|2015-01-07|      shot|      gun|30.0|      M| White| Des Moines| IA|      False|
| attack|Car| False|      Guns|      shot|      gun|37.0|      M| Black| New Orleans| LA|      False|
| 27| Omarr Julian Maxi...|2015-01-07|      shot|      gun|37.0|      M| Black| New Orleans| LA|      False|
| attack|Foot| True|      Guns|      shot|      unknown|28.0|      M| White|      Huntley| MT|      False|unde
| 29| Loren Simpson|2015-01-08|      shot|      unknown|28.0|      M| White|      Huntley| MT|      False|unde
| terminated|Not fleeing| False|      Unknown|      shot|      shovel|42.0|      M| White| Salt Lake City| UT|      False|
| 32| James Dudley Barker|2015-01-08|      shot|      shovel|42.0|      M| White| Salt Lake City| UT|      False|
| attack|Not fleeing| True| Blunt instruments|      shot|      unarmed|36.0|      M| Black|      Strong| AR|      False|
| 36| Artago Damon Howard|2015-01-08|      shot|      unarmed|36.0|      M| Black|      Strong| AR|      False|
| attack|Not fleeing| False|      Unarmed|      shot|      gun|49.0|      M| White| Syracuse| UT|      False|
| 37| Thomas Hamby|2015-01-08|      shot|      gun|49.0|      M| White| Syracuse| UT|      False|
| attack|Not fleeing| True|      Guns|      shot|      gun|49.0|      M| White| Syracuse| UT|      False|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 20 rows

scala>
```

Oppgave C

Oppgavetekst:

Beskriv nøyne hva som skjer bak kulissene når dere kjører programmene deres fra (2.B). Det holder med nøyne beskrivelse av et program. Velg det programmet som gjør dere istrand til å skrive en diskusjon som viser hvor godt dere forstår hvordan HDFS fungerer, og hvordan Spark og HDFS jobber sammen.

Svar:

Vi klarer ikke å finne en passende forklaring som vil tilfredstille det oppgaven spør om.

Milepæl 6 - Cassandra og Joins

Oppgavetekst:

1. Cassandra

1. Implementer kode som bruker Spark til å laste data inn i Cassandradatabasen deres slik dere planla i Milepæl 4.
2. Implementer kode for å kjøre spørninger dere trenger for nettsiden mot Cassandradatabasen fra Spark

2. Joine sammen datasettene deres.

Implementer funksjonalitet for nettsiden som krever at dere kobler sammen minst to av datasettene deres. Selve koblingen skal gjennomføres i Spark, men utover det skal dere selv velge hvordan data vil bli hentet inn i Spark og hvordan resultatet fra Spark kan bli gjort tilgjengelig for webapplikasjonen deres.

Noen muligheter:

- Spark leser inn CSV filene og utfører en beregning som innebærer en join og plasserer resultatet i en Cassandra-database, webapplikasjon kjører spørninger mot Cassandra databasen for å få tak i data nødvendig for visningen i nettsiden.
- Webapplikasjon kjører en Spark spørring, Spark henter data fra CSV filer.
- Data blir lastet i passende Cassandra og Neo4j databaser. Spark henter så data fra Neo4j og Cassandradatabasen, joiner sammen data og plasserer data i en KV-database. Webapplikasjon henter data fra kv-databasen.
- Eller noe helt annet.

For dette punktet er det to leveranser, de teller like mye, så pass på å bruke tid på skriving også:

- En beskrivelse og diskusjon av løsningen dere har valgt, hvor dere argumenterer for hvorfor dette er en god løsning og hvorfor den er bedre enn alternativene. Pass på å bruke teorien til å diskutere om løsningen møter kravene til den tenkte brukergruppen, og kravene dere setter til ytelse. Husk at en veldig kompleks løsning kan også være negativt, så forsvar kompleksiteten om dere mener en veldig kompleks løsning er det beste.
 - En implementasjon av alle elementene (unntatt selve webapplikasjonen)
-

Svar:

Del 1

```
21/11/22 14:02:09 INFO TaskSchedulerImpl: Killing all running tasks in stage 19: Stage finished
21/11/22 14:02:09 INFO DAGScheduler: Job 13 finished: count at filterRace.scala:29, took 0.025921 s
Antalle skytinger filtrert etter rase er: Hvite: 2476 Sorte: 1298 Asiatsk: 93 Latinamerikaner: 902 Boomershootings: 18 Male shooting
s: 4673
21/11/22 14:02:09 INFO SparkUI: Stopped Spark web UI at http://raptor:4041
```

Kode

```
import org.apache.spark.sql.SparkSession

import org.apache.spark.sql.
import org.apache.spark.sql.functions.
import org.apache.spark.sql.types._

object filterForRace {

  def main(args: Array[String]){

    val chosenFile = "shootings.csv"

    val spark = SparkSession.builder
      .config("spark.master", "local")
      .appName("Filter race").getOrCreate()

    val shootingsDf = spark.read.format("csv").option("inferSchema",
      "true").option("header", "true").load(chosenFile)

    val cacheDf = shootingsDf.cache()

    //Filtrering av skytinger etter raser
    val whiteShootings = cacheDf.filter(col("race") = "White").count()
    val blackShootings = cacheDf.filter(col("race") = "Black").count()
```

```

val asianShootings = cacheDf.filter(col("race") = "Asian").count()
val hispanicShootings = cacheDf.filter(col("race") = "Hispanic").count()
val boomerShootings = cacheDf.filter(col("age") = "65").count()
val genderShootings = cacheDf.filter(col("gender") = "M").count

// Printer ut til konsoll antalle skytinger i hver "rase"s
println(
"Antalle skytinger filtrert etter rase er:" +
s" Hvite: $whiteShootings" +
s" Sorte: $blackShootings" +
s" Asiatisk: $asianShootings" +
s" Latinamerikaner: $hispanicShootings" +
s" Boomershootings: $boomerShootings" +
s" Male shootings: $genderShootings"
)

// Planer om å filtrere etter forskjellige type våpen involvert, stat, kjønn,
aldersgruppe og lignende

spark.stop()

}
}

```

Del 2

US Estimated Crimes:

year	state_abbr	state_name	population	violent_crime	homicide	rape_legacy	rape_revised	robbery	aggravated_assault	property_crime	burglary	larceny	motor_vehicle_theft	caveats
1979	null		220899000	1208030	21460	76390	null	480700	629480	11041500	3327700	6681000	1112800	null
1979	AK	Alaska	466800	1994	54	292	null	445	1203	23193	5616	15076	2501	null
1979	AL	Alabama	3769800	15578	496	1037	null	4127	9918	144372	48517	83791	12064	null
1979	AR	Arkansas	2188000	7984	198	595	null	1626	5565	70949	21457	45267	4225	null
1979	AZ	Arizona	2450000	14528	219	1120	null	4305	8884	177977	48916	116976	12085	null
1979	CA	California	22496000	184087	2952	12239	null	75767	93129	1511021	496310	847148	167563	null
1979	CO	Colorado	2772000	14472	161	1472	null	4353	8486	180984	49741	117898	13345	null
1979	CT	Connecticut	3115800	12900	131	752	null	6021	5999	167131	48229	96997	21905	null
1979	DC	District of Columbia	656800	10553	180	489	null	6920	2964	45877	13452	28819	3666	null
1979	DE	Delaware	582000	3127	33	162	null	753	2179	34853	8898	23081	2882	null
1979	FL	Florida	8860000	73881	1084	4576	null	22897	46124	607281	199884	378099	38298	null
1979	GA	Georgia	5118000	28594	877	2216	null	10939	14562	248641	81579	145758	21304	null
1979	HI	Hawaii	915000	2651	66	296	null	1688	601	63664	16538	40586	6546	null
1979	IA	Iowa	2993000	5259	65	320	null	1457	3417	119620	26768	85023	7829	null
1979	ID	Idaho	995000	2613	49	186	null	392	1986	35766	9729	23577	2460	null
1979	IL	Illinois	11230000	83540	1203	3702	null	36056	42579	593750	161776	350602	75912	null
1979	IN	Indiana	5400000	18254	448	1681	null	7167	8958	230223	63176	143666	23381	null
1979	KS	Kansas	2369800	8376	130	626	null	2423	5197	107605	31504	69622	6479	null
1979	KY	Kentucky	3527000	8748	335	719	null	3247	4447	103548	32082	62431	9035	null
1979	LA	Louisiana	4026800	27229	682	1554	null	8832	16161	188514	56237	115856	16421	null

```

scala> val crimePerStateDF = estimatedCrimesDF.groupBy("state_abbr").agg((sum("Homicide")).alias("homicide"), sum("Robbery").as("robbery"))
crimePerStateDF: org.apache.spark.sql.DataFrame = [state_abbr: string, homicide: bigint ... 1 more field]

```

```

scala> crimePerStateDF.show()

```

state_abbr	homicide	robbery
AZ	14423	273632
SC	13699	203021
LA	24454	325634
MN	4702	168416
NJ	15709	654373
DC	9666	210025
OR	4468	141199
VA	17736	267683
null	759281	19125371
RI	1317	35669
KY	9663	135729
WY	727	3820
NH	805	15142
MI	31316	666809
NV	6475	184295
WI	7251	182058
ID	1352	9848
CA	105626	3178754
CT	5403	200701
NE	2145	42139

Shootings by police

```
scala> val shootingByPoliceDF = spark.read.format("csv").option("header", true).option("inferSchema", "true").load("shootings.csv")
shootingByPoliceDF: org.apache.spark.sql.DataFrame = [id: int, name: string ... 13 more fields]
```

id	name	date	manner_of_death	armed	age	gender	race	city	state	signs_of_mental_illness	threat_level	flee	body_camera	arms_category
3	Tim Elliot	2015-01-02	shot	gun	53.0	M	Astan	Shelton	WA	true	attack Not fleeing	false	Guns	
4	Lewis Lee Lembke	2015-01-02	shot	gun	47.0	M	White	Aloha	OR	false	attack Not fleeing	false	Guns	
5	John Paul Quintero	2015-01-03	shot and Tasered	unarmed	23.0	M	H Spanic	Wichita	KS	false	other Not fleeing	false	Unarmed	
8	Matthew Hoffman	2015-01-04	shot	toy weapon	32.0	M	White	San Francisco	CA	true	attack Not fleeing	false	Other unusual obj...	
9	Michael Rodriguez	2015-01-04	shot	ball gun	39.0	M	H Spanic	Evans	CO	false	attack Not fleeing	false	Piercing objects	
11	Kenneth Joe Brown	2015-01-04	shot	gun	18.0	M	White	Guthrie	OK	false	attack Not fleeing	false	Guns	
13	Kenneth Arnold Buck	2015-01-05	shot	gun	22.0	M	H Spanic	Chandler	AZ	false	attack Car	false	Guns	
15	Brock Nichols	2015-01-06	shot	gun	35.0	M	White	Assaria	KS	false	attack Not fleeing	false	Guns	
16	Autumn Steele	2015-01-06	shot	unarmed	34.0	F	White	Burlington	IA	false	other Not fleeing	true	Unarmed	
17	Leslie Sapp III	2015-01-06	shot	toy weapon	47.0	M	Black	Knoxville	PA	false	attack Not fleeing	false	Other unusual obj...	
19	Patrick Wetter	2015-01-06	shot and Tasered	knife	25.0	M	White	Stockton	CA	false	attack Not fleeing	false	Sharp objects	
21	Ron Sneed	2015-01-07	shot	gun	31.0	M	Black	Freeport	TX	false	attack Not fleeing	false	Guns	
22	Hashim Hantf Ibn ...	2015-01-07	shot	knife	41.0	M	Black	Columbus	OH	true	other Not fleeing	false	Sharp objects	
25	Nicholas Ryan Br...	2015-01-07	shot	gun	36.0	M	White	Des Moines	IA	false	attack Car	false	Guns	
27	Omarr Julian Max...	2015-01-07	shot	gun	37.0	M	Black	New Orleans	LA	false	attack Foot	true	Guns	
29	Loren Simpson	2015-01-08	shot	unknown	28.0	M	White	Huntley	MT	false	undetermined Not fleeing	false	Unknown	
32	James Dudley Barker	2015-01-08	shot	shovel	42.0	M	White Salt Lake City	UT	false	attack Not fleeing	true	Blunt instruments		
36	Artago Damon Howard	2015-01-08	shot	unarmed	36.0	M	Black Strong	AR	false	attack Not fleeing	false	Unarmed		
37	Thomas Hamby	2015-01-08	shot	gun	49.0	M	White	Syracuse	UT	false	attack Not fleeing	true	Guns	
38	Jimmy Foreman	2015-01-09	shot	gun	71.0	M	White	England	AR	false	attack Not fleeing	false	Guns	

```
scala> val shootingsWithGunDF = shootingByPoliceDF.groupBy("state").agg((count(when($"race" === "Black", 1)) / count("*")).as("Blacks_shoot"),(count(when($"race" === "White", 1)) / (count("*"))).as("Whites_shoot"))
shootingsWithGunDF: org.apache.spark.sql.DataFrame = [state: string, Blacks_shoot: double ... 1 more field]
```

```
scala> shootingsWithGunDf.show()
+-----+-----+
|state| Blacks_shoot| Whites_shoot|
+-----+-----+
| AZ| 0.07657657657657657| 0.49099099099099097|
| SC| 0.3375| 0.625|
| LA| 0.5882352941176471| 0.38235294117647056|
| MN| 0.16666666666666666| 0.61666666666666667|
| NJ| 0.5| 0.35|
| DC| 0.9230769230769231| 0.07692307692307693|
| OR| 0.09210526315789473| 0.8289473684210527|
| VA| 0.43478260869565216| 0.4891304347826087|
| RI| 0.5| 0.25|
| KY| 0.1724137931034483| 0.7701149425287356|
| WY| 0.0| 0.6923076923076923|
| NH| 0.0| 1.0|
| MI| 0.3380281690140845| 0.6056338028169014|
| NV| 0.17647058823529413| 0.4588235294117647|
| WI| 0.22727272727272727| 0.6363636363636364|
| ID| 0.02702702702702703| 0.8108108108108109|
| CA| 0.17403708987161198| 0.3152639087018545|
| NE| 0.20833333333333334| 0.6666666666666666|
| CT| 0.15| 0.55|
| MT| 0.0| 0.8620689655172413|
+-----+
```

Joiner sammen begge csv filene

```
scala> val joinExpression = crimePerStateDf.col("state_abbr") === shootingsWithGunDf.col("state")
joinExpression: org.apache.spark.sql.Column = (state_abbr = state)

scala> val joinedDf = shootingsWithGunDf.join(crimePerStateDf, joinExpression).show()
+-----+-----+-----+-----+-----+
|state| Blacks_shoot| Whites_shoot|state_abbr|homicide|robbery|
+-----+-----+-----+-----+-----+
| AZ| 0.07657657657657657| 0.49099099099099097| AZ| 14423| 273632|
| SC| 0.3375| 0.625| SC| 13699| 203021|
| LA| 0.5882352941176471| 0.38235294117647056| LA| 24454| 325634|
| MN| 0.16666666666666666| 0.6166666666666667| MN| 4702| 168416|
| NJ| 0.5| 0.35| NJ| 15709| 654373|
| DC| 0.9230769230769231| 0.07692307692307693| DC| 9666| 210025|
| OR| 0.09210526315789473| 0.8289473684210527| OR| 4468| 141199|
| VA| 0.43478260869565216| 0.4891304347826087| VA| 17736| 267683|
| RI| 0.5| 0.25| RI| 1317| 35669|
| KY| 0.1724137931034483| 0.7701149425287356| KY| 9663| 135729|
| WY| 0.0| 0.6923076923076923| WY| 727| 3820|
| NH| 0.0| 1.0| NH| 805| 15142|
| MI| 0.3380281690140845| 0.6056338028169014| MI| 31316| 666809|
| NV| 0.17647058823529413| 0.4588235294117647| NV| 6475| 184295|
| WI| 0.22727272727272727| 0.6363636363636364| WI| 7251| 182058|
| ID| 0.02702702702702703| 0.8108108108108109| ID| 1352| 9848|
| CA| 0.17403708987161198| 0.3152639087018545| CA| 105626| 3178754|
| NE| 0.20833333333333334| 0.6666666666666666| NE| 2145| 42139|
| CT| 0.15| 0.55| CT| 5403| 200701|
| MT| 0.0| 0.8620689655172413| MT| 1229| 9075|
+-----+
only showing top 20 rows

joinedDf: Unit = ()
```

Brukeren får tilgang til resultatet ved å filtrere på siden våres. Brukeren kan velge hvilke datasett som skal ses sammen og hva brukeren ønsker at skal vises. For eksempel da på delen over, hvor mange svarte og hvite mennesker som er skutt av politiet samt hvor mange drap og ran det er begått i hver stat.

Kommentar(NB!):

På grunn av uforventet dropout og halvering av gruppen så ble vi nødt til å starte prosjektet på nytt rett før "Milepæl 4" skulle leveres. Det vil si litt over en(1) månede fra "Milepæl 1" ble levert. Dette førte til at vi kunn klarte, i forhold til tiden, å levere inn de 6 obligatoriske oppgavene. Vi hadde lyst til å levere "Milepæl 7(bonus)" og et arbeid av høyere kvalitet, men tiden stod ikke til.

Vi håper fremdeles at oppgaven er av høy kvalitet og viser vår kompetanse i faget.

-Gruppe 343

Refleksjonsdel(Samlet):

Vi har lært mye, men så har situasjonen som vi forklarte ovenfor svekket vår til å lære faget på et dypere. Istedentfor så har vårt fokus blitt endret til å bli ferdig med oppgavene så fort vi kan. Dette har skapt veldig mye stress på oss som studenter og videre påvirket den endelige rapporten.

Spesielt "Milepæl 4" ble litt for mye å gjennomføre i løpe av den tiden vi hadde på oss og skulle ønsket at den var delt opp i 2 separate milepæler ettersom at den handler om 3 forskjellige databaseparadigmer.

Koding i Neo4j var veldig vanskelig, som sett ovenfor så fikk vi det ikke helt til. Det kunne kanskje gått bedre om det var bedre tid og færre milepæler

For å oppsummere så har vi lært mye og dette var et interessant og lærerikt fag med god undervisning. Veiledningen fra foreleser har vært ekspjonelt. Videre så har foreleseren stått bak oss hele veien for å hjelpe oss tilbake på riktig spor. Dette har ført til at vi har gjennomført de obligatoriske milepælene, men som nevnt ovenfor strakk ikke tiden til å gjennomføre "Milepæl 7".

Vi er fornøyde med det vi har fått til i oppgaven, men vi skulle ønske at vi leverte inn et bedre sluttprodukt. Koden og teorien kunne vært forbedret, men desverre som alle ting her i livet så kan Thanos dukke opp med infinity stones og knippse vekk halve gruppen. Dessverre så hadde vi ikke tilgang til Ant-Man som kunne kanskje sendt oss tilbake med tidsmaskinen sin for å hente tilbake gruppemedlemene vi mistet.



Kilder

- Foreleser og forelesninger i faget