

Milepæl 5 – Spark og HDFS

Gruppe 13: Mats Engelen og Lars Erik Faber

GitHub: <https://github.com/larseriktf/BigData>

Innhold

Lese/skrive CSV/PARQUET filer.....	2
Generell Kode for lesing/skriving.....	2
Student Performance Dataset (KV-database).....	2
Første aggregering	2
Andre Aggregering	3
Tredje Aggregering.....	3
Socio-Economic Country Profiles (Dokumentdatabase).....	4
Første Aggregering.....	4
Andre Aggregering	5
World University Rankings (Kolonnefamiliedatabase)	6
Første Aggregering.....	7
Andre Aggregering	7
Tredje Aggregering.....	8
Government Types Of The World (Grafdatabase).....	9
Første Aggregering.....	9
Endringer på nettsiden	11
HDFS.....	11
Listing/remove	11
Lese fra HDFS	11
Kopiere fra local til HDFS	11
Hva skjer egentlig?	11
Eksempel	12

Lese/skrive CSV/PARQUET filer

Før vi kan begynne å lage aggregeringer, må vi gjøre datasettene om til Parquet filer. Det er alltid lurt å gjøre om til Parquet først, siden de tilbyr komprimering.

Generell Kode for lesing/skriving

```
val myDf = spark.read.option("header","true").csv("student-performance.csv")

myDf.write.format('parquet').option('header','true').mode('overwrite').save('student-
performance.parquet')

val PDF = spark.read.option("header","true").parquet("student-performance.parquet")
```

Student Performance Dataset (KV-database)

Første aggregering

```
val result1 = PDF.groupBy("Fedu").agg(expr(" avg(G1) as FirstYear"), expr(" avg(G2) as
SecondYear"),expr(" avg(G3) as ThirdYear"))
```

```
val result2 = PDF.groupBy("Medu").agg(expr(" avg(G1) as FirstYear"), expr(" avg(G2) as
SecondYear"),expr(" avg(G3) as ThirdYear"))
```

```
val joined = result1.joinWith(result2, result1("Fedu") === result2("Medu"))
```

```
val temp = joined.selectExpr("_1 as Father, "_2 as Mother")
```

```
val temp2 = temp.select(col("Father.*"),col("Mother.*"))
```

```
val flattenTemp = temp2.toDF("FatherEdu","FFirstYear","FSecondYear","FThirdYear",
"MotherEdu","MFirstYear","MSecondYear","MThirdYear")
```

```
flattenTemp.write.format("csv").option("header","true").mode("overwrite").save("grades-
average.csv")
```

Andre Aggregering

```
val temp = PDF.groupBy("studytime").agg(expr(" avg(G1) as FirstYear"), expr( " avg(G2) as  
SecondYear"),expr( " avg(G3) as ThirdYear"))
```

```
val setString = udf {(freetime: Integer) => if(freetime == 1) "Very little" else if(freetime == 2) "Little"  
else if(freetime == 3) "Medium" else if(freetime == 4) "Much" else "Very Much"}
```

```
val temp2 = temp.withColumn("freetime", setString(temp("freetime")))
```

```
temp2.write.format("csv").option("header", "true").mode("overwrite").save("freetime-grades.csv")
```

Tredje Aggregering

```
val temp = PDF.groupBy("studytime").agg(expr(" avg(G1) as FirstYear"), expr( " avg(G2) as  
SecondYear"),expr( " avg(G3) as ThirdYear"))
```

```
val setString = udf {(studytime: Integer) => if(studytime == 1) "Very little" else if(studytime == 2)  
"Little" else if( studytime == 3) "Medium" else if( studytime == 4) "Much" else "Very Much"}
```

```
val temp2 = temp.withColumn("studytime", setString(temp("studytime")))
```

```
temp2.write.format("csv").option("header", "true").mode("overwrite").save("studytime-grades.csv")
```

Socio-Economic Country Profiles (Dokumentdatabase)

Første Aggregering

```
val temp = myDf.select("country", "Population in thousands (2017)", "GDP: Gross domestic product (million current US$)", "GDP per capita (current US$)", "Unemployment (% of labour force)", "Population growth rate (average annual %)", "Urban population (% of total population)_x", "Urban population growth rate (average annual %)", "Health: Total expenditure (% of GDP)", "Education: Government expenditure (% of GDP)", "Individuals using the Internet (per 100 inhabitants)", "Quality Of Life Index", "Purchasing Power Index", "Safety Index", "Health Care Index", "Property price to income ratio", "Affordability Index", "Cost Of Living Index", "Cost Of Living Plus Rent Index", "Life expectancy at birth, total (years)", "Military expenditure (% of GDP)", "Tax revenue (% of GDP)")
```

```
temp.write.option('header', 'true').mode('overwrite').parquet('country-profiles-trimmed.parquet')
```

```
val customSchema = StructType(Array(StructField("country", StringType, true),
StructField("Population in thousands (2017)", IntegerType, true),
StructField("GDP: Gross domestic product (million current US$)", DoubleType, true),
StructField("GDP per capita (current US$)", DoubleType, true),
StructField("Unemployment (% of labour force)", DoubleType, true),
StructField("Population growth rate (average annual %)", DoubleType, true),
StructField("Urban population (% of total population)_x", DoubleType, true),
StructField("Urban population growth rate (average annual %)", DoubleType, true),
StructField("Health: Total expenditure (% of GDP)", DoubleType, true),
StructField("Education: Government expenditure (% of GDP)", DoubleType, true),
StructField("Individuals using the Internet (per 100 inhabitants)", IntegerType, true),
StructField("Quality Of Life Index", DoubleType, true),
StructField("Purchasing Power Index", DoubleType, true),
StructField("Safety Index", DoubleType, true),
StructField("Health Care Index", DoubleType, true),
StructField("Property price to income ratio", DoubleType, true),
StructField("Affordability Index", DoubleType, true),
StructField("Cost Of Living Index", DoubleType, true),
StructField("Cost Of Living Plus Rent Index", DoubleType, true),
StructField("Life expectancy at birth, total (years)", DoubleType, true),
StructField("Military expenditure (% of GDP)", DoubleType, true),
```

```
StructField("Tax revenue (% of GDP)", DoubleType,true))
```

```
val myDf = spark.read.option("header","true").option('customSchema', 'true').parquet("country-  
profiles.parquet")
```

```
val oldCol = Seq("country", "Population in thousands (2017)", "GDP: Gross domestic product (million  
current US$)", "GDP per capita (current US$)", "Unemployment (% of labour force)", "Population  
growth rate (average annual %)", "Urban population (% of total population)_x", "Urban population  
growth rate (average annual %)", "Health: Total expenditure (% of GDP)", "Education: Government  
expenditure (% of GDP)", "Individuals using the Internet (per 100 inhabitants)", "Quality Of Life  
Index", "Purchasing Power Index", "Safety Index", "Health Care Index", "Property price to income  
ratio", "Affordability Index", "Cost Of Living Index", "Cost Of Living Plus Rent Index", "Life expectancy at  
birth, total (years)", "Military expenditure (% of GDP)", "Tax revenue (% of GDP)" )
```

```
val newCol = Seq("country", "population", "gdp", "gdpPerCapita", "unemployment",  
"populationGrowthRate", "urbanPop", "urbanPopGrowth", "healthTotal", "educationTotal",  
"internetUsers", "qualityOfLifeI", "PPI", "safetyI", "HealthI", "propPriceToIncome", "affordabilityI",  
"costI", "costPlusRentI", "lifeExpectancy", "militaryTotal", "taxes")
```

```
val list = oldCol.zip(newCol).map(f=>{col(f._1).as(f._2)})
```

```
val newDF = myDf.select(list: _*)
```

```
newDF.write.format("csv").mode("overwrite").option("header", true).save("country-profiles.csv")
```

[Andre Aggregering](#)

```
val thisTemp = PDF.coalesce(1).select(col("country"), expr("(costPlusRentI / 100) * gdp as RealCost  
")).sort(desc("RealCost"))
```

World University Rankings (Kolonnefamiliedatabase)

Først leser vi csv filen med «inferSchema» satt til «true» for å opprette schema.

```
scala> val universityDf = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("datasets/timesData.csv")
universityDf: org.apache.spark.sql.DataFrame = [world_rank: string, university_name: string ... 12 more fields]
```

```
scala> universityDf.printSchema()
root
|-- world_rank: string (nullable = true)
|-- university_name: string (nullable = true)
|-- country: string (nullable = true)
|-- teaching: double (nullable = true)
|-- international: string (nullable = true)
|-- research: double (nullable = true)
|-- citations: double (nullable = true)
|-- income: string (nullable = true)
|-- total_score: string (nullable = true)
|-- num_students: string (nullable = true)
|-- student_staff_ratio: double (nullable = true)
|-- international_students: string (nullable = true)
|-- female_male_ratio: string (nullable = true)
|-- year: integer (nullable = true)
```

Jeg legger merke til at noen av datatypene ikke stemmer. Jeg ser at «world_rank» er satt til streng, selv om det tilsynelatende er int. Men det går fin, siden den også inneholder verdier som «100-150». Derimot, «international» var satt til streng, fordi null-verdier var byttet ut med bindestrek (-), I tillegg så jeg at «international_students» hadde verdier med tall og %-tegnet. Begge fikset jeg ved å bruke find&replace funksjonen i libreOffice og endre på csv-filen.

Da brukte jeg universityDf.printSchema() igjen for å sjekke at ting stemte, og det gjorde det.

```
scala> universityDf.printSchema()
root
|-- world_rank: string (nullable = true)

scala> universityDf.write.format("parquet").mode("errorIfExists").save("parquets/university_write.parquet")

|-- teaching: double (nullable = true)
|-- international: string (nullable = true)
|-- research: double (nullable = true)
|-- citations: double (nullable = true)
|-- income: string (nullable = true)
|-- total_score: string (nullable = true)
|-- num_students: string (nullable = true)
|-- student_staff_ratio: double (nullable = true)
|-- international_students: string (nullable = true)
|-- female_male_ratio: string (nullable = true)
|-- year: integer (nullable = true)
```

Det neste er å lagre den som en Parquet fil.

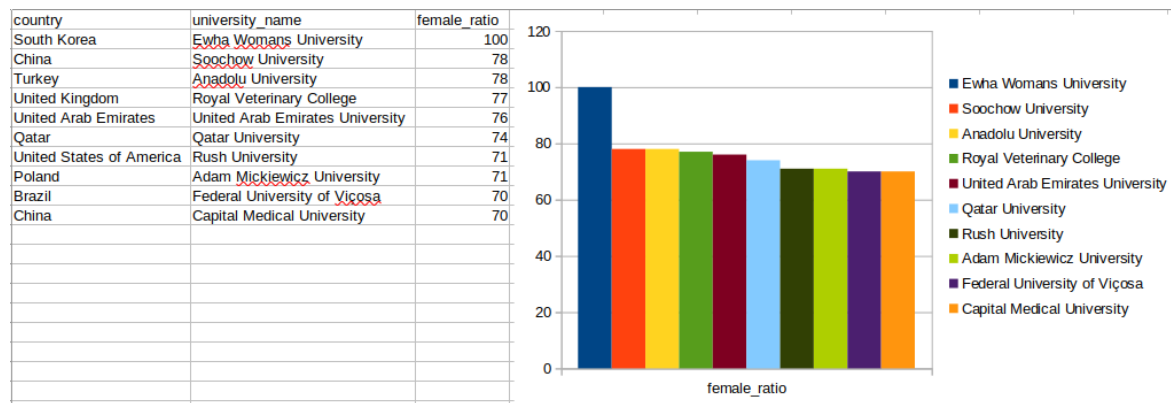
Første Aggregering

Den første aggregeringen jeg lagde viser de ti skolene med flest kvinnelige studenter.

```
scala> val universityByFemaleRatio = universityPdf.select(col("country"), col("university_name"), split(col("female_male_ratio"), " : ").getItem(0).cast("int").as("female_ratio")).distinct().sort(desc_u
lls_last("female_ratio")).limit(10)
universityByFemaleRatio: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [country: string, university_name: string ... 1 more field]
```

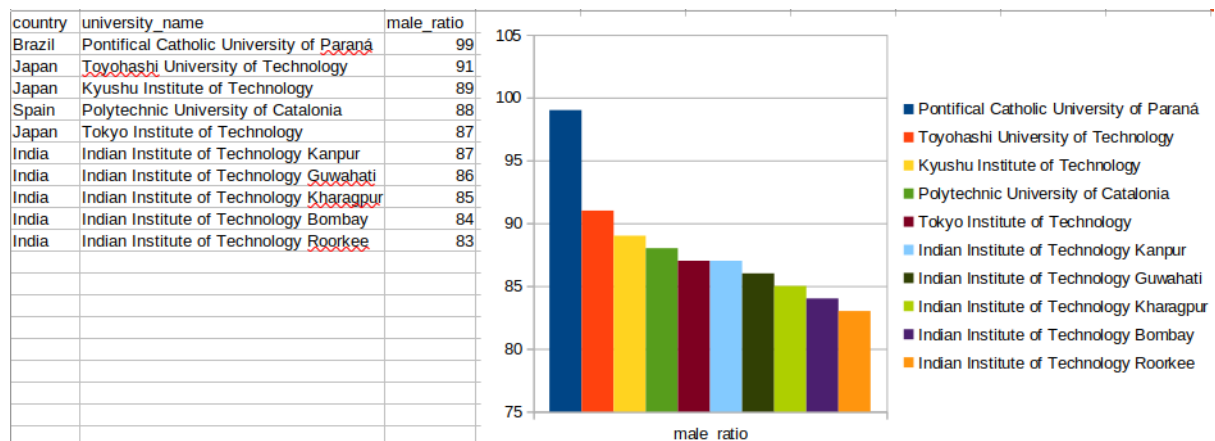
```
scala> university
universityByFemaleRatio universityPdf
```

```
scala> universityByFemaleRatio.show()
+-----+-----+-----+
|country|university_name|female_ratio|
+-----+-----+-----+
|South Korea|Ewha Womans Unive...|100|
|China|Soochow University|78|
|Turkey|Anadolu University|78|
|United Kingdom|Royal Veterinary ...|77|
|United Arab Emirates|United Arab Emira...|76|
|Qatar|Qatar University|74|
|United States of ...|Rush University|71|
|Poland|Adam Mickiewicz U...|71|
|Brazil|Federal Universit...|70|
|China|Capital Medical U...|70|
+-----+-----+-----+
```



Andre Aggregering

Denne er tilsvarende forrige aggregering, men denne gangen for mannlige studenter.

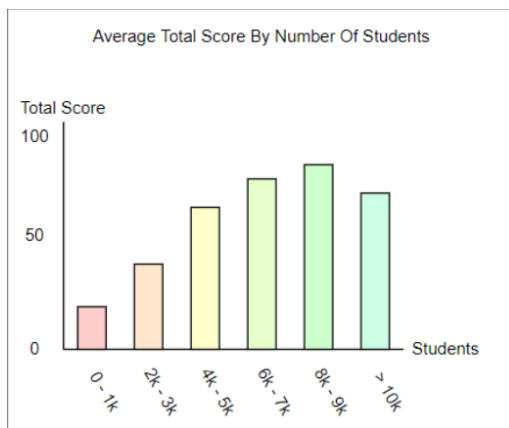


Så lagret jeg begge aggregeringene som CSV filer med én partisjon hver.

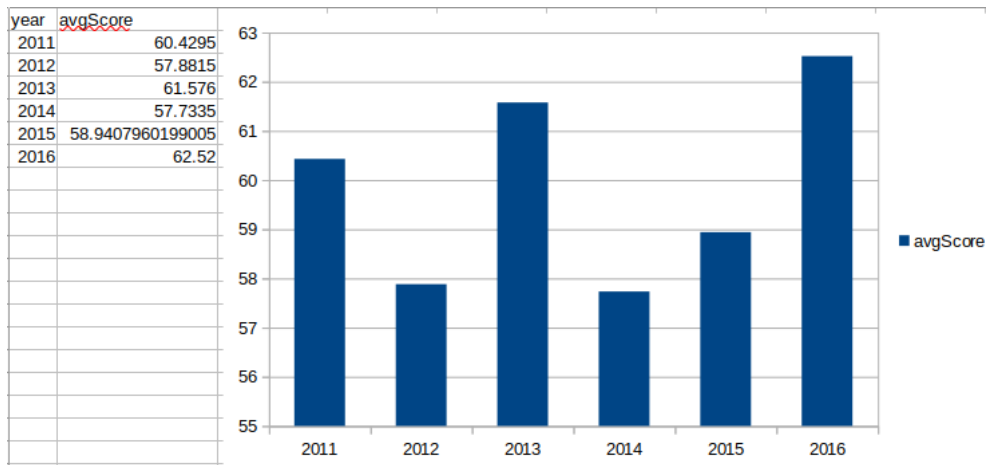
```
scala> universityByFemaleRatio.repartition(1).write.format("csv").option("header", "true").mode("overwrite").save("components/university_by_female_ratio")
scala> universityByMaleRatio.repartition(1).write.format("csv").option("header", "true").mode("overwrite").save("components/university_by_male_ratio")
```

Tredje Aggregering

Jeg lagde også en aggregering som viser gjennomsnittspoeng per år. Denne var, til å begynne med, ment til å representere «Average Total Score by Number Of Students» komponentet fra forrige milepæl:



Her er den nye aggregeringen (Glemte å ta bilde av scala koden):



Government Types Of The World (Grafdatabase)

Begynner med å lese csv filene. Merk at i dette datasettet er det 4 forskjellige csv-filer.

```
scala> val leaderDf = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("datasets/REIGN_dataset/leader_list_6_21.csv")
leaderDf: org.apache.spark.sql.DataFrame = [stateabb: string, ccode: int ... 16 more fields]

scala> val reignDf = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("datasets/REIGN_dataset/REIGN_2021_6.csv")
reignDf: org.apache.spark.sql.DataFrame = [ccode: double, country: string ... 39 more fields]

scala> val regimeDf = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("datasets/REIGN_dataset/regime_list.csv")
regimeDf: org.apache.spark.sql.DataFrame = [cowcode: int, gwf_country: string ... 4 more fields]

scala> val electionDf = spark.read.format("csv").option("header", "true").load("datasets/REIGN_dataset/election_6_21.csv")
```

Så skrive de til parquet filer.

```
scala> reignDf.repartition(1).write.format("parquet").mode("overwrite").save("parquets/reign_write.parquet")

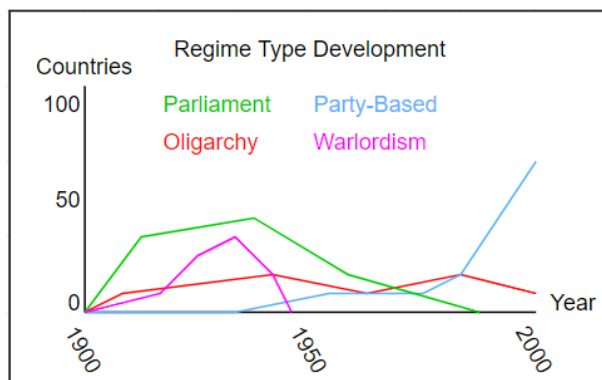
scala> regimeDf.repartition(1).write.format("parquet").option("header", "true").save("parquets/regime_write.parquet")

scala> electionDf.repartition(1).write.format("parquet").option("header", "true").save("parquets/election_write.parquet")

scala> leaderDf = repartition(1).write.format("parquet").mode("overwrite").save("parquets/leader_write.parquet")
```

Første Aggregering

Her lagde jeg kun en aggregering, men til prosjektinnleveringen vil vi bruke de andre også. Aggregeringen viser hvordan hver styremåte har utviklet seg i popularitet gjennom årene. Den opprinnelige skissen fra milepæl 4 ser slik ut:



Til å begynne med, tenkte jeg å gruppere først på år, så på styremåte, og dermed telle antall opplistinginger.

```
scala> val governmentPopularity = reignPDF.groupBy(col("year"), col("government")).agg(count("*")).as("count").orderBy(asc("year"), asc("government"))
governmentPopularity: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [year: double, government: string ... 1 more field]

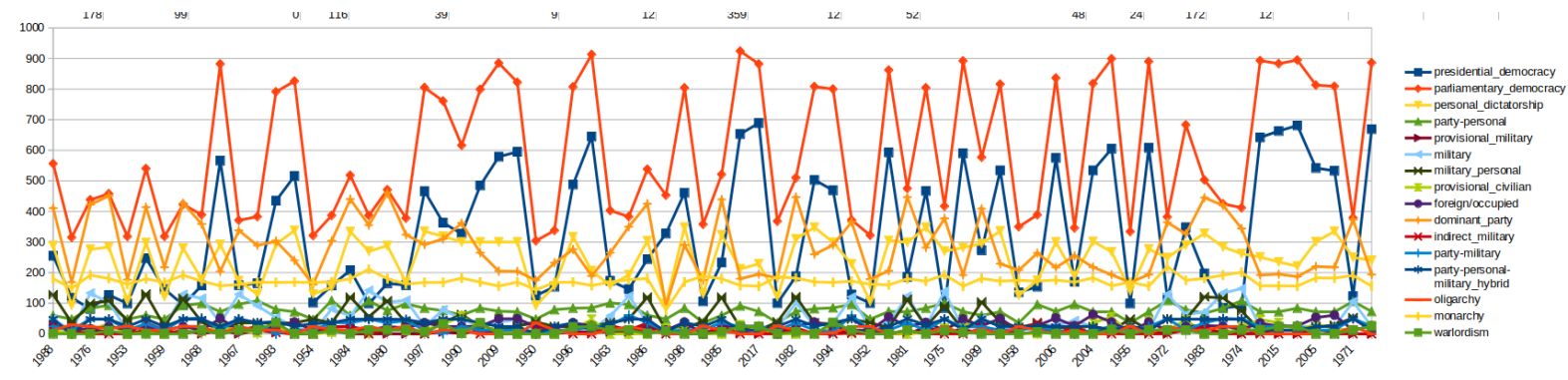
scala> governmentPopularity.show(10)
+-----+-----+-----+
| year|      government|count(1)|
+-----+-----+-----+
|1950.0|    Dominant Party|      175|
|1950.0| Foreign/Occupied|       24|
|1950.0| Indirect Military|       12|
|1950.0|      Military|        9|
|1950.0| Military-Personal|       39|
|1950.0|      Monarchy|      144|
|1950.0|      Oligarchy|       36|
|1950.0|Parliamentary Dem...|      304|
|1950.0|   Party-Military|        4|
|1950.0|   Party-Personal|       48|
+-----+-----+-----+
only showing top 10 rows

scala> governmentPopularity.repartition(1).write.format("csv").option("header", "true").save("components/government_popularity.csv")
```

Men dessverre var ikke dette optimalt for lesing av grafiske programmer (excel, libreoffice). Så jeg bestemte meg for å gjøre om på komponenten. I denne nye versjonen har jeg en kolonne for år, og en kolonne for hver styremåte. Da ble aggregeringen slik:

```
scala> val governmentPopularityByYear = reignPDF.groupBy("year").agg(
  count(when(col("government") === "Presidential Democracy", true)).as("presidential_democracy"),
  count(when(col("government") === "Parliamentary Democracy", true)).as("parliamentary_democracy"),
  count(when(col("government") === "Personal Dictatorship", true)).as("personal_dictatorship"),
  count(when(col("government") === "Party-Personal", true)).as("party-personal"),
  count(when(col("government") === "Provisional - Military", true)).as("provisional_military"),
  count(when(col("government") === "Military", true)).as("military"),
  count(when(col("government") === "Military-Personal", true)).as("military_personal"),
  count(when(col("government") === "Provisional - Civilian", true)).as("provisional_civilian"),
  count(when(col("government") === "Foreign/Occupied", true)).as("foreign/occupied"),
  count(when(col("government") === "Dominant Party", true)).as("dominant_party"),
  count(when(col("government") === "Indirect Military", true)).as("indirect_military"),
  count(when(col("government") === "Party-Military", true)).as("party-military"),
  count(when(col("government") === "Party-Personal-Military Hybrid", true)).as("party-personal-military_hybrid"),
  count(when(col("government") === "Oligarchy", true)).as("oligarchy"),
  count(when(col("government") === "Monarchy", true)).as("monarchy"),
  count(when(col("government") === "Warlordism", true)).as("warlordism"),
)
```

Framstilling i libreoffice:



Endringer på nettsiden

Vi fant ut i denne milepælen at dynamisk brukerinput ville være vanskelig å inkorporere i noen av aggregeringene, som f.eks land-aggregeringene i dokumentdatabasene. Derfor ville det vært vesentlig mindre brukerinteraksjon på nettsiden.

HDFS

Listing/remove

```
hadoop fs -ls hdfs:///          -rm
```

Lese fra HDFS

```
val PDF = spark.read.option("header","true").option("inferSchema","true").load("hdfs:///country-  
profiles.parquet")
```

```
val PDF = spark.read.option("header","true").parquet("hdfs://localhost:9000/filepath")
```

(.csv()) kommando hvis fil er av type CSV)

Kopiere fra local til HDFS

```
hadoop fs -copyFromLocal country-profiles.parquet /country-profiles.parquet
```

```
hadoop fs -copyFromLocal university_write.parquet /university-write.parquet
```

```
hadoop fs -copyFromLocal student-performance.parquet /student-performance.parquet
```

```
hadoop fs -copyFromLocal reign_write.parquet /reign-write.parquet
```

```
hadoop fs -copyFromLocal regime_write.parquet /regime-write.parquet
```

```
hadoop fs -copyFromLocal leader_write.parquet /leader-write.parquet
```

```
hadoop fs -copyFromLocal election_write.parquet /election-write.parquet
```

```
//-copyToLocal for motsatt
```

Hva skjer egentlig?

Det første vi gjør er å kopiere de lokale parquet-filene til HDFS, det vil fungere ca slik:

1. ber namenode om å opprette en fil. Namenode vil returnere en liste over noder for å lage replika blokker (første replika er lokalt plassert, andre på en annen rack, tredje på samme rack som replika 2)

- Bare en replika per node

- To replika per rack(om det er nok racks)

2. blokkdata skrives da til første node i namenode listen

3. ber namenode å hente ut neste sett med blokklokasjoner, skriv blokken

4. iformerer namenode om at filen er ferdig skrevet og gjør filen tilgjengelig

Nå som det er lagret på noder i HDFS kan vi da hente ut parquet-filene derfra via Spark med read-kommando og gjøre aggregeringer.

Eksempel

1. I average-grades.csv aggregeringen (aggregering 1) henter først ut hele listen fra HDFS og legger det i en dataframe.
2. Etter det lages det to Row() elementer som holder på hver sin del av dataen, en for fars utdanning og en for mors.
3. Etter det gjør vi en joinWith på dataen, og legger de sammen basert på verdien i utdanningsnivå som er 0-4.
4. Når har vi en dataframe bestående av to structs som vi bytter navn på
5. Vi «flater» så ut struct-typen som har blitt skapt av den tidligere joinWith for å pakke ut til to hovedgrupper.
6. Til slutt skrives filen til disk som csv fil, eller den kan skrives til HDFS.