



# Dapla-manual

# Innhold

<b>Velkommen</b>	<b>6</b>
Bidragsyttere . . . . .	6
<b>Forord</b>	<b>7</b>
<b>I    Introduksjon</b>	<b>8</b>
<b>1    Hva er Dapla?</b>	<b>10</b>
<b>2    Hvorfor Dapla?</b>	<b>11</b>
<b>3    Arkitektur</b>	<b>12</b>
<b>4    Innlogging</b>	<b>13</b>
4.1 Dapla . . . . .	13
4.1.1 Prod . . . . .	14
4.1.2 Staging . . . . .	18
4.2 Bakkemiljøet . . . . .	18
4.2.1 Prod . . . . .	18
4.2.2 Staging . . . . .	18
<b>5    Jupyterlab</b>	<b>19</b>
5.1 Hva er Jupyterlab? . . . . .	19
5.2 Terminalen . . . . .	19
5.3 Pakkeinstallasjoner . . . . .	19
5.4 Extensions . . . . .	19
5.5 Tips & triks . . . . .	19
<b>6    Bakke vs. sky</b>	<b>20</b>
<b>II   Opprette Dapla-team</b>	<b>21</b>
<b>7    Hva er Dapla-team?</b>	<b>23</b>
<b>8    Opprette Dapla-team</b>	<b>24</b>

<b>9 Bøtter</b>	<b>25</b>
9.1 Bøttenavn i Produksjonsmiljøet . . . . .	25
9.2 Andre miljøer . . . . .	26
9.3 Fellesbøtter . . . . .	26
<b>10 Google Cloud Console</b>	<b>27</b>
<b>11 Overføre data</b>	<b>28</b>
11.1 Forberedelser . . . . .	28
11.1.1 Logg inn i produksjonssonen . . . . .	28
11.2 Få tilgang til Linuxstammen . . . . .	28
11.2.1 Få tilgang til overføringsmapper . . . . .	29
11.3 Sette opp overføringsjobber . . . . .	29
11.3.1 Overføring fra Linuxstammen til Dapla . . . . .	31
11.3.2 Overføring fra Dapla til Linuxstammen . . . . .	32
<b>12 Lagre data</b>	<b>33</b>
<b>13 Hente data</b>	<b>34</b>
<b>14 Slette data fra bøtter</b>	<b>35</b>
<b>15 Gjenopprette data fra bøtter</b>	<b>39</b>
15.1 Gjenopprette en slettet fil . . . . .	39
15.2 Gjenopprette en fil til en tidligere versjon . . . . .	40
<b>16 Fra bakke til sky</b>	<b>41</b>
<b>17 Administrasjon av team</b>	<b>42</b>
17.1 Legge til eller fjerne medlemmer . . . . .	42
<b>III SSB-project</b>	<b>43</b>
<b>18 Nytt ssb-project</b>	<b>45</b>
18.1 Forberedelser . . . . .	46
18.2 Opprett ssb-project . . . . .	46
18.2.1 Uten GitHub-repo . . . . .	46
18.2.2 Med Github-repo . . . . .	48
18.3 Installere pakker . . . . .	48
18.4 Push til GitHub . . . . .	50
18.5 Bygg eksisterende ssb-project . . . . .	50
18.6 Rydd opp etter deg . . . . .	51
18.6.1 Lokalt . . . . .	51

18.6.2	Arkiver GitHub-repo . . . . .	51
18.7	Hva med R? . . . . .	54
<b>19</b>	<b>Git og Github</b>	<b>55</b>
19.1	Git  . . . . .	55
19.1.1	Hva er Git? . . . . .	55
19.1.2	Oppsett av Git . . . . .	56
19.1.3	Git og Notebooks . . . . .	56
19.1.4	Vanlige Git-operasjoner . . . . .	56
19.2	GitHub  . . . . .	56
19.2.1	Opprett GitHub-bruker . . . . .	57
19.2.2	To-faktor autentisering . . . . .	58
19.2.3	Koble deg til SSB . . . . .	61
19.2.4	Personal Access Token (PAT) . . . . .	64
<b>20</b>	<b>Virtuelle miljøer</b>	<b>68</b>
20.1	Python . . . . .	68
20.1.1	Anbefaling . . . . .	68
20.2	R . . . . .	68
<b>21</b>	<b>Jupyter-kernels</b>	<b>69</b>
<b>22</b>	<b>Installere pakker</b>	<b>70</b>
22.1	Python . . . . .	70
22.1.1	Poetry prosjekt eksempel . . . . .	70
22.1.2	Installering . . . . .	71
22.1.3	Avinstallering . . . . .	71
22.1.4	Oppgradere pakker . . . . .	71
22.1.5	Legge til kernel for poetry . . . . .	71
22.1.6	Fjerne kernel . . . . .	72
22.1.7	Sikkerhet . . . . .	72
22.2	R . . . . .	72
22.2.1	Installering . . . . .	72
22.2.2	Avinstallering . . . . .	74
22.2.3	Oppgradere pakker . . . . .	74
<b>23</b>	<b>Bruk av pyspark i virtuelle miljøer</b>	<b>75</b>
23.1	Legg til pyspark i det virtuelle miljøet . . . . .	75
23.2	Pakkehåndtering i pyspark . . . . .	75
23.2.1	Pyspark på lokal maskin . . . . .	75
23.2.2	Pyspark i et cluster . . . . .	76
<b>24</b>	<b>Samarbeid</b>	<b>77</b>

<b>25 Vedlikehold</b>	<b>78</b>
 <b>IV Jupyterlab på bakken</b>	 <b>79</b>
<b>26 Installere pakker</b>	<b>80</b>
26.1 Python . . . . .	80
26.1.1 Pip . . . . .	80
26.1.2 Poetry . . . . .	80
26.2 R . . . . .	80
26.2.1 Installering . . . . .	81
26.2.2 Avinstallering . . . . .	81
26.2.3 Oppgradere pakker . . . . .	81
<b>27 Lese inn filer</b>	<b>82</b>
27.1 sas7bdat . . . . .	82
27.2 Oracle . . . . .	82
27.3 Fame . . . . .	82
27.4 Tekstfiler . . . . .	82
27.5 Parquet . . . . .	82
 <b>V Avansert</b>	 <b>83</b>
<b>28 Lese filer fra bønne</b>	<b>84</b>
28.1 Eksempler . . . . .	84
28.2 Vanlige problemer . . . . .	84
28.2.1 Feil miljø . . . . .	84
28.2.2 Omstart av Jupyter . . . . .	85
28.2.3 Opprett TMS sak . . . . .	85
<b>29 IDE'er</b>	<b>86</b>
29.1 RStudio . . . . .	86
29.2 VSCode . . . . .	86
29.3 Pycharm . . . . .	86
<b>30 Scheduling</b>	<b>87</b>
<b>31 Databaser</b>	<b>88</b>
31.1 BigQuery . . . . .	88
31.2 CloudSQL . . . . .	88
<b>Referanser</b>	<b>89</b>

# Velkommen

Denne boken er ment som enkel-å-bruke manual for å ta i bruk SSBs nye dataplattform **Dapla**. Plattformen forsøker å gjøre statistikkprodusenter og forskere så selvhjulpne som mulig. Målsetningen er at tjenestene som tilbys skal kunne tas i bruk på en enkel og intuitiv måte. Men uansett hvor lett tilgjengelig tjenester er, og hvor mye arbeid som er lagt å gjøre løsninger brukervennlige, så trenger de fleste i SSB en klar og tydelig veiledning for hvordan de skal brukes og i hvilken større sammenheng tjenestene inngår. Dapla-manualen er ment å være en sånn støtte i statistikkernes hverdag. Uansett om man lurer på hvordan man logger seg inn på plattformen, eller om man ønsker informasjon om kjøremiljøet for mer kompliserte maskinlæringsmodeller, så skal man finne veiledning i denne manualen. Målgruppen er både nybegynneren og den mer erfarne.

**i** Denne boken er skrevet med [Quarto](https://quarto.org/) og er publisert på <https://statisticsnorway.github.io/dapla-manual/>. Alle ansatte i SSB kan bidra til boken ved kloner [dette repoet](#), gjøre endringer i en branch, og sende en pull request til administratorene av repoet (Team Statistikk-tjenester).

## Bidragstgere

Dapla-manualen er initiert og skrevet av **Team Statistikk-tjenester** i SSB. Bidragstgere er Øyvind Bruer-Skarsbø, Miles Winther, Bjørn Andre Skaar og Anders Lunde. Ved behov for oppdateringer og nytt innhold håper vi at alle i SSB kan bidra.

# Forord

Denne boken tar sikte på å gi SSB-ansatte mulighet til å ta i bruk grunnleggende funksjonalitet på DAPLA uten hjelp fra *eksperter*. Boken er bygget opp som den reisen vi mener en statistikker skal gjennom når de flytter sin produksjon fra bakke til sky<sup>1</sup>. Første del inneholder en del grunnleggende kunnskap som vi mener er viktig å ha før man skal starte å jobbe i skyen. Andre del forklarer hvordan man søker om å opprette et Dapla-team, en forutsetning for å drive databehandling på plattformen. Det vil ofte være første steget i ta i bruk plattformen, siden det er slik man får et sted å lagre data. Her forklarer vi hvilke tjenester som inkluderes i et statistikkteam og hvordan man bruker og administrerer dem. Den tredje delen tar utgangspunkt i at man skal starte å kode opp sin statistikkproduksjon eller kjøre eksisterende kode. *ssb-project* er et verktøy som er utviklet i SSB for å gjøre denne prosessen så enkel som mulig. Da kan brukerne implementere det som anses som god praksis i SSB med noen få tastetrykk, samtidig som vi også forklarer mer detaljert hva som skjer *under panseret*.

Det er tilrettelagt for en *treningsarena* i bakkemiljøet. Dette miljøet er nesten identisk med det som møter deg på Dapla, med unntak av at du her har tilgang til mange av de gamle systemene og mye mindre *hestekrefter* i maskinene. Ideen er at SSB-ere ofte vil ønske å lære seg de nye verktøyene<sup>2</sup> i kjente og kjære omgivelser først, og deretter flytte et ferdig skrevet produksjonsløp til Dapla. Del 4 av denne boken beskriver mer utfyllende hvordan dette miljøet skiller seg fra Dapla, og hvordan man gjør en del vanlige operasjoner mot de *gamle* bakkesystemene.

Siste delen av boken kaller vi **Avansert** og tar for seg ulike emner som mer avanserte brukere typisk trenger informasjon om. Her finner man blant annet informasjon om hvilke databaser man kan bruke og hvilke formål de er egnet for. Her beskrives også hvordan man kan bruke andre IDE-er enn Jupyterlab hvis man ønsker det. Tjenester for scheduledte kjøring av Notebooks blir også diskutert.

Forhåpentligvis senker denne boken terskelen for å ta i bruk Dapla. Kommentarer og ønsker vedrørende boken tas imot med åpne armer.

God fornøyelse

---

<sup>1</sup>I denne boken omtaler vi den gamle produksjonssonen, ofte kalt prodsonen, som **bakke**, og det nye skymiljøet Google Cloud som **sky**. Det er ikke helt presist men duger for formålene i denne boken.

<sup>2</sup>Det som omtales som *nye verktøy* er vil som regel bety R, Python, Git, GitHub og Jupyterlab.

# **Part I**

## **Introduksjon**



Målet med dette kapitlet er å gi en grunnleggende innføring i hva som legges i ordet **Dapla**. I tillegg gis en forklaring på hvorfor disse valgene er tatt.

# 1 Hva er Dapla?

Dapla står for **dataplattform**, og er en skybasert løsning for statistikkproduksjon og forskning.

## 2 Hvorfor Dapla?

Som dataplattform skal Dapla stimulerere til økt kvalitet på statistikk og forskning, samtidig som den gjør organisasjonen mer tilpasningsdyktig i møte med fremtiden.

**Den nye skybaserte dataplattformen (Dapla) skal bli viktig for å effektivisere arbeids-og produksjons**

Kilde: [Langtidsplan for SSB \(2022-2024\)](#)

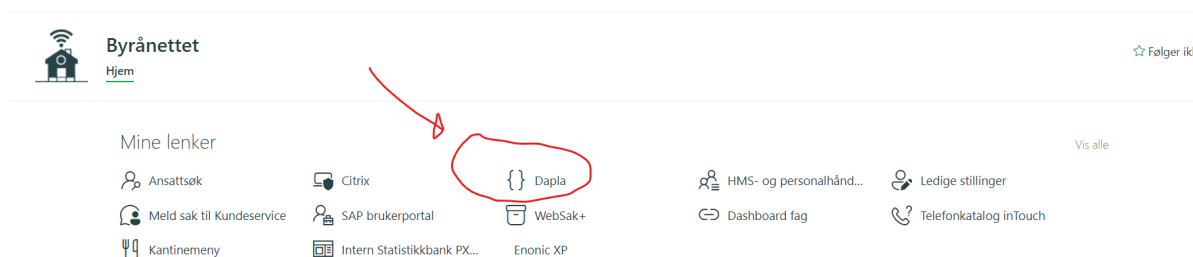
Målet med Dapla er å tilby tjenester og verktøy som lar statistikkprodusenter og forskere produsere resultater på en sikker og effektiv måte.

## 3 Arkitektur

Hvilke komponenter er plattformen bygd opp på? Forklart på lettest mulig måte.

## 4 Innlogging

Innlogging på Dapla er veldig enkelt. Dapla er en nettadresse som alle SSB-ere kan gå inn på hvis de er logget på SSB sitt nettverk. Å være logget på SSB sitt nettverk betyr i denne sammenhengen at man er logget på med **VPN**, enten man er på kontoret eller på hjemmekontor. For å gjøre det enda enklere har vi laget en fast snarvei til denne nettadressen [på vårt intranett/Byrånettet](#) (se Figur 4.1).



Figur 4.1: Snarvei til Dapla fra intranett

Men samtidig som det er lett å logge seg på, så er det noen kompliserende ting som fortjener en forklaring. Noe skyldes at vi mangler et klart språk for å definere bakkemiljøet og skymiljøet slik at alle skjønner hva man snakker om. I denne boken definerer bakkemiljøet som stedet der man har drevet med statistikkproduksjon de siste tiårene. Skymiljøet er den nye dataplattformen **Dapla** på Google Cloud.

Det som gjør ting litt komplisert er at vi har 2 Jupyter-miljøer på både bakke og sky. Årsaken er at vi har ett test- og ett prod-område for hver, og det blir i alt 4 Jupyter-miljøer. Figur 4.2 viser dette.

Hver av disse miljøene har sin egen nettadresse og sitt eget bruksområde.

### 4.1 Dapla

I de fleste tilfeller vil en statistiker eller forsker ønske å logge seg inn i prod-miljøet. Det er her man skal kjøre koden sin i et produksjonsløp som skal publiseres eller utvikles. I noen tilfeller hvor man ber om å få tilgjengeliggjort en ny tjeneste så vil denne først rulles ut i testområdet som vi kaller **staging-området**. Årsaken er at vi ønsker å beskytte prod-miljøet fra software

Bakkemiljø	Skymiljø/Dapla
PROD	PROD
STAGING	STAGING

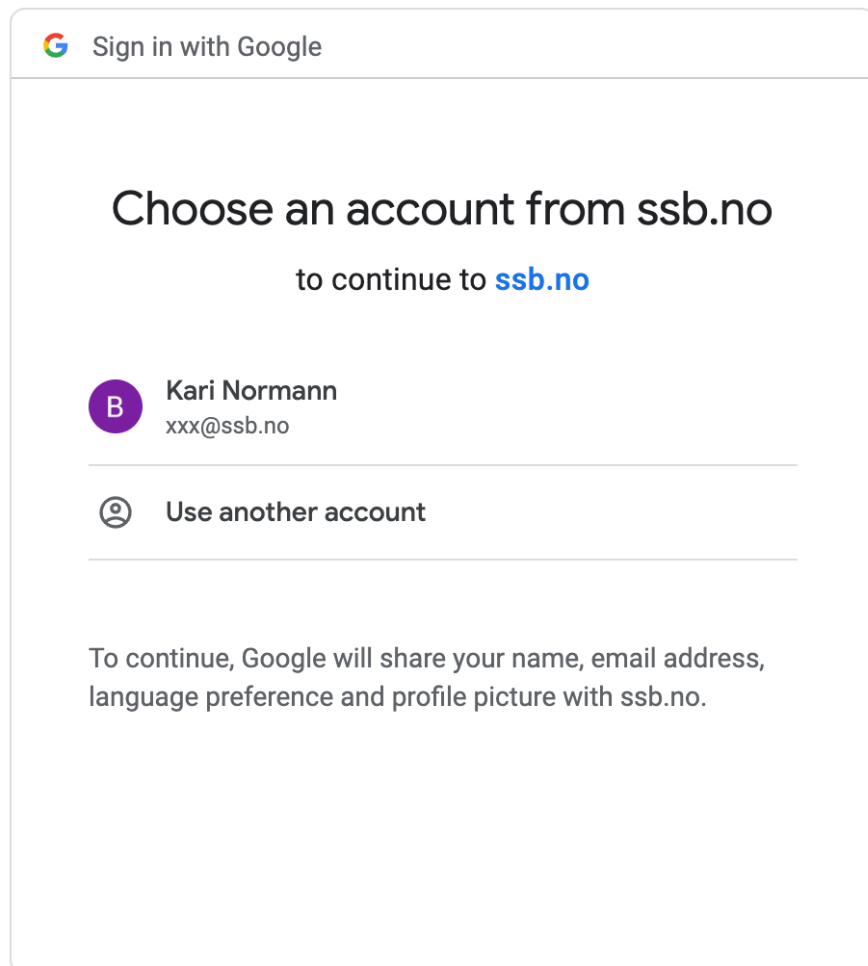
Figur 4.2: De 4 Jupyter-miljøene i SSB. Et test-miljø og et prod-miljø på bakke og sky/Dapla

som potensielt ødelegger for eksisterende funksjonalitet. Derfor ruller vi ut nye ting i staging først. Av den grunn vil de fleste oppleve å bli bedt om å logge seg inn der for testing en eller annen gang. Under forklarer vi hvordan man går frem for å logge seg på de to ulike miljøene på Dapla.

#### 4.1.1 Prod

For å logge seg inn i prod-miljøet på Dapla kan man gjøre følgende:

1. Gå inn på lenken <https://jupyter.dapla.ssb.no/> i en Chrome-nettleser eller klikk på lenken på Byrånettet som vist i Figur 4.1.
2. Alle i SSB har en Google Cloud-konto som må brukes når man logger seg på Dapla. Brukernavnet i Google er det samme som din *korte* epost-adresse (f.eks. cth@ssb.no). Hvis du ikke allerede er logget inn i Google vil du få spørsmål om å velge hvilken Google-konto som skal brukes (Figur 4.3). Logg inn med din Google-konto (ssb.no) og ditt AD-passord.
3. Deretter blir man spurt om man godtar at ssb.no (altså Dapla) kan bruke din Google Cloud-konto (Figur 4.4). Trykk **Allow**.
4. Deretter lander man på en side som lar deg avgjøre hvor mye maskinkraft som skal holdes av til deg (Figur 4.5). Det øverste alternativet er valgt som standard, og er tilstrekkelig for de fleste.
5. Vent til maskinen din starter opp (Figur 4.6). Oppstartstiden kan variere.





Figur 4.3: Velg en Google-konto

 Sign in with Google

## ssb.no wants to access your Google Account

 xxx@ssb.no

This will allow **ssb.no** to:

- See, edit, configure and delete your Google Cloud data and see the email address for your Google Account. 
- Manage your data in Cloud Storage and see the email address of your Google Account 

### Make sure that you trust ssb.no

You may be sharing sensitive info with this site or app. You can always see or remove access in your [Google Account](#).

Learn how Google helps you [share data safely](#).

See ssb.no's privacy policy and Terms of Service.

Cancel

Allow

Figur 4.4: Tillat at ssb.no får bruke din Google Cloud-konto



## Server Options

- ☒ **Jupyter arbeidsmiljø**

Standard arbeidsmiljø for Notebooks med Python og R på Dapla

- ☐ **Jupyter for kjøring av produksjonsløp**

**Serveren kan trenge ekstra oppstartstid.**

Denne jupyterinstansen gir garantier for mengde maskinkraft du får tildelt, noe som gir stabilitet og fart når du kjører et produksjonsløp.

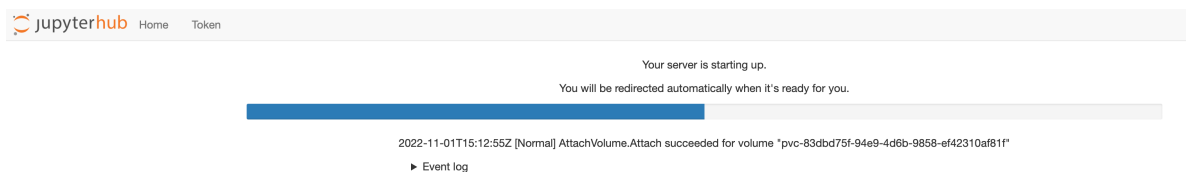
- ☐ **Jupyter stor maskin**

**NB! Denne er dyr å bruke, og skal KUN brukes for tunge beregninger. Serveren kan trenge ekstra oppstartstid.**

Denne jupyterinstansen gir ekstra mye minne og prosessorkraft, for spesielle arbeidsoppgaver hvor det kreves.

Start

Figur 4.5: Velg hvor mye maskinkraft du trenger



Figur 4.6: Starter opp Jupyter

Etter dette er man logget inn i et Jupyter-miljø som kjører på en minimal Ubuntu-maskin. Hvis man er del av [et Dapla-team](#) får man også tilgang til alt teamet har tilgang til.

### 4.1.2 Staging

Innlogging til staging-miljøet er identisk med innloggingen til prod-miljøet, med ett viktig unntak: nettadressen er nå <https://jupyter.dapla-staging.ssb.no/>.

Litt mer om hva som er tilgjengelig her kommer.

## 4.2 Bakkemiljøet

Jupyter-miljøet på bakken bruker [samme base-image](#)<sup>1</sup> for å installere Jupyterlab, og er derfor identisk på mange måter. Men innloggingen er ganske forskjellig.

### 4.2.1 Prod

Du logger deg inn på prod i bakkemiljøet på følgende måte:

1. Logg deg inn på Citrix-Windows i bakkemiljøet. Det kan gjøres ved å bruke lenken **Citrix** på Byrånettet, som også vises i Figur 4.1.
2. Åpne Chrome-nettleseren
3. Gå inn på denne nettadressen: <https://sl-jupyter-p.ssb.no/>
4. Skriv inn ditt brukernavn og passord som du bruker når du logger deg på maskinen din hver morgen.

### 4.2.2 Staging

Innlogging til staging-miljøet er identisk med innloggingen til prod-miljøet, med ett viktig unntak: nettadressen er nå <https://sl-jupyter-t.ssb.no/>.

---

<sup>1</sup>Hva er base-image?

# 5 Jupyterlab

## 5.1 Hva er Jupyterlab?

Mer kommer.

## 5.2 Terminalen

Må nevne operativsystemet og at noe programvare ligger installert her (git, jwsacruncher, quarto, ++)

## 5.3 Pakkeinstallasjoner

Noe er i base-image, noe bør gjøres i virtuelle miljøer. Hvordan liste ut pakker som er pre-installert?

## 5.4 Extensions

Jupyterlab er en samling extension. Kan bare installeres av admin. Sikkerhet. Hvilke extension har vi tilgjengeliggjort?

## 5.5 Tips & triks

Sane defaults for Jupyterlab.

## **6 Bakke vs. sky**

## **Part II**

# **Opprette Dapla-team**

I forrige del beskrev vi noen grunnleggende ting rundt **Dapla**. I denne delen tar vi for oss hvordan du kan begynne å jobbe med skarpe data på plattformen.

Kapittelet som beskriver hvordan man [logger seg inn på Dapla](#) vil fungere uten at du må gjøre noen forberedelser. Er man koblet på SSB sitt nettverk så vil alle SSB-ansatte kunne gå inn på plattformen og kode i Python og R. Men du har ikke tilgang til SSBs område for datalagring på plattformen. I praksis vil det si at man generere data med kode, men man ikke jobbe med skarpe data.

For å få muligheten til å jobbe skarpe data MÅ du først opprette et **dapla-team**. Dette er det første naturlige steget å ta når man skal begynne å jobbe med statistikkproduksjon på dapla. I dette kapittelet vil vi forklare det du trenger å vite om det å opprette og jobbe innenfor et team.

## 7 Hva er Dapla-team?

Et Dapla-team fokuserer på statistikkproduksjon innen et eller flere emneområder på Dapla. Teamet er egentlig et arbeidsområde på Dapla, som gir medlemmene av teamet tilgang på teamet sine felles datalagre, roller og bakke-sky synkroniseringsområder.

Hvert Dapla-team får opprettet et prosjektområde i Google Cloud Platform (GCP), som er SSBs leverandør av skytjenester.

## 8 Opprette Dapla-team

For å komme i gang med å opprette et Dapla-team trengs det en oversikt over teamets medlemmer og hvilke tilgangsgrupper medlemmene skal være med i. Det trengs også informasjon om hvilke Dapla-tjenester som er aktuelle for teamet å ta i bruk. Derfor har det blitt opprettet en egen veileder for dette kalt *Dapla Start*.

**i** Gå til [Dapla Start](#) for starte bestilling av et nytt Dapla-team.

Når teamet er opprettet får alle medlemmene tilgang til sitt eget prosjekt i Google Cloud Platform (GCP), som er SSBs leverandør av skytjenester. Videre får hvert prosjekt et sett med tjenester og tilganger som knyttes til teamet. Det opprettes også datalagringsområder (kalt [bøtter](#)) som bare kan aksesseres av brukere som er med i teamets tilgangsgrupper.

Dapla-teamet vil også få sin egen gruppe i SSBs Active Directory slik at medlemskapet i gruppen kan administreres av Kundeservice.



## 9 Bøtter

Hvert statistikkteam har sitt eget datalager som heter Google Cloud Storage (GCS). Disse er delt inn i flere datalagringsområder som kalles *bøtter*. Dette kan sees på som et filsystem som kan organiseres med flere undermapper og filer. Navnet på bøttene må være unikt på tvers av alle Dapla-team. Derfor blir disse opprettet etter en navnekonvensjon basert på kjøremiljø, teamnavn og hvilke data bøtta skal inneholde.

### 9.1 Bøttenavn i Produksjonsmiljøet

- **ssb-prod-teamnavn-data-kilde:** Inneholder pseudonymiserte rådata fra datakildene
- **ssb-prod-teamnavn-data-produkt:** Inneholder data knyttet til statistikkproduktet, med følgende underkataloger:
  - *inndata*
  - *klargjorte-data*
  - *statistikk*
  - *utdata*
- **ssb-prod-teamnavn-data-delt:** Inneholder data knyttet til statistikkproduktet som kan deles med andre statistikkteam. Disse vil ha følgende underkataloger:
  - *inndata*
  - *klargjorte-data*
  - *statistikk*
  - *utdata*

**i** Underkatalogene *inndata*, *klargjorte-data*, *statistikk* og *utdata* gjenspeiler SSBs datatilstander. Se [Datatilstander i SSB](#) for mer informasjon.

## 9.2 Andre miljøer

På samme måte som i produksjonsmiljøet finnes det bønner for utviklings- og testformål:

- **ssb-staging-*teamnavn*-data-kilde**
- **ssb-staging-*teamnavn*-data-produkt**
- **ssb-staging-*teamnavn*-data-delt**

## 9.3 Fellesbønner

I tillegg til disse finnes det noen bønner med data som kan deles med alle i SSB og som kan brukes til kurs og opplæring (bl.a. denne manualen). Disse bønnene er:

- **ssb-prod-dapla-data-delt**
- **ssb-staging-dapla-data-delt**

# 10 Google Cloud Console

Google Cloud er SSBs leverandør av skytjenester som Dapla er bygget på.

Google Cloud Console er et web-basert grensesnitt for å administrere ressurser og tjenester på Google Cloud. For å bruke denne må man ha en Google-konto. Alle i SSB har en konto knyttet opp mot Google.

**i** Gå til [Google Cloud Console](#) og logg på med din SSB-bruker.

# 11 Overføre data

For å overføre data mellom bakke og sky brukes **Data Transfer**, som er en tjeneste i [Google Cloud Console](#). Denne tjenesten kan brukes til å flytte data både til og fra Linuxstammen og Dapla, og er tilgjengelig for teamets kildedataansvarlige.

Teamets kildedataansvarlige vil være spesifisert som en del av å [opprette et Dapla-team](#).

## 11.1 Forberedelser

### 11.1.1 Logg inn i produksjonssonen

Logg inn i produksjonssonen [fra denne linken](#). Dette krever pålogging med SSBs brukernavn og passord, samt tofaktorautentisering.

## 11.2 Få tilgang til Linuxstammen

For å få tilgang til Linuxstammen må du først koble deg opp mot en Linux-maskin. Dette kan gjøres på følgende måte:

- Åpne FileZilla
- Velg **Fil > Tjeneroppsett** fra menyen
- Under Tjeneroppsett velg mappen **Tjenere**. Trykk på **Ny Tjener** og skriv inn et valgfritt navn.
- I feltene til høyre må du fylle inn **Protokoll**. Velg **SFTP - SSH File Transfer Protocol**
- I feltet **Vert** må du skrive inn en adresse til en Linux-maskin som du har tilgang til (prøv f.eks `sl-sas-work-1.ssb.no` ).
- I feltet **Port** skriv inn 22
- Nedtrekkslisten **Innloggingstype** kan være satt til **Normal**
- I feltet **Bruker** skriv inn ditt 3-bokstavers brukernavn (samme som i eposten din)
- I feltet **Passord** skriver du inn ditt SSB-passord (fra AD)
- Trykk **Koble til** og OK. Du vil kanskje få opp et spørsmål med **Ukjent vertsnøkkel**: Tjenerens vertsnøkkel er ukjent etc. Kryss av **Godta alltid denne tjeneren** og trykk OK.

- Vent på at FileZilla får koblet opp og skriver ut meldingen **Status: Connected to <machine-name>**
- Du kan nå navigere i filsystemet til Linuxstammen under **Ekstern tjener**

FileZilla kan også brukes til å kopiere filer mellom andre nettverksdisker og Linuxstammen.

### 11.2.1 Få tilgang til overføringsmapper

Mapper som brukes til å overføre data mellom produksjonssonen og Dapla ligger på Linuxstammen.

Hvert team vil ha mappene `/ssb/cloud_sync/<teamnavn>/data/tilsky` og `/ssb/cloud_sync/<teamnavn>/data/frasky` (erstatt `<teamnavn>` med navnet på ditt Dapla-team, f.eks “kostra”, “skatt-naering”, “gro-grunnskole”, etc.).

Filer som flyttes fra produksjonssonen *til Dapla* legges i mappen **tilsky** og filer som skal flyttes *fra Dapla* til produksjonssonen havner i mappen **frasky**. Det er viktig og huske at filer som flyttes enten opp til skyen eller ned til bakken vil bli slettet fra den respektive mappen når overførselen er gjennomført.

## 11.3 Sette opp overføringsjobber

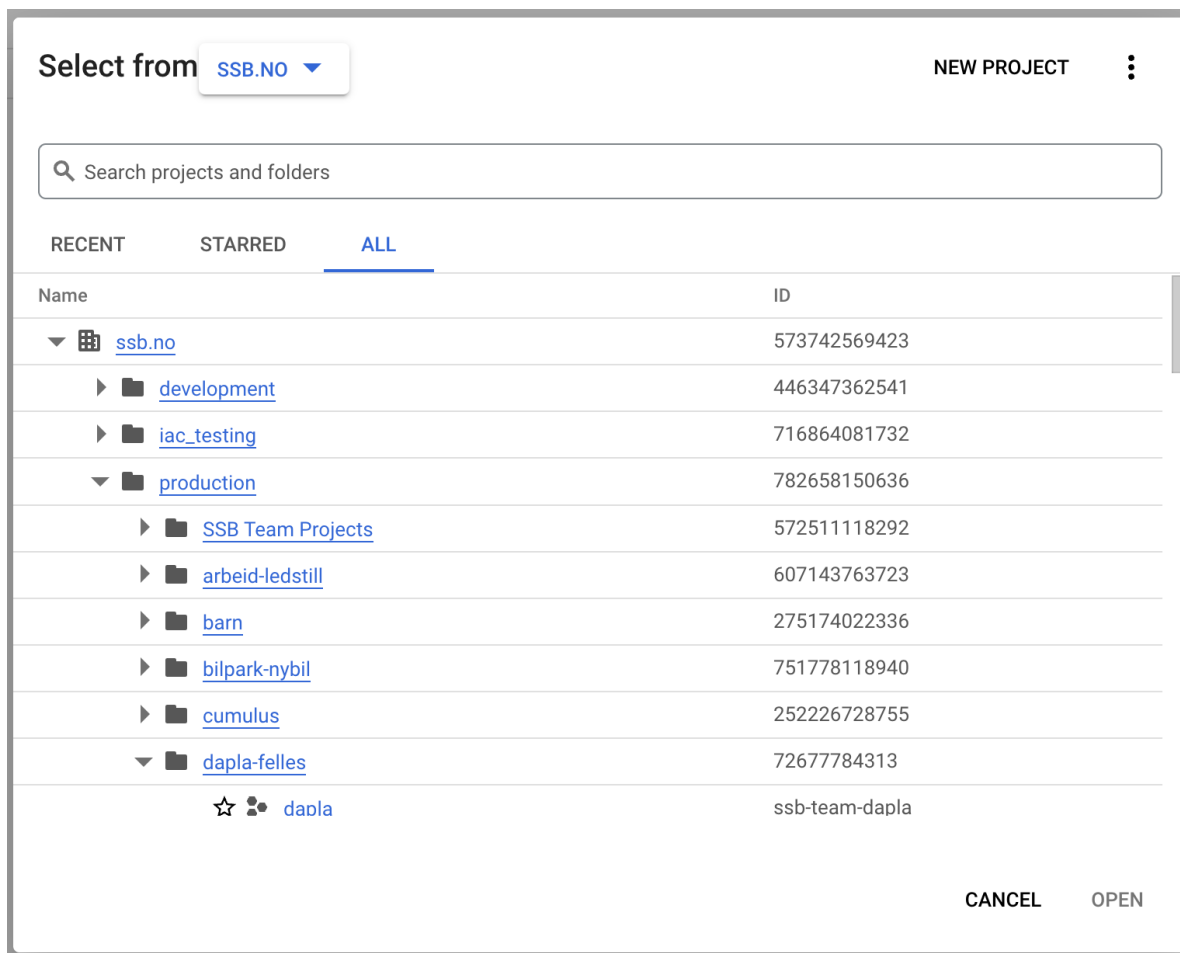
Enten man skal overføre filer opp til sky eller ned til bakken så bruker man den samme Data Transfer tjenesten. For å få tilgang til denne må man først logge seg inn i [Google Cloud Console](#). Sjekk at du er logget inn med din SSB-konto (xxx@ssb.no).

Øverst på siden, til høyre for teksten **Google Cloud** finnes det en prosjektvelger, og her er det viktig å velge korrekt Google prosjekt. Hvis du trykker på prosjektvelgeren vil det åpnes opp et nytt vindu. Sjekk at det står **SSB.NO** øverst i dette vinduet. Trykk deretter på fanen **ALL** for å få opp alle tilgjengelige Google-prosjekter under organisasjonen **ssb.no** (Figur 11.1)

Under **ssb.no** vil det ligge flere mapper. Åpne mappen som heter **production** og let frem en undermappe som har navnet på ditt Dapla-team. Strukturen skal se slik ut:

```
ssb.no
  production
    <teamnavn>
      prod-<teamnavn>
      <teamnavn>-ts
```

Det underste nivået (`prod-<teamnavn>` og `<teamnavn>-ts`) viser prosjektene, nivået i mellom er mapper, og toppnivået er organisasjonen (ssb.no). Prosjektet `<teamnavn>-ts` er et separat



Figur 11.1: Prosjektvelgeren i Google Cloud Console

prosjekt som bare teamets kildedataansvarlige har tilgang til, og det er her tjenesten Data Transfer skal settes opp.

- Velg derfor prosjektet `<teamnavn>-ts`.
- I søkefeltet til Google Cloud Console, skriv **Data transfer** og trykk på det valget som kommer opp.
- Første gang man kommer inn på siden til Transfer Services vil man bli vist en blå knapp med teksten **Set Up Connection**. Når du trykker på denne vil det dukke opp et nytt felt hvor du får valget **Create Pub-Sub Resources**. Dette er noe som bare trengs å gjøre én gang. Trykk på den blå **CREATE** knappen, og deretter trykk på **Close** lenger nede.
- I navigasjonsmenyen til venstre trykk **Transfer jobs**, og deretter trykk på **+ Create transfer job** øverst på siden for å opprette en ny overføringsjobb.

### 11.3.1 Overføring fra Linuxstammen til Dapla

Følgende oppskrift tar utgangspunkt i siden **Create a transfer job** (Figur 11.2):

The screenshot shows the 'Create a transfer job' page in the Google Cloud Console. On the left is a sidebar with navigation links: 'Data Transfer', 'Transfer jobs', 'Agent pools', 'Transfer Appliance | request', and 'Transfer Appliance | monitor'. The main area is titled 'Create a transfer job' and features a 'Get started' section with a five-step progress indicator. Step 1, 'Get started', is highlighted and includes the subtext 'POSIX filesystem to Google Cloud Storage'. Below the steps are 'CREATE' and 'CANCEL' buttons. To the right, the 'Get started' section provides instructions: 'To optimise this form for your transfer needs, select the type of source and destination that you'll use for this transfer job.' It contains two dropdown menus: 'Source type' (set to 'POSIX filesystem') and 'Destination type' (set to 'Google Cloud Storage'). Below these is a link 'LOOKING FOR MORE OPTIONS?' and a 'NEXT STEP' button.

Figur 11.2: Opprett overføringsjobb i Google Cloud Console

1. Velg **POSIX filesystem** under “Source type” og **Google cloud storage** under “Destination type” (eller motsatt hvis overføringsjobben skal gå fra Dapla til Linuxstammen). Trykk **Next step**
2. Nedtrekkslisten “Agent pool” skal normalt bare ha ett valg: **transfer\_service\_default**. Velg denne.
3. I feltet “Source directory path” skal man kun skrive **data/tilsky** siden overføringsagenten kun har tilgang til mapper som ligger relativt plassert under **/ssb/cloud\_sync/<teamnavn>/**. Trykk **Next step**

4. Velg en destinasjon for overføringsjobben. Trykk på **Browse** og velg bøtten med navn som passer til `ssb-prod-<teamnavn>-data-synk-opp`. Vi anbefaler at du også oppretter en mappe inne i denne bøtten. Det gjøres ved å trykke på mappeikonet med et **+**-tegn foran. Skriv inn et passende mappenavn og trykk **Select** i bunnen av siden. Trykk deretter **Next step**
5. Neste steg “Choose how and when to run this job” er opp til brukeren å bestemme. Hvis man f.eks. velger at Data Transfer skal overføre data en gang i uken, vil den kun starte en overføring hvis det finnes nye data. Trykk **Next step**
6. Beskriv overføringsjobben, f.eks: “Flytter data for til sky.”. Resten av feltene er opp til brukeren å bestemme. Standardverdiene er OK.

Trykk til slutt på den blå **Create**-knappen. Du vil kunne se kjørende jobber under menyen **Transfer jobs**.

For å sjekke om data har blitt overført, skriv inn **cloud storage** i søkefeltet øverst på siden og trykk på det første valget som kommer opp. Her vil du finne en oversikt over alle teamets bølter, deriblant en med navn `ssb-prod-<team-name>-data-synk-opp`. Når overføringsjobben er ferdig vil du kunne finne igjen dataene i den mappen som ble definert i stegene overnfor.

### 11.3.2 Overføring fra Dapla til Linuxstammen

Overføringsjobben settes opp nesten identisk med [Overføring fra Linuxstammen til Dapla](#) med unntak av følgende:

- Steg 1: Velg **Google cloud storage** under “Source type” og **POSIX filesystem** under “Destination type”
- Steg 2: Velg bøtten `ssb-prod-<team-name>-data-synk-ned`
- Step 3: Velg **transfer\_service\_default** som “Agent pool” og skriv `data/frasky` inn i feltet for “Destination directory path”.

For å se om data har blitt overført til Linuxstammen må du nå gå til mappen `/ssb/cloud_sync/<team-name>/data/frasky` fra FileZilla.

Husk: Du kan alltid gå tilbake og se på tidligere fullførte jobber, og starte en overføringsjobb manuelt fra menyen **Transfer jobs**.



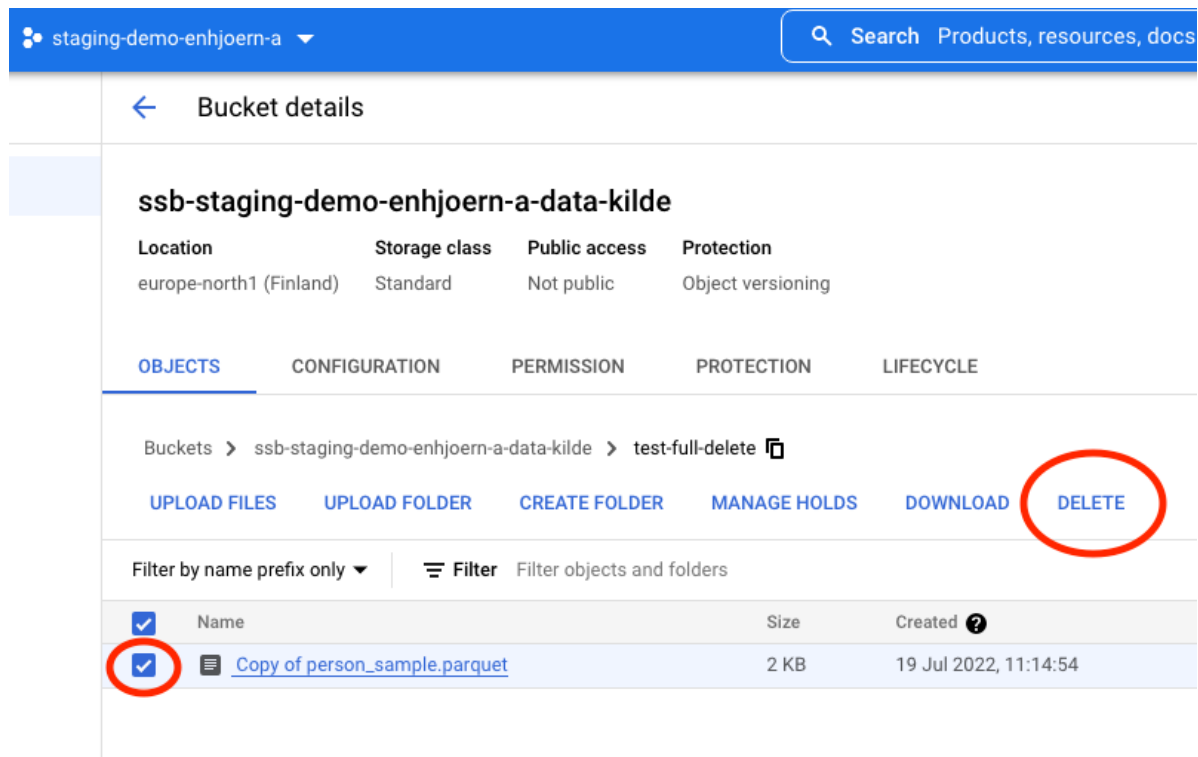
## 12 Lagre data

## 13 Hente data

## 14 Slette data fra bølter

Sletting av filer og mapper fra bølter kan gjøres fra [Google Cloud Console](#). Søk opp “Cloud Storage” i søkefeltet og klikk på den bølten hvor filen er lagret under “Buckets”.

Kryss av filen/katalogen som du ønsker å slette og trykk “Delete” (Figur 14.1)

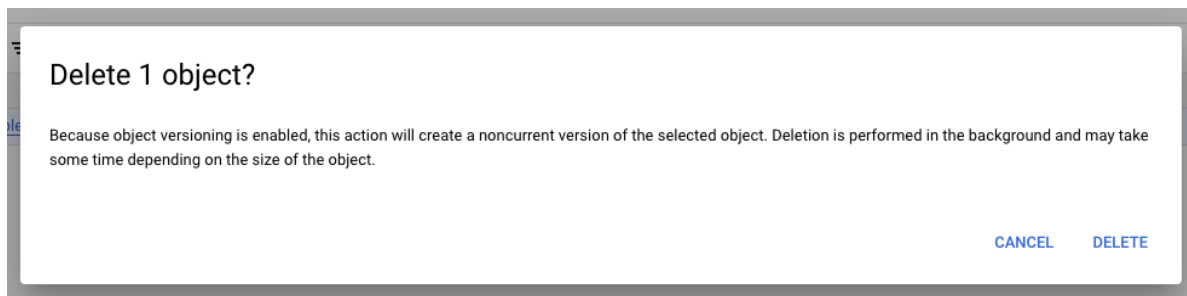


Figur 14.1: Sletting av en fil

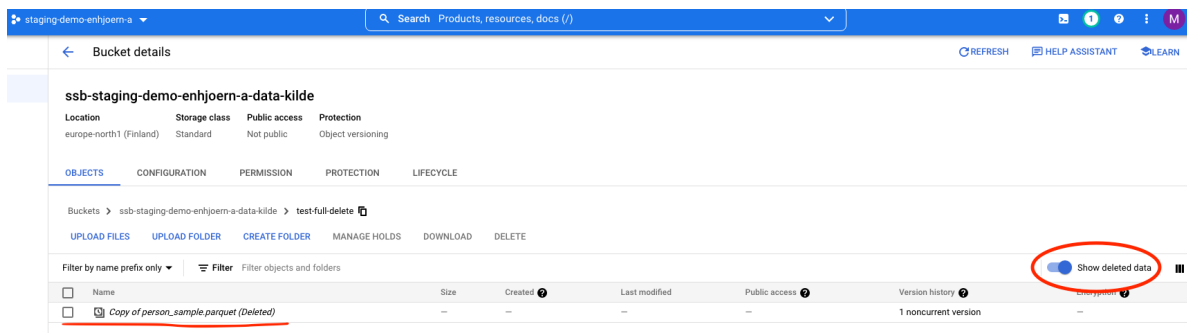
Siden bølter på Dapla har versjonering får man opp en dialogboks som informerer om at objektet (dvs. filen) er versjonert (Figur 14.2). Trykk på “Delete”.

Slettingen kan ta noe tid. Når denne er ferdig vil filen være slettet, men den kan fortsatt gjenopprettes. Hvis du ønsker at filen skal slettes *permanent*, gjør følgende:

1. Skru på visning av slettede filer med å bruke radioknappen “Show deleted data” (Figur 14.3)

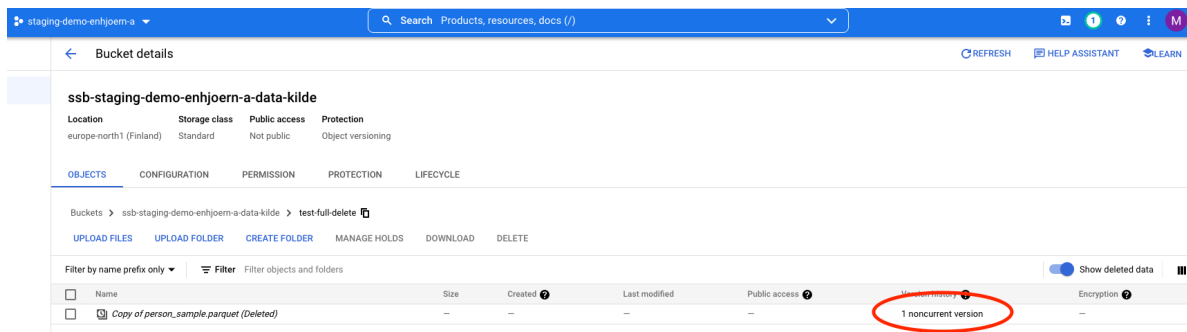


Figur 14.2: Bekreft sletting av fil

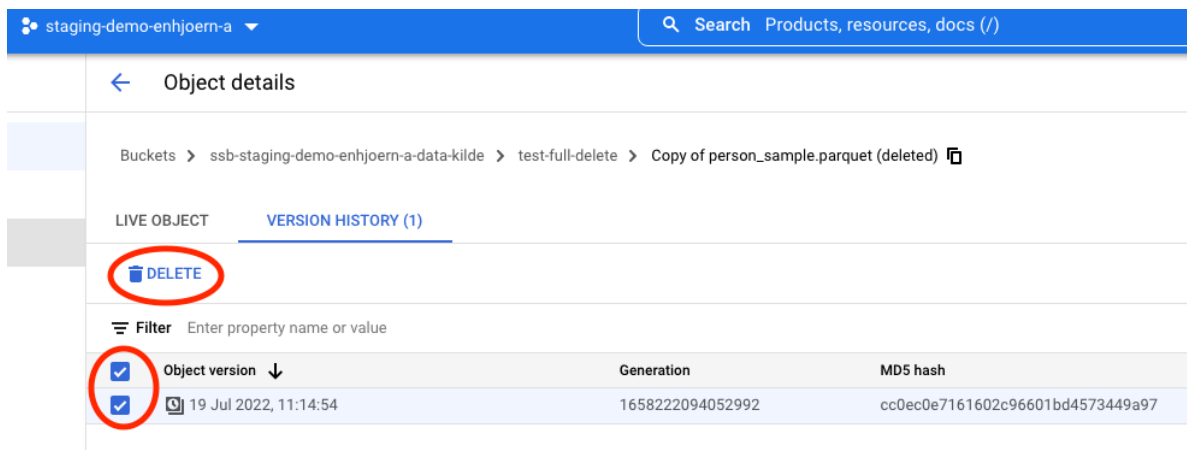


Figur 14.3: Skru på visning av slettede filer

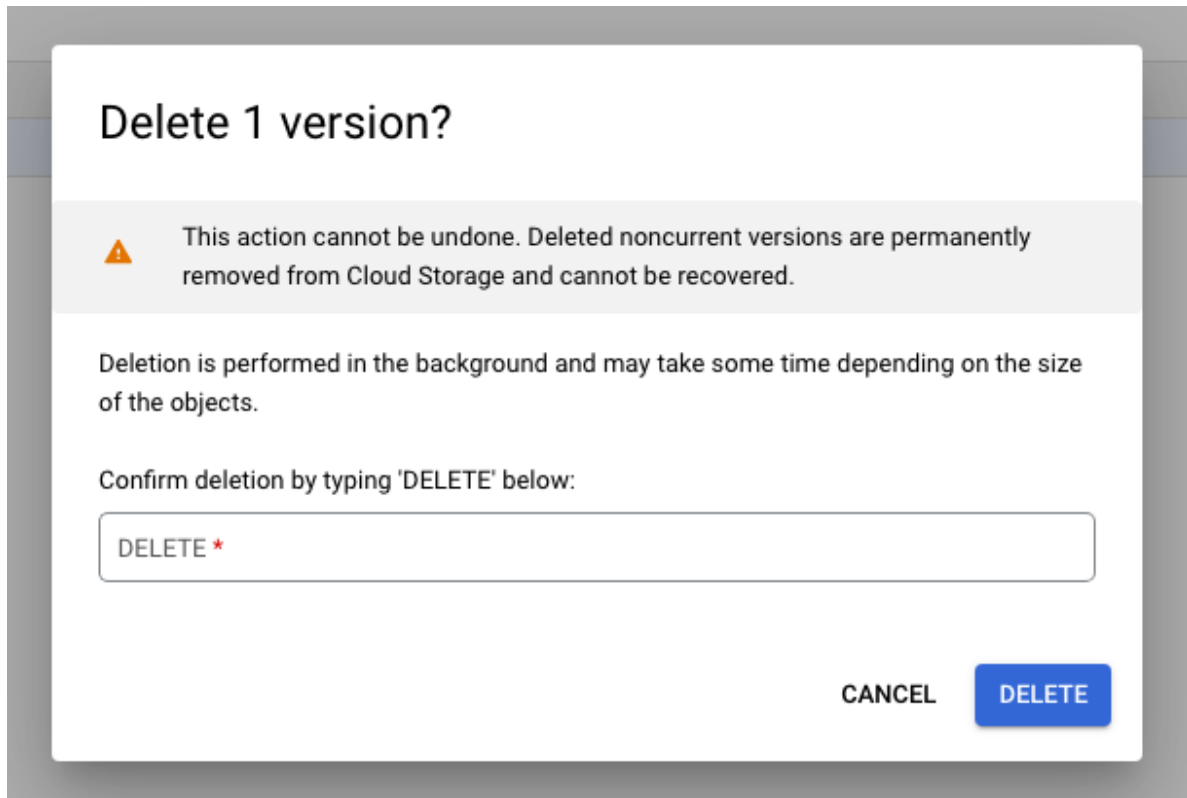
2. Finn frem til den slettede filen og trykk på linken “1 noncurrent version” eller tilsvarende (Figur 14.4). Dette vil ta deg direkte til en side som viser filens versjonshistorikk.
3. Velg alle versjoner som vist på Figur 14.5 og trykk “Delete”
4. Til slutt må man bekrefte at man ønsker å slette alle versjoner (Figur 14.6) med å skrive inn DELETE og trykke på den blå “Delete”-knappen:



Figur 14.4: Velg versjonshistorikk



Figur 14.5: Slett alle versjoner



Figur 14.6: Bekreft sletting av alle versjoner

# 15 Gjenopprette data fra bøtter

Alle bøtter har automatisk versjonering. Dette gjør det mulig å tilbakeføre filer til en tidligere versjon eller gjenopprette filer som er slettet ved et uhell.

Logg inn på [Google Cloud Console](#) og søk opp “Cloud Storage” i søkefeltet. Klikk på den bøtten hvor filen er lagret under “Buckets”.

## 15.1 Gjenopprette en slettet fil

Fra Cloud Storage skjermbildet kan man navigere seg frem til den mappen hvor filen tidligere er lagret og skru på radioknappen “Show deleted data” (Figur 15.1)

Cloud Storage

Buckets

Monitoring

Settings

← Bucket details

REFRESH

HELP ASSISTANT

LEARN

ssb-staging-dapla-felles-data-delt

Location

europa-north1 (Finland)

Storage class

Standard

Public access

Not public

Protection

Object versioning

OBJECTS

CONFIGURATION

PERMISSION

PROTECTION

LIFECYCLE

Buckets > ssb-staging-dapla-felles-data-delt

UPLOAD FILES

UPLOAD FOLDER

CREATE FOLDER

TRANSFER DATA

MANAGE HOLDS

DOWNLOAD

DELETE

Filter by name prefix

Filter

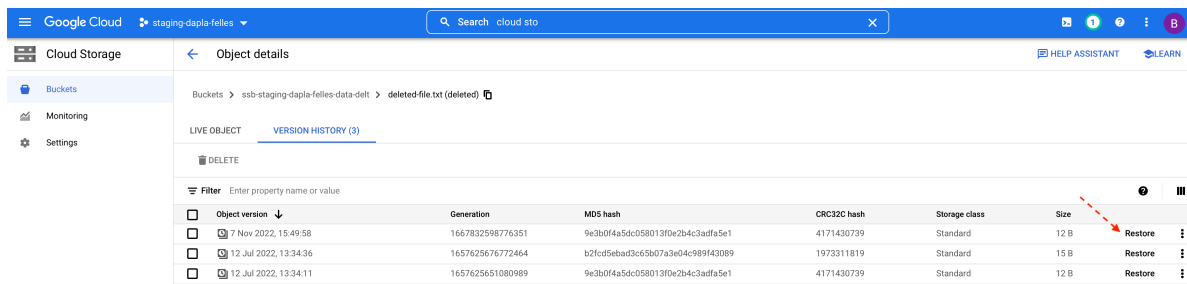
Filter objects and folders

<input type="checkbox"/>	Name	Size	Type	Created	Storage class	Last modified	Public access	Version history	Encryption	Retention expiry date	Holds	
<input type="checkbox"/>	R_arnoke_test/	--	Folder	--	--	--	--	--	--	--	--	⋮
<input type="checkbox"/>	Recovery-data-demo/	--	Folder	--	--	--	--	--	--	--	--	⋮
<input type="checkbox"/>	SSB_logo_black.svg	8.4 KB	image/svg+xml	19 May 20...	Standard	19 May 20...	Not public	--	Google-managed key	--	None	⋮
<input type="checkbox"/>	artifact-registry/	--	Folder	--	--	--	--	--	--	--	--	⋮
<input type="checkbox"/>	ba-export/	--	Folder	--	--	--	--	--	--	--	--	⋮
<input type="checkbox"/>	data.txt	102.3 KB	text/plain	7 Nov 20...	Standard	7 Nov 202...	Not public	1 noncurrent version	Google-managed key	--	None	⋮
<input type="checkbox"/>	data2.txt (Deleted)	--	--	--	--	--	--	1 noncurrent version	--	--	--	⋮
<input type="checkbox"/>	datadoc/	--	Folder	--	--	--	--	--	--	--	--	⋮
<input type="checkbox"/>	deleted-file.txt (Deleted)	--	--	--	--	--	--	3 noncurrent versions	--	--	--	⋮
<input type="checkbox"/>	felles/	--	Folder	--	--	--	--	--	--	--	--	⋮
<input type="checkbox"/>	homer-brain.png (Deleted)	--	--	--	--	--	--	1 noncurrent version	--	--	--	⋮
<input type="checkbox"/>	sqlite/	--	Folder	--	--	--	--	--	--	--	--	⋮
<input type="checkbox"/>	test/	--	Folder	--	--	--	--	--	--	--	--	⋮
<input type="checkbox"/>	tdsseries/	--	Folder	--	--	--	--	--	--	--	--	⋮
<input type="checkbox"/>	villain-ssb.jpg	60.4 KB	image/jpeg	19 May 20...	Standard	19 May 20...	Not public	--	Google-managed key	--	None	⋮

Rows per page: 50 1 - 15 of 15

Figur 15.1: Skru på visning av slettede filer

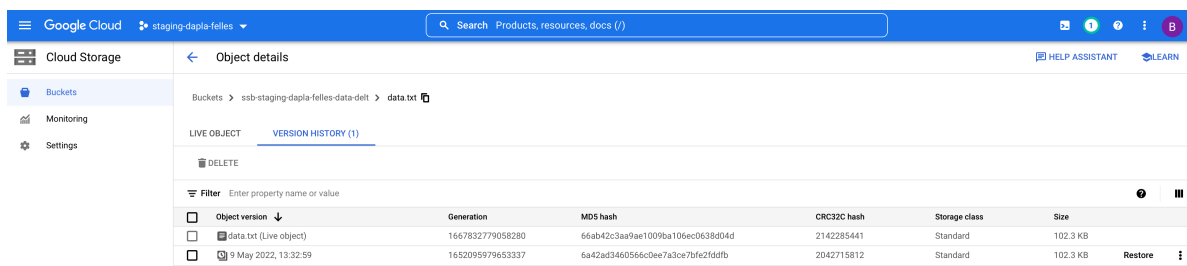
Nå vil man kunne se slettede filer i kursiv med teksten *(Deleted)* på slutten. Kolonnen “Version history” vil også vise hvor mange tidligere versjoner som finnes av denne filen. Trykk på filnavnet du ønsker å gjenopprette og velg deretter fanen “Version history”. I listen av versjoner til denne filen har man mulighet til å gjenopprette til en tidligere versjon ved å klikke på “Restore” (Figur 15.2).



Figur 15.2: Gjenoppretting av en slettet fil

## 15.2 Gjenopprette en fil til en tidligere versjon

Fra Cloud Storage skjermbildet kan man navigere seg frem til den mappen hvor filen er lagret, og trykke på filnavnet. Velg deretter fanen “Version history”. I listen av versjoner til denne filen har man mulighet til å gjenopprette til en tidligere versjon ved å klikke på “Restore” (Figur 15.3).



Figur 15.3: Versjonshistorikk til en fil



## **16 Fra bakke til sky**

## **17 Administrasjon av team**

### **17.1 Legge til eller fjerne medlemmer**

Dapla-teamet vil ha sitt eget gruppenavn i SSBs Active Directory. Endringer i denne gruppen, slik som å legge til eller fjerne medlemmer, gjøres ved å sende en epost til Kundeservice.

**Part III**

**SSB-project**

I forrige del forklarte vi hvordan man jobber med skarpe data på Dapla. Det neste steget vil ofte være å begynne å utvikle kode i Python og/eller R. Dette innebærer at man helst skal:

- versjonshåndtere med **Git**
- opprette et **GitHub**-repo
- opprette et **virtuelt miljø** som husker hvilke versjoner av pakker og programmeringsspråk du brukte

I tillegg må alt dette konfigureres for hvordan SSB sine systemer er satt opp. Dette har vist seg å være unødvendig krevende for mange. Team Statistiktjenester har derfor utviklet et program som gjør alt dette for deg på en enkel måte som heter **ssb-project**.

Vi mener at **ssb-project** er et naturlig sted å starte når man skal bygge opp koden i Python eller R. Det gjelder både på bakken og på sky. I denne delen av boken forklarer vi først hvordan du bruker **ssb-project** i det første kapitlet. Siden programmet skjuler mye av kompleksiteten rundt dette, så bruker vi de andre kapitlene til å forklare hvordan man ville satt opp dette uten hjelp av programmet. Dermed vil det være lett for SSB-ansatte å skjønne hva som gjøres og hvorfor det er nødvendig.

## 18 Nytt ssb-project

I dette kapittelet forklarer vi hvordan du oppretter et **ssb-project** og hva det innebærer. **ssb-project** er et CLI<sup>1</sup> for å raskt komme i gang med koding på Dapla, hvor en del SSB-spesifikke beste-prakiser er ivaretatt. Kode som naturlig hører sammen, f.eks. koden til et produksjonsløp for en statistikk, er målgruppen for dette programmet. Kort fortalt kan du kjøre denne kommandoen i en terminal

```
ssb-project create stat-testprod
```

og du vil få en mappe som heter **stat-testprod** med følgende innhold:

1. **Standard mappestruktur**

En standard mappestruktur gjør det lettere å dele og samarbeide om kode, som igjen reduserer sårbarheten knyttet til at få personer kjenner koden.

2. **Virtuelt miljø**

Virtuelle miljøer isolerer og lagrer informasjon knyttet til kode. For eksempel hvilken versjon av Python du bruker og tilhørende pakkeversjoner. Det er viktig for at publiserte tall skal være reproduserbare. Verktøyet for å lage virtuelt miljø er [Poetry](#).

3. **Versjonshåndtering med Git**

Initierer versjonshåndtering med [Git](#) og legger til SSBs anbefalte *.gitignore* og *.gitattributes*. Det sikrer at du ikke versjonshandterer filer/informasjon som ikke skal versjonshåndteres.

I tillegg lar **ssb-project** deg opprette et GitHub-repo hvis du ønsker. Les mer om hvordan du kan ta i bruk dette verktøyet under.

### Merk

Dokumentasjonen for **ssb-project** finnes her: <https://statisticsnorway.github.io/ssb-project-cli/>. Det oppdateres hver gang en ny versjon av **ssb-project** slippes.

---

<sup>1</sup>CLI = Command-Line-Interface. Dvs. et program som er skrevet for å brukes terminalen ved hjelp av enkle kommandoer.


## 18.1 Forberedelser

Før du kan ta i bruk **ssb-project** så er det et par ting som må være på plass:

1. Du må ha opprettet en **git-bruker** og **git-epost** lokalt der du skal kalle på programmet ([les mer om hvordan her](#)).
2. Hvis du ønsker at **ssb-project** også skal opprette et GitHub-repo for deg må du også følgende være på plass:
  - a. Du må ha en **GitHub-bruker** ([les hvordan her](#))
  - b. Skru på **2-faktor autentifisering** for GitHub-brukeren din ([les hvordan her](#))
  - c. Være koblet mot SSBs organisasjon [statisticsnorway](#) på GitHub ([les hvordan her](#))
  - d. Opprette **Personal Access Token (PAT)** og godkjenne det for bruk mot **statisticsnorway** ([les hvordan her](#))

Det er også å anbefale at du [lagrer PAT lokalt](#) slik at du ikke trenger å forholde deg til det når jobber med Git og GitHub. Hvis du har alt dette på plass så kan du bare fortsette å følge de neste kapitlene.

## 18.2 Opprett ssb-project

 Har du Github bruker? Noe funksjonalitet i **ssb-project** krever det. Finn ut hvordan ved å lese [forrige kapittel](#).

**ssb-project** lar deg opprette en prosjekt-mappe med og uten GitHub-repo. La oss ta for oss hver av alternativene.

### 18.2.1 Uten GitHub-repo

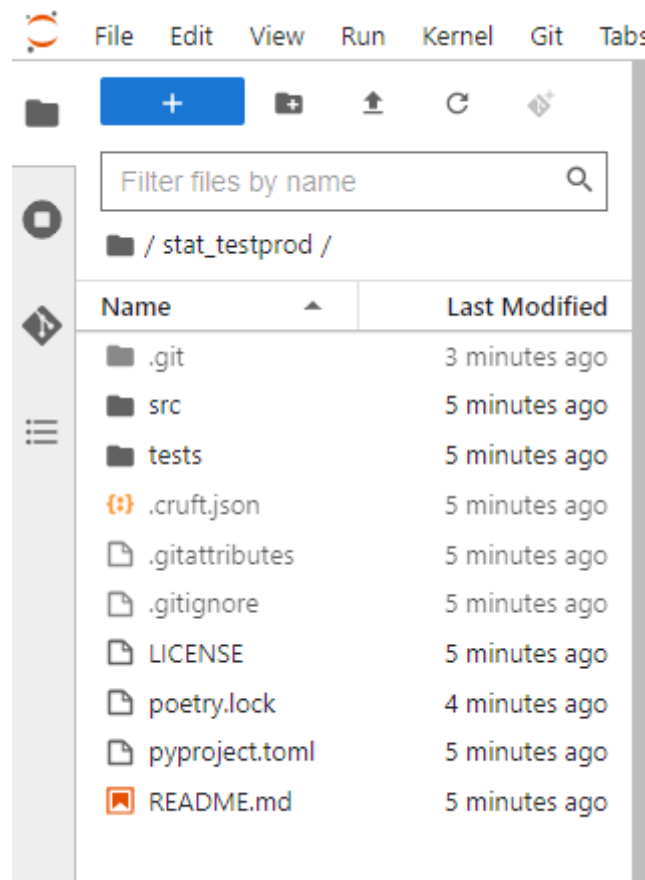
For å opprette et nytt **ssb-project** uten GitHub-repo gjør du følgende:

1. Åpne en terminal. De fleste vil gjøre dette i Jupyterlab på bakke eller sky og da kan de bare trykke på det blå -tegnet i Jupyterlab og velge **Terminal**.
2. Før vi kjører programmet må vi være obs på at **ssb-project** vil opprette en ny mappe der vi står. Gå derfor til den mappen du ønsker å ha den nye prosjektmappen. For å opprette et prosjekt som heter **stat-testprod** så skriver du følgende i terminalen:

```
ssb-project create stat-testprod
```

Hvis du stod i hjemmemappen din på når du skrev inn kommandoen over i terminalen, så har du fått mappestrukturen som vises i Figur 18.1.<sup>2</sup> Den inneholder følgende :

- **.git**-mappe som blir opprettet for å versjonshåndtere med Git.
- **src**-mappe som skal inneholde all koden som utgjør produksjonsløpet.
- **tests**-mappe som inneholder tester du skriver for koden din.
- **LICENCE**-fil som skal benyttes for public-repos i SSB.
- **poetry.lock**-fil som inneholder alle versjoner av Python-pakker som blir brukt.
- **README.md**-fil som brukes for tekstlig innhold på GitHub-siden for prosjektet.



Figur 18.1: Mappen som ble opprettet av ssb-project.

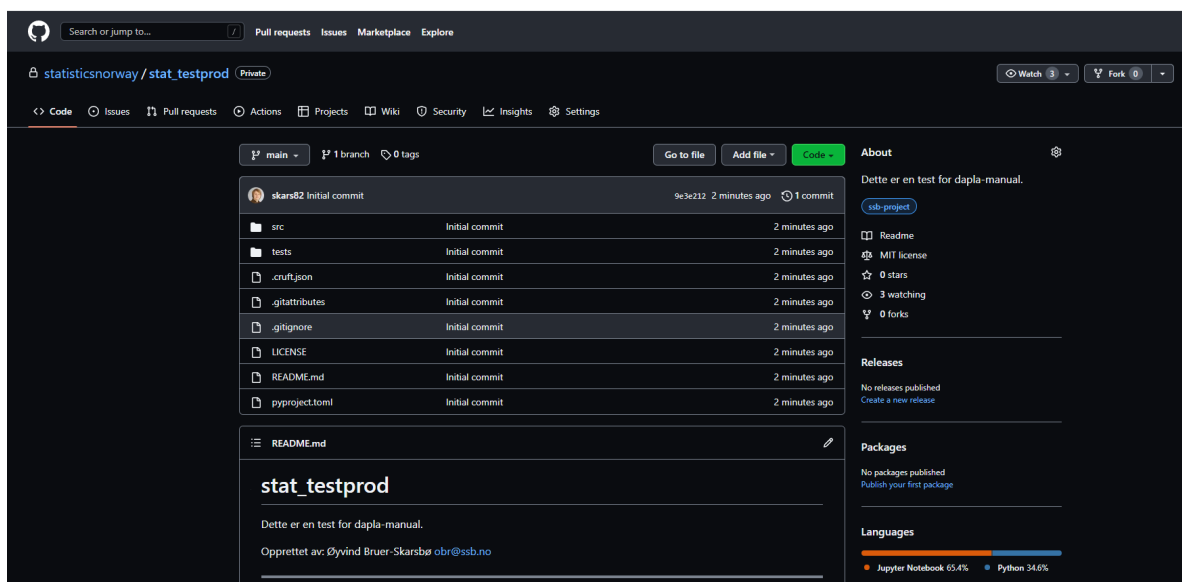
<sup>2</sup>Filer og mapper som starter med punktum er skjulte med mindre man ber om å se dem. I Jupyterlab kan disse vises i filutforskeren ved å velge **View** fra menylinjen, og deretter velge **Show hidden files**. I en terminal skriver man `ls -a` for å se de.

## 18.2.2 Med Github-repo

Over så opprettet vi et **ssb-project** uten å opprette et GitHub-repo. Hvis du ønsker å opprette et GitHub-repo også må du endre kommandoen over til:

```
ssb-project create stat-testprod --github --github-token='blablabla'
```

Kommandoen over oppretter en mappestruktur slik vi så tidligere, men også et ssb-project som heter **stat-testprod** med et GitHub-repo med samme navn. Som du ser så må vi da sende med opsjonen `--github` og PAT med opsjonen `--github-token='blablabla'`. Repoet i GitHub ser da ut som i Figur 18.2.



Figur 18.2: GitHub-repo som er opprettet av ssb-project

⚠ Når du har opprettet et nytt ssb-project, eller bygget et eksisterende prosjekt, så kan det ta rundt 30 sekunder før kernelen viser seg i Jupyterlab-launcher. Vær tålmodig!

## 18.3 Installere pakker

Når du har opprettet et **ssb-project** så kan du installere de python-pakkene du trenger fra [PyPI](#). Hvis du for eksempel ønsker å installere [Pandas](#), et populært data wrangling bibliotek, så kan du gjøre følgende:

1. Åpne en terminal i Jupyterlab.



2. Gå inn i prosjektmappen din ved å skrive

```
cd <sti til prosjektmappe>
```

3. Lag en *branch*/utviklingsgren som f.eks. heter **install-pandas**:

```
git checkout -b install-pandas
```

4. Installer **Pandas** ved å skrive følgende

```
poetry add pandas
```



```
Terminal 2  x  _quarto.yml  x  nytt-ssbproject.q  x  Terminal 1  x  git-github.qmd  x  +

jovyan@jupyter-obr-40ssb-2eno ~ $ cd stat_testprod/
jovyan@jupyter-obr-40ssb-2eno ~/stat_testprod (main) $ poetry add pandas
Using version ^1.5.1 for pandas

Updating dependencies
Resolving dependencies... (1.5s)

Writing lock file

No dependencies to install or update
jovyan@jupyter-obr-40ssb-2eno ~/stat_testprod (main) $
```

Figur 18.3: Installasjon av Pandas med ssb-project

Figur 18.3 viser hvordan dette vil se ut i en Jupyterlab-terminal. Kommandoen for å installere noe er **poetry add** etterfulgt av pakkenavnet. Vi ser også at den automatisk legger til Pandas-versjonen i filen **poetry.lock**. Les mer om [hvordan man installerer pakker](#) her.

## 18.4 Push til GitHub

Når du nå har installert en pakke så har filen **poetry.lock** endret seg. La oss for eksempelets skyld anta at du ønsker å bruke **Git** til å dokumentere denne hendelsen, og dele det med en kollega via GitHub. Hvis vi har opprettet et **ssb-project** med et [GitHub-repo](#) så kan vi gjøre akkurat dette:

1. Vi kan *stage* alle endringer med følgende kommando i terminalen når vi står i prosjektmappen:

```
git add -A
```

2. Videre kan *commit* en endring, dvs. ta et stillbilde av koden i dette øyeblikket, ved å skrive følgende:

```
git commit -m "Installert pandas"
```

3. Push det opp til GitHub<sup>3</sup>. Anta at vi gjorde dette i *branchen* *install-pandas* som ble opprettet tidligere. Da kan vi skrive følgende:

```
git push --set-upstream origin install-pandas
```

Mer kommer her.

## 18.5 Bygg eksisterende ssb-project

Når vi skal samarbeide med andre om kode så gjør vi dette via **GitHub**. Når du *pusher* koden din til GitHub, så kan samarbeidspartnere *pulle* ned koden og jobbe videre med den. Men når de henter ned koden så vil de bare hente ned selve koden, ikke pakker og Python-versjonen som ble brukt. De må installere alt som du hadde installert. I tillegg trenger de en kernel hvis de skal jobbe i Jupyterlab. **ssb-project** gjør det svært enkelt å bygge opp det du trenger, siden det virtuelle miljøet har all informasjon om hva som trengs.

For at samarbeidspartneren din skal kunne bygge miljøet på nytt, må de ha gjort en minimal konfigurering av Git. [Les mer om hvordan](#) du frem for å gjøre dette her.

For å bygge opp et eksisterende miljø gjør du følgende:

1. Først må du kopiere prosjektet ned lokalt, eller *clone* repoet med git-terminologi

---

<sup>3</sup>Å pushe til GitHub uten å sende ved *Personal Access Token* fordrer at du har lagret det lokalt så Git kan finne det. [Her et eksempel på hvordan det kan gjøres.](#)

```
git clone https://github.com/statisticsnorway/<prosjektnavn>
```

2. Gå inn i mappen du klonet

```
cd <prosjektnavn>
```

3. Skape et virtuelt miljø og installere en tilsvarende Jupyter kernel med

```
ssb-project build
```

## 18.6 Rydd opp etter deg

Det vil være tilfeller hvor man ønsker å slette et ssb-project, enten fordi man ikke trenger koden lenger eller fordi man bare testet litt.

### 18.6.1 Lokalt

Hvis man jobber med flere prosjekter så kan det fort bli mange Jupyter kerneler hengende igjen. Derfor er det også mulighet å kjøre

```
ssb-project clean stat-testprod
```

som sletter Jupyter-kernelen og de installerte pakkene i prosjektet. Hvis du også ønsker å slette selve mappen med kode må du gjøre det manuelt<sup>4</sup>:

```
rm -rf ~/stat-testprod/
```

Prosjektmappen over lå direkte i hjemmemappen min og hjemmemappen på Linux kan alltid referes til med et tilda-tegn ~.

### 18.6.2 Arkiver GitHub-repo

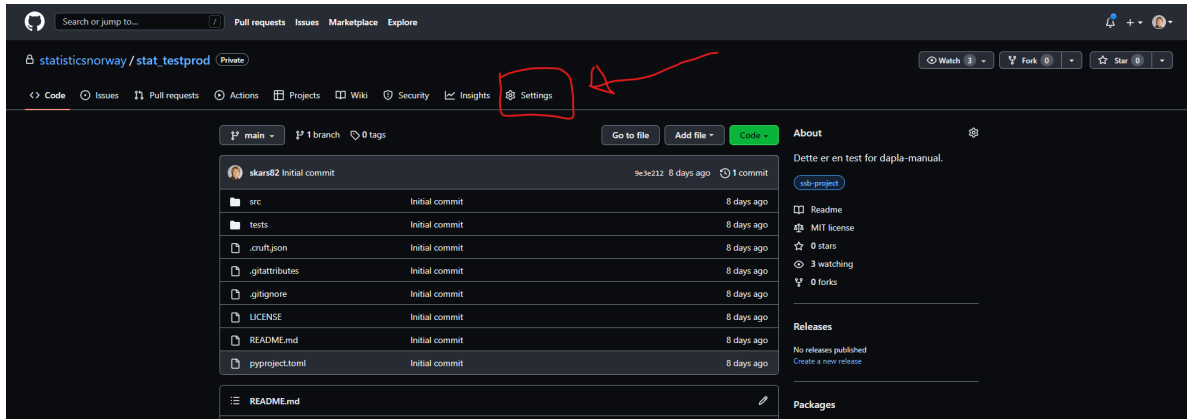
GitHub-repoer som er opprettet under SSB sin organisasjons statisticsnorway på GitHub kan ikke slettes, bare arkiveres. Grunnen er at hvis man oppdager en sårbarhet senere så er det viktig å kunne se repoet for å forstå hva som har skjedd.

Hvis du ikke trenger et GitHub-repo lenger kan man **arkivere repoet**. Det gjør du på følgende måte:

---

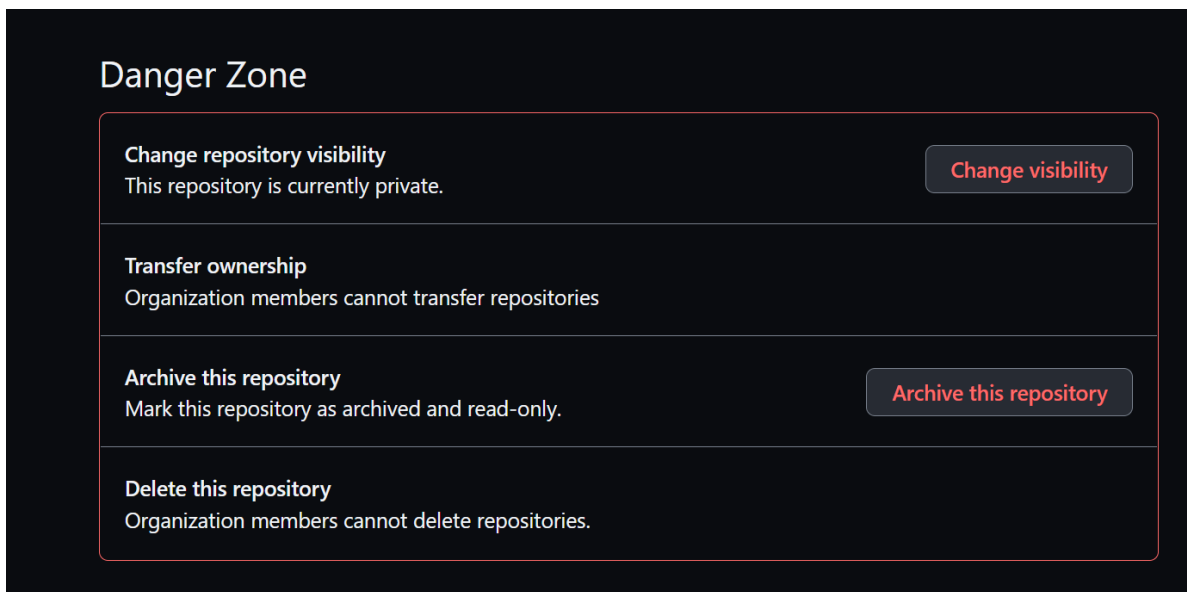
<sup>4</sup>Dette kan også gjøres ved å høyreklikke på mappen i Jupyterlab sin filutforsker og velge **Delete**.

1. Gi inn i repoet **Settings** slik som vist med rød pil i Figur 18.4.



Figur 18.4: Settings for repoet.

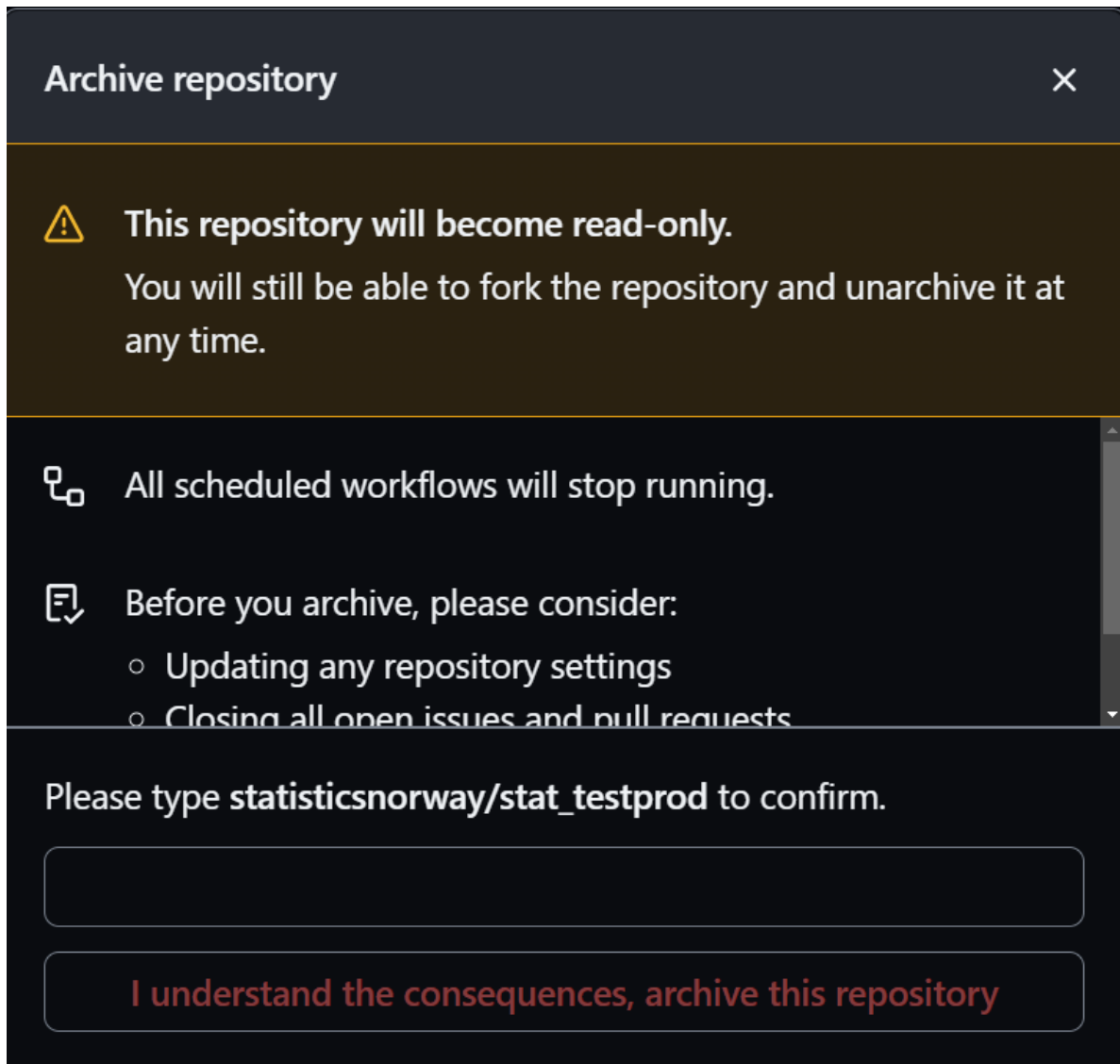
2. Under **General** scroller du deg ned til delen som heter **Danger Zone** og velger **Archive this repository**, slik som vist på Figur 18.5.



Figur 18.5: Arkivering av et repo.

3. I dialogboksen som dukker opp fyller du inn reponavnet som beskrevet og trykker på **I understand the consequences, archive this repository**, som vist i Figur 18.6.

Når det er gjort så er repoet lesbart, men man kan ikke jobbe med det. Men som vi ser



Figur 18.6: Bekreftelse av arkiveringen.

av @#fig-github-repo-settings-archive-warning kan man omgjøre arkiveringen senere hvis det skulle være ønskelig.

## 18.7 Hva med R?

Vi har foreløpig ikke integret **R** i **ssb-project**. Grunnen er at det mest populære *virtuelle miljø*-verktøyet for R, [renv](#), kun tilbyr å passe på versjoner av R-pakker og ikke selve R-installasjonen. Det er en svakhet som trolig gjør det vanskeligere enn nødvendig å gjenskape tidligere publiserte resultater med **ssb-project**. I tillegg klarer den ikke å gjenkjenne pakker som blir brukt i ipynb-filer.

Planen er å finne et annet verktøy enn **renv** som kan også reprodusere R-versjonen. Team Statistiktjenester ser nærmere på hvilke alternativer som finnes og vil tilby noe i fremtiden.

I mellomtiden kan man bruke renv slik det er [beskrevet her for skymiljøet](#), og med [denne modifiseringen](#) for bakkemiljøet.

# 19 Git og Github

I SSB anbefales det man versjonhåndterer koden sin med [Git](#) og deler koden via [GitHub](#). For å lære seg å bruke disse verktøyene på en god måte er det derfor viktig å forstå forskjellen mellom Git og Github. Helt overordnet er forskjellen følgende:

- **Git** er programvare som er installert på maskinen du jobber på og som sporer endringer i koden din.
- **GitHub** er et slags felles mappesystem på internett som lar deg dele og samarbeide med andre om kode.

Av definisjonene over så skjønner vi at det er **Git** som gir oss all funksjonalitet for å lagre versjoner av koden vår. GitHub er mer som et valg av mappesystem. Men måten kodemiljøene våre er satt opp på **Dapla** så har vi ingen fellesmappe som alle kan kjøre koden fra. Man utvikler kode i sin egen hjemmemappe, som bare du har tilgang til, og når du skal samarbeide med andre, så må du sende koden til GitHub. De du samarbeider med må deretter hente ned denne koden før de kan kjøre den.

I dette kapitlet ser vi nærmere på Git og Github og hvordan de er implementert i SSB. Selv om SSB har laget programmet **ssb-project** for å gjøre det lettere å bl.a. forholde seg til Git og GitHub, så vil vi i dette kapitlet forklare nærmere hvordan det funker uten dette hjelpemiddelet. Forhåpentligvis vil det gjøre det lettere å håndtere mer kompliserte situasjoner som oppstår i arbeidshverdagen som statistikker.

## 19.1 Git

**Git** er terminalprogram som installert på maskinen du jobber. Hvis man ikke liker å bruke terminalen finnes det mange pek-og-klikk versjoner av **Git**, blant annet i **Jupyterlab**, **SAS EG** og **RStudio**. Men typisk vil det en eller annen gang oppstå situasjoner der det ikke finnes løsninger i pek-og-klikk versjonen, og man må ordne opp i terminalen. Av den grunn velger vi her å fokusere på hvordan **Git** fungerer fra terminalen. Vi vil også fokusere på hvordan **Git** fungerer fra **terminalen** i **Jupyterlab** på **Dapla**.

### 19.1.1 Hva er Git?

Kommer snart. Kort forklaring med lenke til mer utfyllende svar.

## 19.1.2 Oppsett av Git

Mer kommer.

### 19.1.2.1 Minimal Git-konfigurasjon

For å brukt Git er det strengt tatt to ting som må konfigureres:

1. Brukernavn
2. E-post

Denne informasjonen brukes av Git hver gang du sjekker inn en endring i koden slik at man kan vite hvem som gjorde endringen senere. Dette må settes én gang per miljø hvor du skal jobbe med Git. Hvis du f.eks. jobber i **Jupyterlab** på **Dapla** så kan du åpne en terminal og skrive følgende for å lagre ditt brukernavn:

```
git config --global user.name "Ola Nordmann"
```

For å sette e-post gjør du veldig lignende:

```
git config --global user.email olanordamnn@ssb.no
```

Når du har kjørt disse to kommandoene så kan du bruke Git. Informasjonen du la til over brukes ikke til noe annet enn å fortelle de du samarbeider med om at du har gjort endringer på koden. Den er verken knyttet opp mot din SSB-bruker eller din GitHub-bruker.

## 19.1.3 Git og Notebooks

Kommer snart. Jupyter og nbstripout. json.

## 19.1.4 Vanlige Git-operasjoner

Kommer snart. clone, add, commit, push, pull, merge, revert, etc.

## 19.2 GitHub

GitHub er et nettsted som fungerer som vårt felles mappesystem for deling av kode. SSB har sin egen organisasjonskonto med navn [statisticsnorway](#)



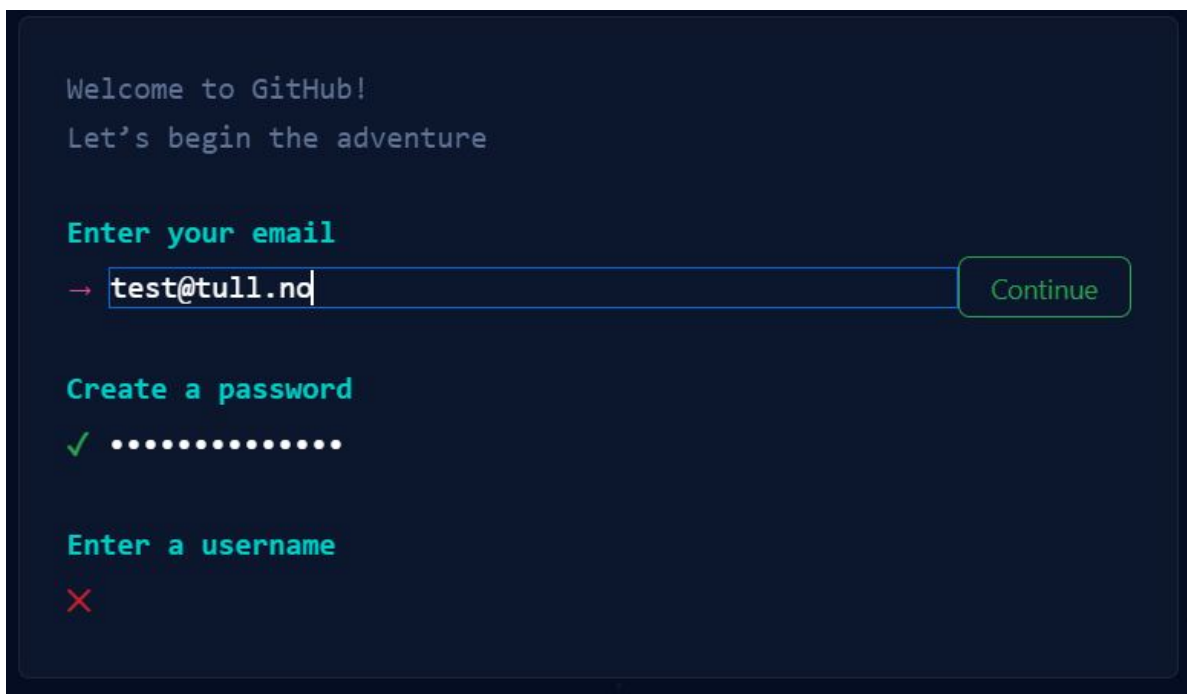
### 19.2.1 Opprett GitHub-bruker

Dette kapitlet er bare relevant hvis man ikke har en GitHub-brukerkonto fra før. For å bruke **ssb-project**-programmet til å generere et **remote repo** på GitHub må du ha en konto. Derfor starter vi med å gjøre dette. Det er en engangsjobb og du trenger aldri gjøre det igjen.

**i** SSB har valgt å ikke sette opp SSB-brukerne til de ansatte som GitHub-brukere. En viktig årsak er at en GitHub-konto ofte regnes som en del av den ansattes CV. For de som aldri har brukt GitHub før kan det virke fremmed, men det er nok en fordel på sikt når alle blir godt kjent med denne arbeidsformen.

Slik gjør du det:

1. Gå til <https://github.com/>
2. Trykk **Sign up** øverst i høyre hjørne
3. I dialogboksen som åpnes, se Figur 19.1, skriver du inn din e-postkonto og lager et passord. Dette trenger ikke være din SSB-bruker og e-post. Vi anbefaler at du bruker en personlig e-postkonto og velger ditt eget passord. Det samme gjelder **brukernavn** også.

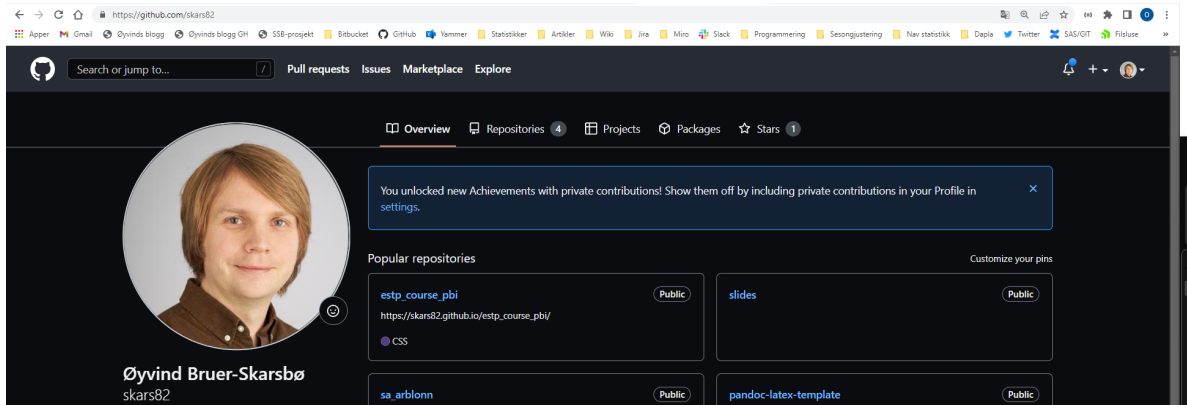
The image shows a dark-themed GitHub sign-up dialog box. At the top, it says "Welcome to GitHub!" and "Let's begin the adventure". Below this, there are three sections: "Enter your email" with a text input field containing "test@tull.no" and a "Continue" button; "Create a password" with a green checkmark and a series of dots representing a password; and "Enter a username" with a red 'X' icon, indicating an error or required field. The background is dark blue/black.

Figur 19.1: Dialogboks for opprettelse av GitHub-bruker.

Du har nå laget en egen GitHub-bruker. I neste steg skal vi knytte denne kontoen til din SSB-bruker.

### 19.2.2 To-faktor autentisering

Hvis du har fullført forrige steg så har du nå en GitHub-konto. Hvis du står på din profil-side så ser den ut som i Figur 19.2.



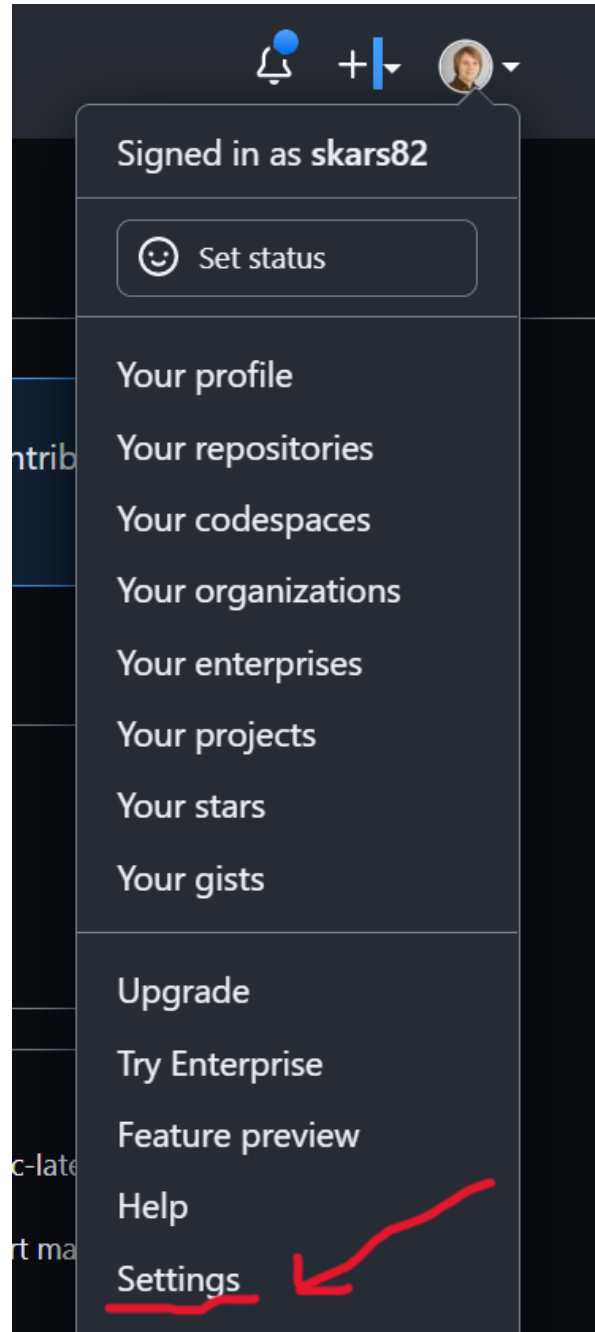
Figur 19.2: Et eksempel på hjemmeområdet til en GitHub-bruker

Det neste vi må gjøre er å aktivere 2-faktor autentisering, siden det er dette som benyttes i SSB. Hvis du står på siden i bildet over, så gjør du følgende for å aktivere 2-faktor autentisering mot GitHub:

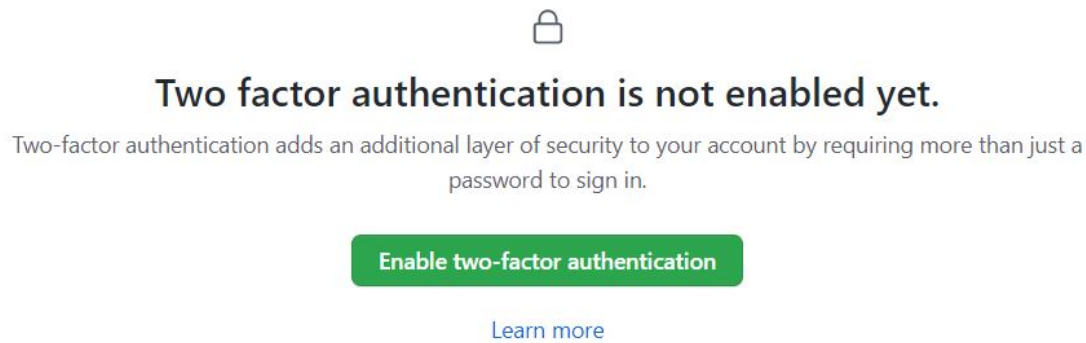
1. Trykk på den lille pilen øverst til høyre og velg **Settings** (se Figur 19.3).
2. Deretter velger du **Password and authentication** i menyen til venstre.
3. Under **Two-factor authentication** trykker du på **Enable**.
4. Figur 19.4 viser dialogboksen som vises. Velg **Enable two-factor authentication**.
5. Figur 19.5 viser dialogboksen som vises for å velge hvordan man skal autentisere seg. Her anbefales det å velge **Set up using an app**, slik at du kan bruke *Microsoft Authenticator*-appen på din mobil.

Figur 19.6 viser QR-koden som vises. Denne skal vi bruke i neste steg.

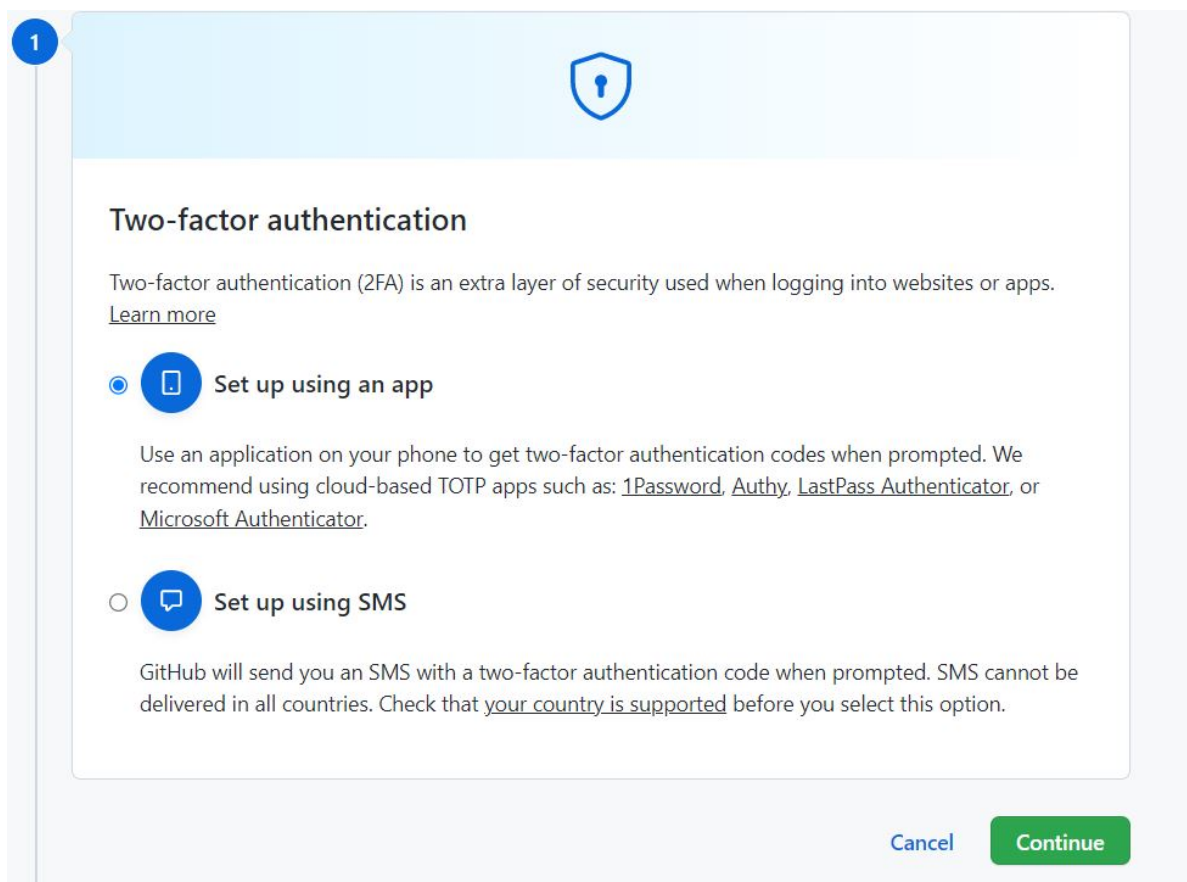
6. Strekkoden over skal skannes i din **Microsoft Authenticator**-app på mobilen, som vist i Figur 19.7. Åpne appen, trykk på **Bekreftede ID-er**, og til slutt trykk på **Skann QR-kode**. Deretter skanner du QR-koden fra punkt 5.



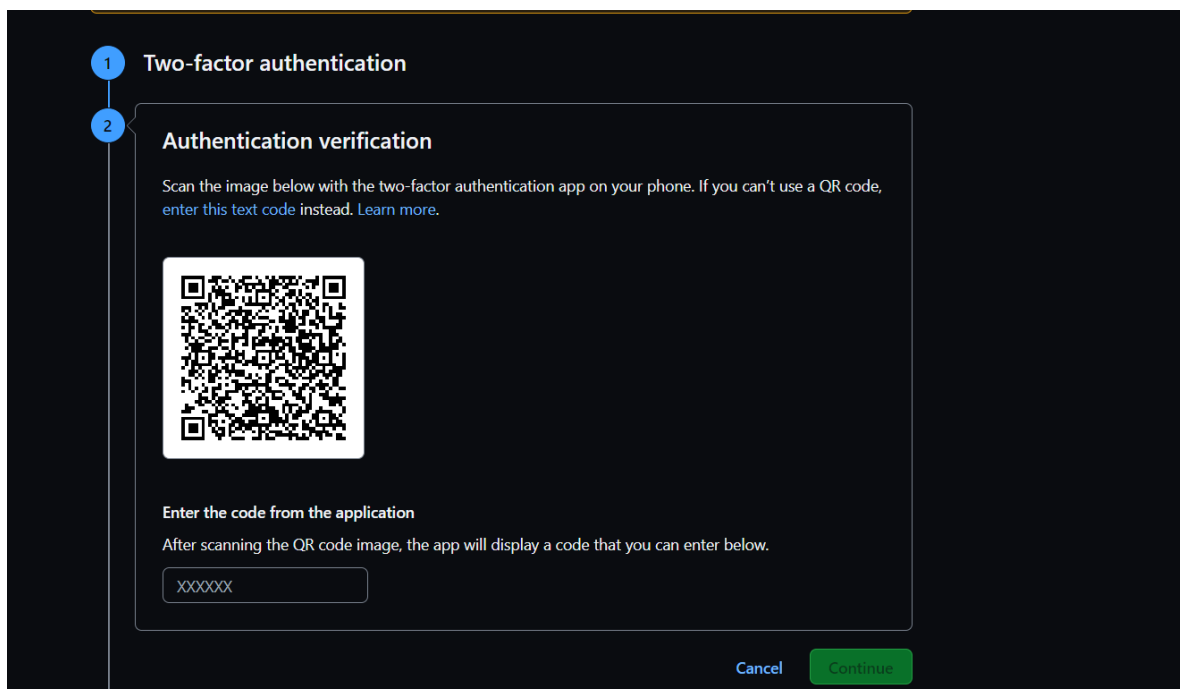
Figur 19.3: Åpne settings for din GitHub-bruker.



Figur 19.4: Dialogboks som åpnes når 2FA skrur på første gang.



Figur 19.5: Dialogboks for å velge hvordan man skal autentisere seg med 2FA.



Figur 19.6: QR-kode som skannes av Microsoft Authenticator.

7. Når koden er skannet har du fått opp følgende bilde på appens hovedside (se bilde til høyre). Skriv inn den 6-siffer koden på GitHub-siden med QR-koden.
8. Til slutt lagrer du **Recovery-codes** et trygt sted som bare du har tilgang til.

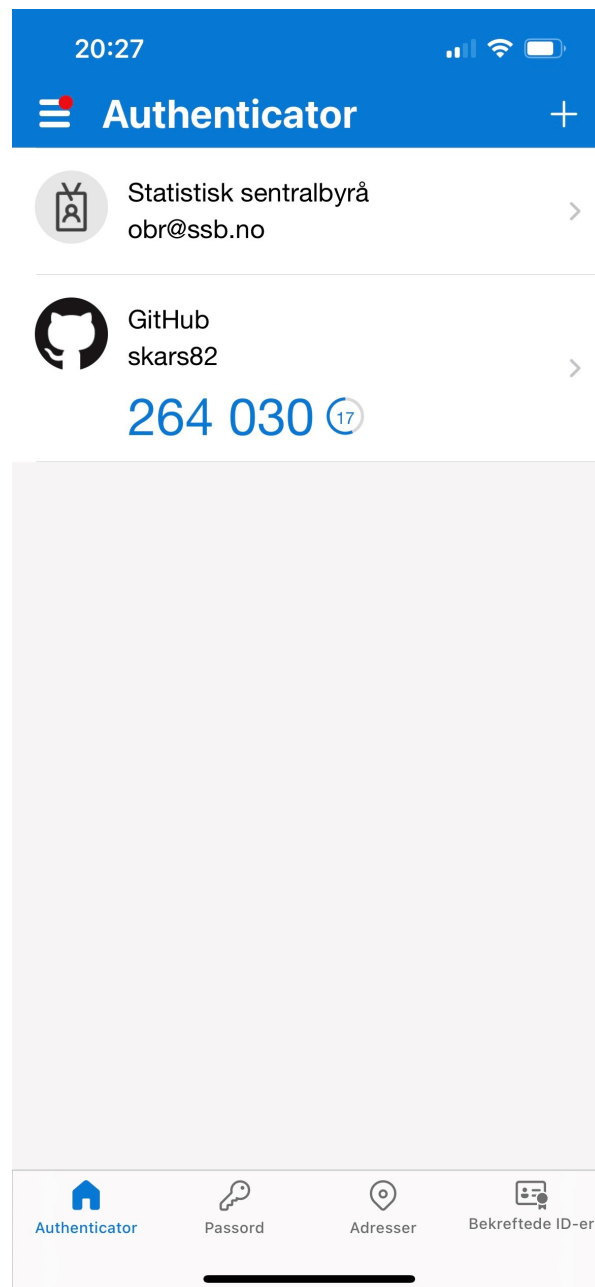
Nå har vi aktivert 2-faktor autentisering for GitHub og er klare til å knytte vår personlige konto til vår SSB-bruker på SSBs “Github organisation” [statisticsnorway](https://github.com/orgs/statisticsnorway).

### 19.2.3 Koble deg til SSB

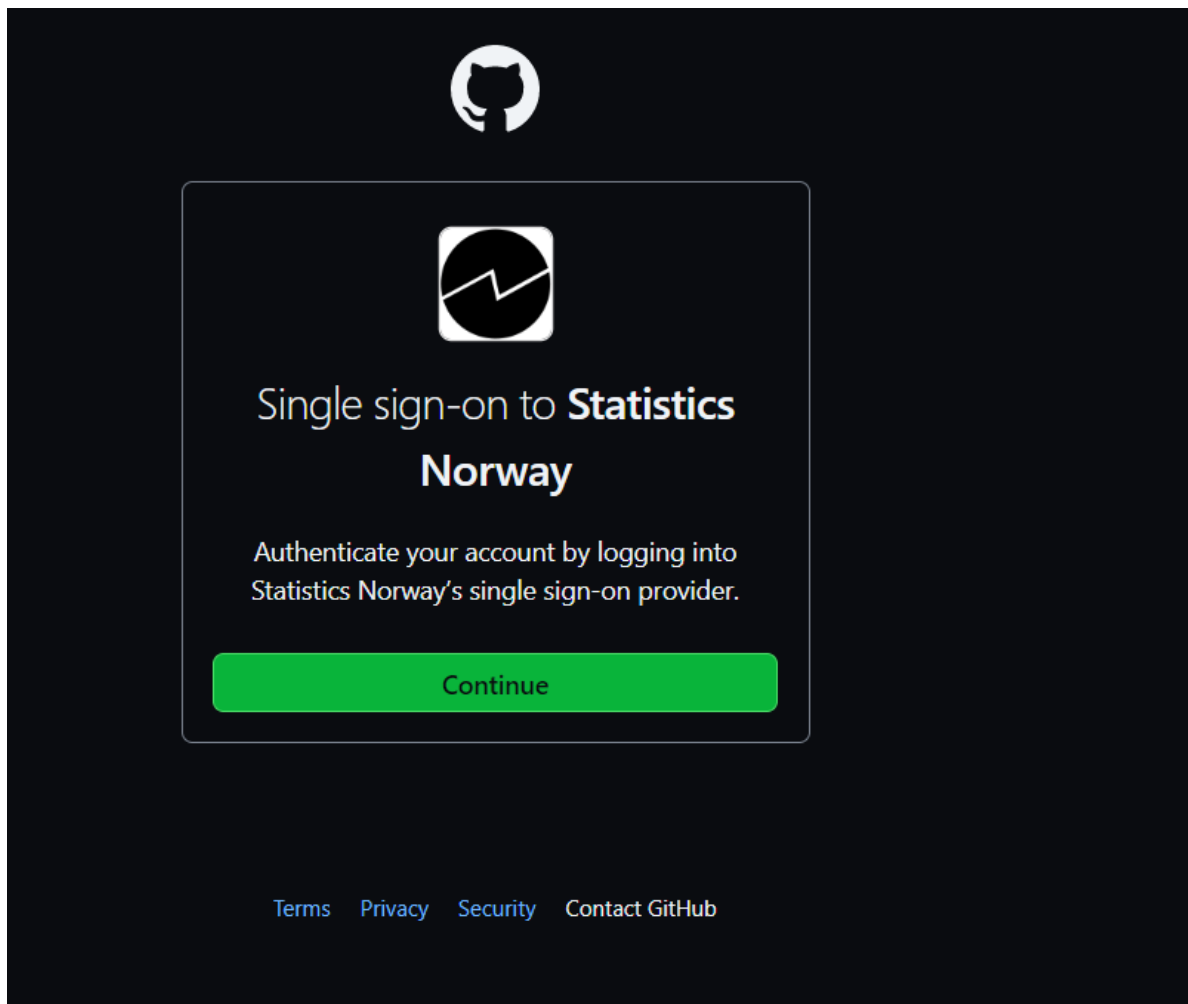
I forrige steg aktiverte vi 2-faktor autentisering for GitHub. Det neste vi må gjøre er å koble oss til **Single Sign On (SSO)** for SSB sin organisasjon på GitHub:

1. Trykk på lenken <https://github.com/orgs/statisticsnorway/sso>
2. I dialogboksen som dukker opp trykker du på **Continue**, slik som vist i Figur 19.8.

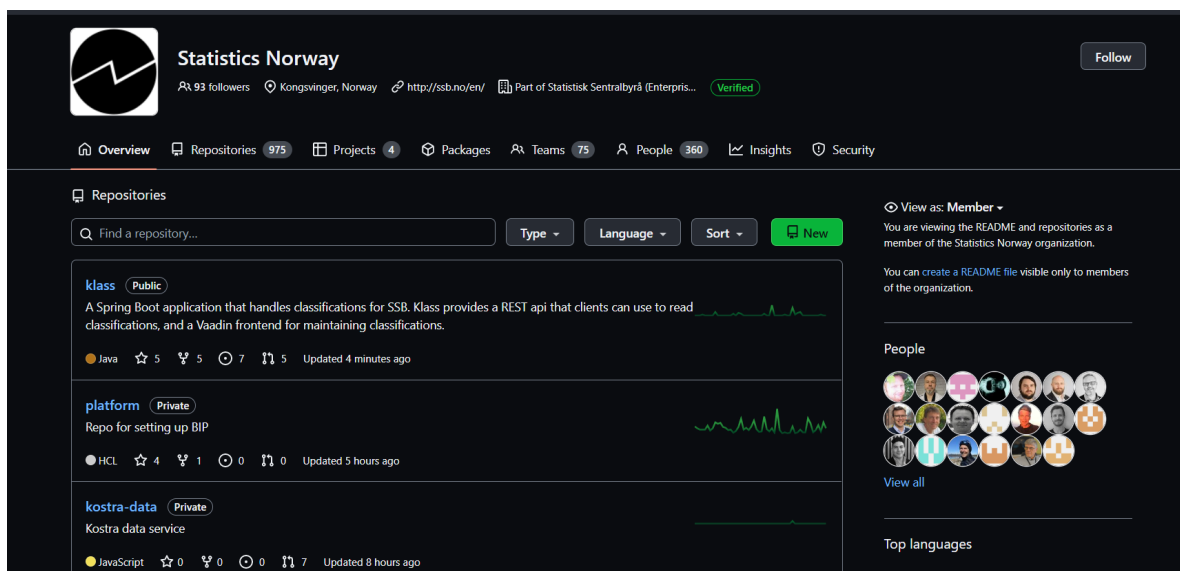
Når du har gjennomført dette så har du tilgang til **statisticsnorway** på GitHub. Går du inn på [denne lenken](#) så skal du nå kunne lese både **Public**, **Private** og **Internal** repoer, slik som vist i Figur 19.9.



Figur 19.7: Mobilappen *Microsoft authenticator*



Figur 19.8: Single Sign on (SSO) for SSB sin organisasjon på GitHub



Figur 19.9: Medlemsvisning for SSB sin GitHub-organisasjon.

## 19.2.4 Personal Access Token (PAT)

Når vi skal jobbe med SSB-kode som ligger lagret hos **statistcsnorway** på GitHub, så må vi autentisere oss. Måten vi gjøre det på er ved å generere et **Personal Access Token** (ofte forkortet **PAT**) som vi oppgir når vi vil hente eller oppdatere kode på GitHub. Da sender vi med PAT for å autentisere oss for GitHub.

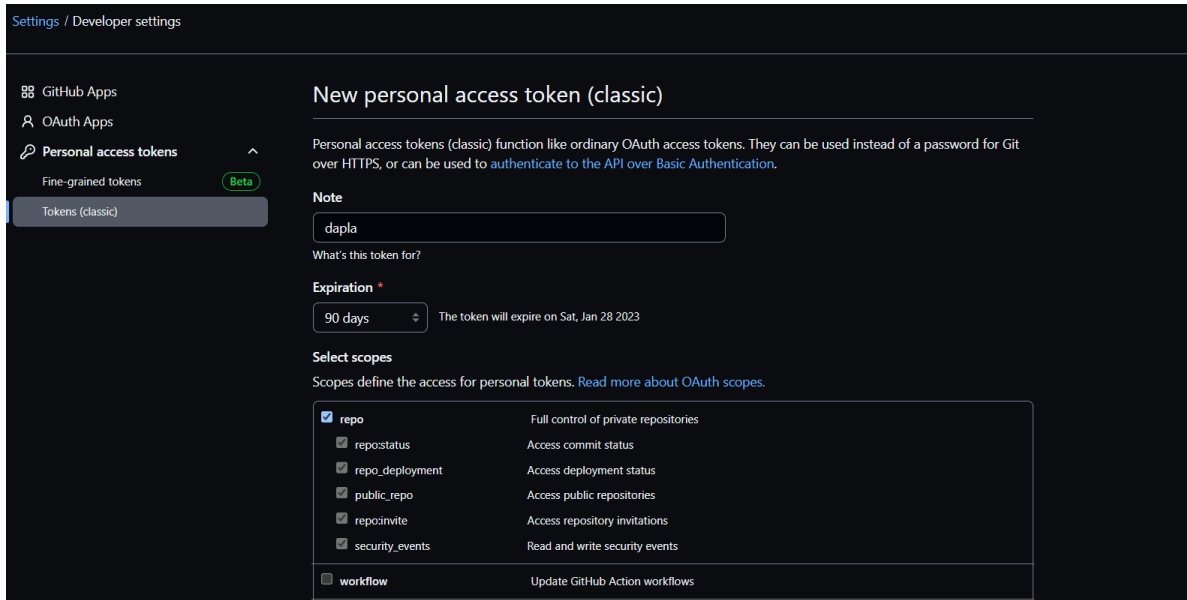
### 19.2.4.1 Opprette PAT

For å lage en PAT som er godkjent mot *statisticsnorway* så gjør man følgende:

1. Gå til din profilside på GitHub og åpne **Settings** slik som ble vist Seksjon 19.2.2.
2. Velg **Developer Settings** i menyen til venstre.
3. I menyen til venstre velger du **Personal Access Token**, og deretter **Tokens (classic)**.
4. Under **Note** kan du gi PAT'en et navn. Velg et navn som er intuitivt for deg. Hvis du skal bruke PAT til å jobbe mot Dapla, så ville jeg ganske enkelt kalt den *dapla*. Hvis du skal bruke den mot bakkemiljøet ville jeg kalt den *prodsone* eller noe annet som gjør det lett for det skjønne innholdet i ettertid.
5. Under **Expiration** velger du hvor lang tid som skal gå før PAT blir ugyldig. Dette er en avveining mellom sikkerhet og hva som er praktisk. Det anbefales at du velger **365 dager**. Når PAT utløper må du gjenta stegene i dette kapitlet.

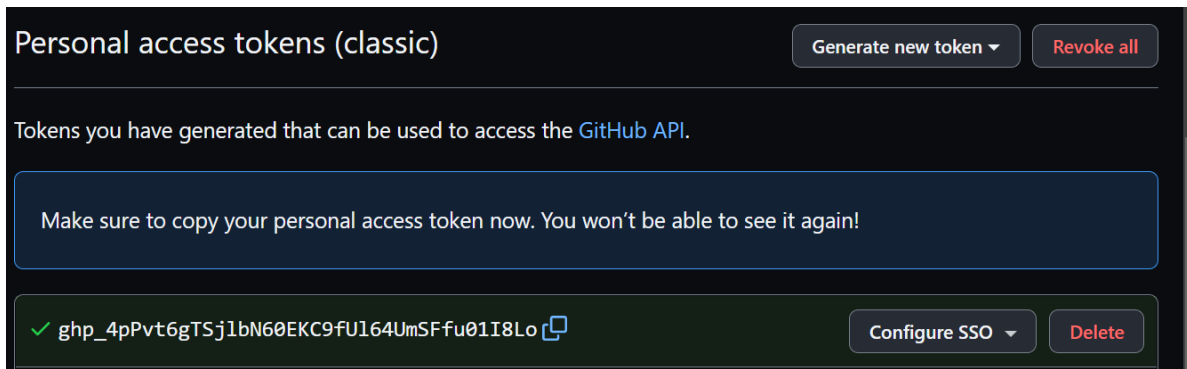


6. Under **Select scopes** velger du **Repo** slik som vist i Figur 19.10.



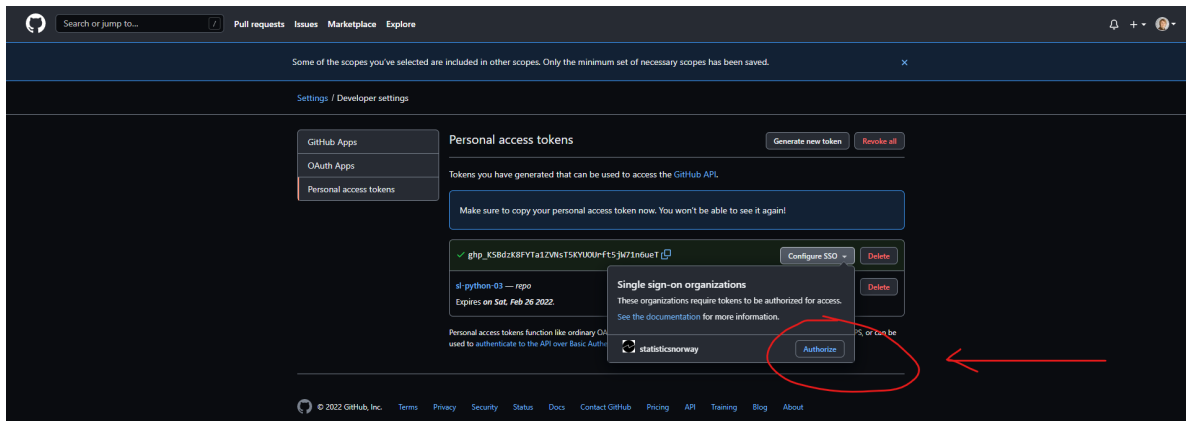
Figur 19.10: Gi token et kort og beskrivende navn

8. Trykk på **Generate token** nederst på siden og du får noe lignende det du ser i Figur 19.11.



Figur 19.11: Token som ble generert.

9. Kopier deretter PAT til en midlertidig fil. Grunnen er at du aldri vil se det igjen her etter at vi har gjennomført neste steg.
10. Deretter trykker du på **Configure SSO** og velger **Authorize** ved siden statisticsnorway, slik som vist i Figur 19.12. Svar deretter på spørsmålene som dukker opp.



Figur 19.12: Autorisering av Token mot SSBs GitHub-organisasjon.

Vi har nå opprettet en PAT som er godkjent for bruk mot SSB sin kode på GitHub. Det betyr at hvis vi vil jobbe med **Git** på SSB sine maskiner i sky eller på bakken, så må vi sende med dette tokenet for å få lov til å jobbe med koden som ligger på **statisticsnorway** på GitHub.

#### 19.2.4.2 Lagre PAT

Det er ganske upraktisk å måtte sende med tokenet hver gang vi skal jobbe med GitHub. Vi bør derfor lagre det lokalt der vi jobber, slik at Git automatisk finner det. Det finnes mange måter å gjøre dette på og det er ikke bestemt hva som skal være beste-praksis i SSB. Men en måte å gjøre det er via en **.netrc**-fil. Vi oppretter da en fil som heter **.netrc** på vårt hjemmeområde, og legger følgende informasjon på en (hvilken som helst) linje i filen:

```
machine github.com login <github-bruker> password <Personal Access Token>
```

**GitHub-bruker** er da din personlige bruker og IKKE brukernavnet ditt i SSB. **Personal Access Token** er det vi lagde i forrige kapittelet.

En veldig enkel måte å lagre dette er som følger. Anta at min personlige GitHub-bruker er **SSB-Chad** og at min Personal Access Token er **blablabla**. Da kan jeg gjøre følgende for å lagre det i **.netrc**:

1. Gå inn i Jupyterlab og åpne en Python-notebook.
2. I den første kodecellen skriver du:

```
!echo "machine github.com login SSB-Chad password blablabla" >> ~/.netrc
```

Alternativt kan du droppe det utropstegnet og kjøre det direkte i en terminal. Det vil gi samme resultat. Koden over legger til en linje med teksten

**machine github.com login SSB-Chad password blablabla**

i en `.netrc`-fil på din hjemmeområdet, uavhengig av om du har en fra før eller ikke. Hvis du har en fil fra før som allerede har et token fra GitHub, ville jeg nok slettet det før jeg legger en et nytt token.

Hver gang du jobber mot GitHub vil Git sjekke om informasjon om autentisering ligger i denne filen, og bruke den hvis den ligger der.

### 19.2.4.3 Oppdater PAT

I eksempelet over lagde vi en PAT som var gyldig i 90 dager. Dermed vil du ikke kunne jobbe mot GitHub med dette tokenet etter 90 dager. For å oppdatere tokenet gjør du følgende:

1. Lag et nytt PAT ved å repetere Seksjon [19.2.4.1](#).
2. I miljøet der du skal jobbe med Git og GitHub går du inn i din `.netrc` og bytter ut token med det nye.

Og med det er du klar til å jobbe mot *statisticsnorway* på **GitHub**.

## 20 Virtuelle miljøer

### 20.1 Python

Et python virtuelt miljø inneholder en spesifikk versjon av python og et sett med pakker. Pakkene er kun tilgjengelige når det virtuelle miljøet er aktivert. Dette gjør at man unngår avhengighetskonflikter på tvers av prosjekter.

Se her for [mer informasjon om virtuelle miljøer](#).

#### 20.1.1 Anbefaling

Det er anbefalt å benytte verktøyet poetry for å administrere prosjekter og deres virtuelle miljø.

Poetry setter opp virtuelt miljø, gjør det enkelt å oppdatere avhengigheter, sette versjonsbegrensninger og reprodusere prosjektet.

Poetry gjør dette ved å lagre avhengigheters eksakte versjon i prosjektets “poetry.lock”. Og eventuelle begrensninger i “pyproject.toml”. Dette gjør det enkelt for andre å bygge prosjektet med akkurat de samme pakkene og begrensningene.

### 20.2 R

## 21 Jupyter-kernels

## 22 Installere pakker

### 22.1 Python

Installering av pakker er kun mulig i et [virtuelt miljø](#). Det er [anbefalt å benytte poetry](#) til dette. Eksempelene videre tar derfor utgangspunkt i et poetry prosjekt.

Det er mulig å [installere pakker med pip](#). Pakker kan installeres som normalt, hvis man har satt opp og aktivert et [virtuelt miljø](#).

#### 22.1.1 Poetry prosjekt eksempel

Dette eksemplet viser hvordan man setter opp et enkelt poetry prosjekt kalt test, hvis man ønsker å benytte et annet prosjektnavn må man endre dette i hver av kommandoene.

Sett opp prosjektet:

```
poetry new test
```

Naviger inn i prosjektmappen:

```
cd test
```

Bruk poetry install for å bygge prosjektet:

```
poetry install
```

Hvis man får en tilbakemelding som denne er prosjektet satt opp korrekt:

```
Creating virtualenv test-EojoH6Zm-py3.10 in /home/jovyan/.cache/pypoetry/virtualenvs
Updating dependencies
Resolving dependencies... (0.1s)

Writing lock file
```

### 22.1.2 Installering

For å legge til pakker i et prosjekt benyttes kommandoen `poetry add`.

Skal man legge til pakken “pendulum” vil det se slik ut:

```
poetry add pendulum
```

Poetry tilbyr måter å sette versjonsbegrensninger for pakker som legges til i et prosjekt, dette kan man [lese mer om her](#).

### 22.1.3 Avinstallering

For å fjerne pakker fra et prosjekt benytter man `poetry remove`.

Hvis man ønsker å fjerne “pendulum” fra et prosjekt vil kommandoen se slik ut:

```
poetry remove pendulum
```

### 22.1.4 Oppgradere pakker

For å oppdatere pakker i et prosjekt benytter man kommandoen `poetry update`.

Skal man oppdatere pakken “pendulum” bruker man:

```
poetry update pendulum
```

Skal man oppdatere alle pakken i et prosjekt benytter man:

```
poetry update
```

### 22.1.5 Legge til kernel for poetry

For å kunne benytte det virtuelle miljøet i en notebook må man sette opp en kernel. Kernel burde gis samme navn som prosjektet.

Først legger man til ipykernel:

```
poetry add ipykernel
```

Så opprettes kernel med:

```
poetry run python -m ipykernel install --user --name test
```

Etter dette er kernelen test opprettet og kan velges for å benytte miljøet i en notebook.

### 22.1.6 Fjerne kernel

For å fjerne en kernel med navn test bruker man:

```
jupyter kernelspec remove test
```

Du vil bli spurt om å bekrefte, trykk y hvis man ønsker å slette:

```
Kernel specs to remove:
  test                               /home/jovyan/.local/share/jupyter/kernels/test
Remove 1 kernel specs [y/N]: y
```

Etter dette er kernelen fjernet.

### 22.1.7 Sikkerhet

Hvem som helst kan legge til pakker på PyPi, det betyr at de i verstefall, kan inneholde skadelig kode. Her er en list med viktige tiltak som minimere risikoen:

- a) Før man installerer pakker bør man alltid søke de opp på <https://pypi.org>. Det er anbefalt å klippe og lime inn pakkenavnet når man skal legge det til i et prosjekt.
- b) Er det et populært/velkjent prosjekt? Hvor mange stjerner og forks har repoet?

## 22.2 R

Installering av pakker for R-miljøet i Jupyterlab er foreløpig ikke en del av [ssb-project](#). Men vi kan bruke [renv](#). Mer kommer.

### 22.2.1 Installering

For å installere dine egne R-pakker må du opprette et virtuelt miljø med **renv**. Gå inn i **Jupyterlab** og åpne R-notebook. Deretter skriver du inn følgende i kodecelle:

```
renv::init()
```



Denne kommandoer aktiverer et virtuelt miljø i mappen du står i. Rent praktisk vil det si at du fikk følgende filer/mapper i mappen din:

### **renv.lock**

En fil som inneholder versjoner av alle pakker du benytter i koden din.

### **renv**

Mappe som inneholder alle pakkene du installerer.

Nå som vi har et virtuelle miljøet på plass kan vi installere en R-pakke. Du kan gjøre dette fra både terminalen og fra en Notebook. Vi anbefaler på gjøre det fra terminalen fordi du da får tilbakemelding på om installeringen gikk bra heller ikke. For å installere i terminalen gjør du følgende:

1. Åpne en terminal i Jupyterlab
2. Stå i mappen der du aktiverte det virtuelle miljøet
3. Skriv in R og trykk enter.

Det vi nå har gjort er å åpne **R** fra terminalen slik at vi kan skrive R-kode direkte i terminalen. Det omtales ofte som en *R Console*. Nå kan du skrive inn en vanlig kommando for å installere R-pakker:

```
renv::install("PxWebApiData")
```

Over installerte vi pakken **PxWebApiData** som er en pakke skrevet i SSB for å hente ut data fra vår statistikkbank. La oss bruke pakken i koden vår med ved å skrive følgende i kodecelle i Notebooken vår:

```
library(PxWebApiData)
ApiData("https://data.ssb.no/api/v0/en/table/04861",
        Region = c("1103", "0301"), ContentsCode = "Bosatte", Tid = c(1, 2, -2, -1))
```

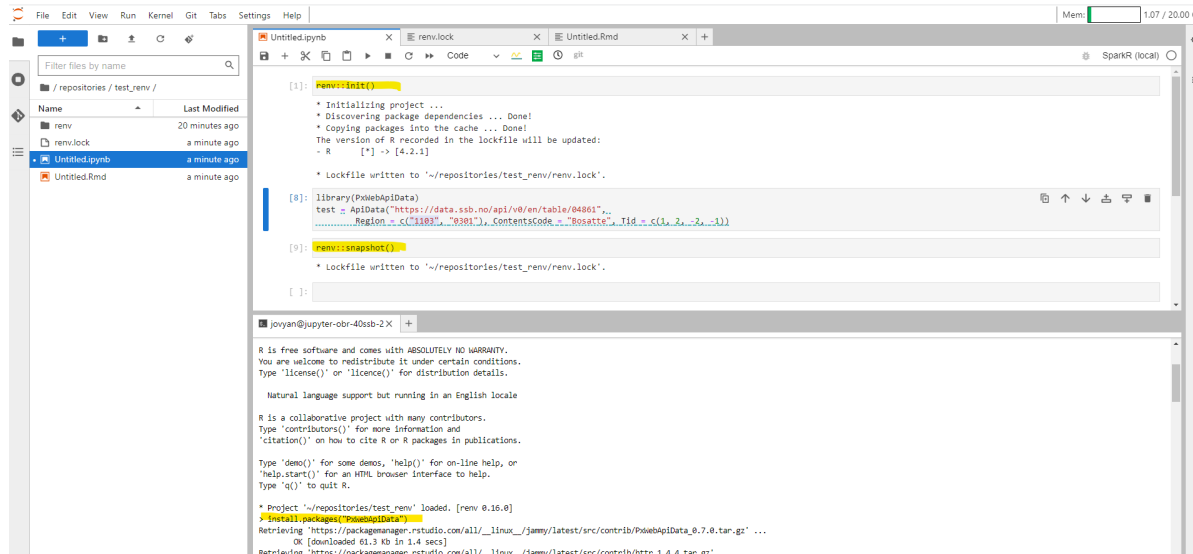
Når vi nå har brukt **PxWebApiData** i koden vår så kan vi kjøre en kommando som legger til den pakken i **renv.lock**. Men før vi kan gjøre det må vi være obs på at **renv** ikke klarer å gjenkjenne pakker som er i bruk Notebooks (ipynb-filer). Det er veldig upraktisk, men noe vi må forholde oss til når vi jobber med **renv** i Jupyterlab. En mulig løsning for dette er å bruke **Jupytertext** til å synkronisere en ipynb-fil med en Rmd-fil. **renv** kjenner igjen både R- og Rmd-filer. For å synkronisere filene gjør du følgende:

1. Trykk **Ctrl+Shift C**
2. Skriv inn **Pair** i søkefeltet som dukker opp
3. Velg **Pair Notebook with R Markdown**

Hvis du nå endrer en av filene så vil den andre oppdatere seg, og **renv** vil kunne oppdage om du bruker en pakke i koden din. Men for å trigge **renv** til å lete etter pakker som er i bruk så må du skrive følgende kode i Notebooken eller *R Console*:

```
renv::snapshot()
```

Kikker du nå inne i **renv.lock**-filen så ser du nå at verjsonen av **PxWebApiData** er lagt til. I bildet under ser du hvordan et arbeidsmiljø typisk kan se ut når man installerer sine egne pakker.



The screenshot shows the RStudio interface with a Jupyter Notebook open. The notebook contains the following R code:

```
[1]: renv::init()
# Initializing project ...
# Discovering package dependencies ... Done!
# Copying packages into the cache ... Done!
# The version of R recorded in the lockfile will be updated:
# R ["*"] -> [4.2.1]
# Lockfile written to '~/repositories/test_renv/renv.lock'.

[8]: library(PxwebApiData)
test <- ApiData("https://data.ssb.no/api/v0/en/table/04861", ..
  region = c("1883", "9381"), ContentsCode = "Boskille", Tid = c(1, 2, 3, 4))

[9]: renv::snapshot()
# Lockfile written to '~/repositories/test_renv/renv.lock'.

[ ]:
```

The **renv.lock** file is also visible in the file explorer on the left. The console output shows the following messages:

```
jovyan@jupyter-ubr-40sb-2X | +
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> Project '~/repositories/test_renv' loaded. [renv 0.16.0]
> Install packages from 'renv.lock'
Retrieving 'https://packagemanager.rstudio.com/all/_linux_/jammy/latest/src/contrib/PxwebApiData_0.7.0.tar.gz' ...
OK [Downloaded 61.3 Kb in 1.4 secs]
Retrieving 'https://packagemanager.rstudio.com/all/_linux_/jammy/latest/src/contrib/httptr 1.4.4.tar.gz' ...
```

## 22.2.2 Avinstallering

## 22.2.3 Oppgradere pakker

## 23 Bruk av pyspark i virtuelle miljøer

For å kunne benytte det virtuelle miljøet i en notebook må man sette opp en kernel. Dette er beskrevet [her](#). Denne kernelen er imidlertid ikke satt opp til å bruke pyspark som standard, og for å få til det må man gjøre noen manuelle steg.

### 23.1 Legg til pyspark i det virtuelle miljøet

Denne kommandoen legger til pakken pyspark i prosjektet (med samme versjon som på jupyterlab).

```
poetry add pyspark==$(pip show pyspark | grep Version | egrep -o "([0-9]{1,}\.)+[0-9]{1,}")
```

### 23.2 Pakkehåndtering i pyspark

Pyspark kan kjøres enten på lokal maskin eller på flere maskiner samtidig i en såkalt klynge (cluster). Sistnevnte kan være mer effektivt å bruke når man har større mengder data, men det krever også mer konfigurasjon.

#### 23.2.1 Pyspark på lokal maskin

Oppsettet for Pyspark på lokal maskin er det enkleste å sette opp siden Pyspark vil ha direkte tilgang til det lokale filsystemet. Man kan bruke miljøvariabelen `PYSPARK_PYTHON` til å peke på det virtuelle miljøet, og dermed vil Pyspark også ha tilgang til alle pakkene som er installert der. I en notebook vil dette kunne settes opp slik:

```
import os
import subprocess

# Finner filstien til det virtuelle miljøet
python_path = subprocess.run(['poetry', 'run', 'which', 'python'],
                             capture_output=True, text=True).stdout.rstrip('\n')
```

```
os.environ["PYSPARK_PYTHON"] = python_path
os.environ["PYSPARK_SUBMIT_ARGS"] = os.environ["PYSPARK_LOCAL_SUBMIT_ARGS"]
```

Til slutt må man kjøre et script for å initialisere pyspark for lokal maskin:

```
%run /usr/local/share/jupyter/kernels/pyspark_local/init.py
```

Dette scriptet vil sette et `spark` objekt som brukes for å kalle API'et til pyspark.

### 23.2.2 Pyspark i et cluster

Hvis man vil kjøre Pyspark i et cluster (dvs. på flere maskiner) så vil databehandlingen foregå på andre maskiner som ikke har tilgang til det lokale filsystemet. Man må dermed lage en “pakke” av det virtuelle miljøet på lokal maskin og tilgjengeliggjøre dette for alle maskinene i clusteret. For å lage en slik “pakke” kan man bruke et bibliotek som heter `venv-pack`. Dette kan kjøres fra et terminalvindu slik:

```
venv-pack -p .venv -o pyspark_venv.tar.gz
```

Merk at kommandoen over må kjøres fra rot-mappen i prosjektet ditt. Her er `pyspark_venv.tar.gz` et tilfeldig valgt filnavn, men dette filnavnet skal brukes videre i notebooken.

```
import os
import subprocess

# Miljøvariabel som peker på en utpakket versjon av det virtuelle miljøet
os.environ["PYSPARK_PYTHON"] = "./environment/bin/python"

# Legg til et flagg, --archives, som peker på "pakken" med det virtuelle miljøet
conf = os.environ["PYSPARK_K8S_SUBMIT_ARGS"].split(' ')
last_index = conf.index('pyspark-shell')
conf[last_index:last_index] = ['--archives', 'pyspark_venv.tar.gz#environment']
os.environ["PYSPARK_SUBMIT_ARGS"] = ' '.join(conf)
```

Til slutt må man kjøre et script for å initialisere pyspark cluster:

```
%run /usr/local/share/jupyter/kernels/pyspark_k8s/init.py
```

Dette scriptet vil sette et `spark` objekt som brukes for å kalle API'et til pyspark.

## 24 Samarbeid

Noen har opprettet et ssb-project og pushet til Github. Hvordan skal kollegaer gå frem for å bidra inn i koden?

## **25 Vedlikehold**

## **Part IV**

# **Jupyterlab på bakken**

## 26 Installere pakker

### 26.1 Python

Installering av pakker i Jupyter miljøer på bakken (f.eks <https://sl-jupyter-p.ssb.no>) foregår stort sett helt lik [som på Dapla](#). Det er én viktig forskjell, og det er at installasjon skjer via en proxy som heter Nexus.

#### 26.1.1 Pip

Pip er ferdig konfigurert for bruk av Nexus og kan kjøres som [beskrevet for Dapla](#)


#### 26.1.2 Poetry

Hvis man bruker Poetry for håndtering av pakker i et prosjekt, så må man kjøre følgende kommando i prosjekt-mappe etter prosjektet er opprettet.

```
poetry source add --default nexus `echo $PIP_INDEX_URL`
```

Da får man installere pakker som vanlig f.eks

```
poetry add matplotlib
```

 Hvis man forsøker å installere prosjektet i et annet miljø (f.eks Dapla), så må man fjerne nexus kilden ved å kjøre

```
poetry source remove nexus
```

### 26.2 R

Prosessen med å installere pakker for R på bakken er veldig lik slik det gjøres [på Dapla](#). Under beskriver hvordan det avviker fra prosedyren på Dapla.



### 26.2.1 Installering

Vi installerer fra en proxy-server på bakken, og derfor må vi spesifisere denne adressen manuelt før vi kan installere R-pakker.

```
repos <- c(CRAN = "https://nexus.ssb.no/repository/CRAN/")
options(repos = repos)
```

Deretter kan du initiere det virtuelle miljøet med følgende kommando:

```
renv::init()
```

Resten er likt som det som er forklart [for Dapla](#).

### 26.2.2 Avinstallering

### 26.2.3 Oppgradere pakker

## **27 Lese inn filer**

Mer kommer.

### **27.1 sas7bdat**

### **27.2 Oracle**

### **27.3 Fame**

### **27.4 Tekstfiler**

### **27.5 Parquet**

**Part V**

**Avansert**

## 28 Lese filer fra bømte

Skal man lese fra en bømte må man autentisere seg. For å gjøre dette kan man benytte pakken [dapla-toolbelt](#).

### 28.1 Eksempler

⚠️ “navn-boette” eksisterer ikke og må byttes med en reel bømte.

Les json fil fra bømte:

```
import dapla as dp

data_frame = dp.read_pandas("gs://ssb-staging-navn-boette/schema.json",
                             file_format="json")
```

List ut mapper i bømte:

```
from dapla import FileClient

FileClient.ls("gs://ssb-staging-navn-boette/")
```

### 28.2 Vanlige problemer

#### 28.2.1 Feil miljø

En vanlig årsak til feil er at man forsøker å lese data fra et annet miljø enn det man befinner seg i. Sjekk at url feltet i nettleseren stemmer overens med bømte man forsøker å aksessere.

Stagingbømter starter med: gs://ssb-staging-

Produksjonsbømter starter med: gs://ssb-prod-

⚠ I <https://jupyter.dapla-staging.ssb.no/> kan man ikke lese produksjonsbøtter.

⚠ I <https://jupyter.dapla.ssb.no/> kan man ikke lese stagingbøtter.

### 28.2.2 Omstart av Jupyter

Noen ganger kan en restart av Jupyter løse problemet.

I Jupyter's filmeny velg: fil -> Hub Controll Panel.

Trykk på knappen “Stop My Server”. Etter dette kan man trykke knappen “Start My Server”.



Figur 28.1: Stop My Server

### 28.2.3 Opprett TMS sak

Hvis man fortsatt ikke har tilgang, kan man opprette en [TMS sak](#). For at vi lettes mulig skal kunne hjelpe bør saken inneholde full feilmelding & relevant kode.

## **29 IDE'er**

Forklare situasjonen nå. Kun Jupyterlab. Kan kjøre remote session med Rstudio, Pycharm og VSCode.

### **29.1 RStudio**

### **29.2 VSCode**

### **29.3 Pycharm**

## 30 Scheduling

## **31 Databaser**

### **31.1 BigQuery**

### **31.2 CloudSQL**



## Referanser