

# **Kom i gang med DAPLA**

Øyvind Bruer-Skarsbø

10/9/2022

# Innhold

<b>Velkommen</b>	<b>5</b>
<b>Forord</b>	<b>6</b>
<b>I Introduksjon</b>	<b>7</b>
1 Hva er Dapla?	9
2 Hvorfor Dapla?	10
3 Arkitektur	11
4 Innlogging	12
5 Jupyterlab	13
6 Bakke vs. sky	14
<b>II Opprette Dapla-team</b>	<b>15</b>
7 Hva er Dapla-team?	17
8 Opprette Dapla-team	18
9 Google Cloud Console	19
10 Lagre data	20
11 Hente data	21
12 Fra bakke til sky	22
13 Administrasjon av team	23

<b>III Beste-praksis for koding</b>	<b>24</b>
<b>14 SSB-project</b>	<b>25</b>
14.1 Opprett GitHub-bruker . . . . .	26
14.2 To-faktor autentifisering . . . . .	26
14.3 Koble deg til SSB . . . . .	30
14.4 Personal Access Token (PAT) . . . . .	30
14.4.1 Opprette PAT . . . . .	30
14.4.2 Lagre PAT . . . . .	32
14.4.3 Oppdater PAT . . . . .	33
14.5 Opprett ssb-project . . . . .	33
<b>15 Git og Github</b>	<b>36</b>
15.1 Git . . . . .	36
15.1.1 Hva er Git? . . . . .	36
15.1.2 Oppsett av Git . . . . .	37
15.1.3 Git og Notebooks . . . . .	37
15.1.4 Vanlige Git-operasjoner . . . . .	37
15.2 GitHub . . . . .	37
<b>16 Virtuelle miljøer</b>	<b>38</b>
16.1 Python . . . . .	38
16.1.1 Anbefaling . . . . .	38
16.2 R . . . . .	38
<b>17 Jupyter-kernels</b>	<b>39</b>
<b>18 Installere pakker</b>	<b>40</b>
18.1 Python . . . . .	40
18.1.1 Poetry prosjekt eksempel . . . . .	40
18.1.2 Installering . . . . .	41
18.1.3 Avinstallering . . . . .	41
18.1.4 Oppgradere pakker . . . . .	41
18.1.5 Legge til kernel for poetry . . . . .	41
18.1.6 Fjerne kernel . . . . .	42
18.1.7 Sikkerhet . . . . .	42
18.2 R . . . . .	42
18.2.1 Installering . . . . .	42
18.2.2 Avinstallering . . . . .	44
18.2.3 Oppgradere pakker . . . . .	44
<b>19 Samarbeid</b>	<b>45</b>
<b>20 Vedlikehold</b>	<b>46</b>

<b>IV Jupyterlab på bakken</b>	<b>47</b>
<b>21 Installere pakker</b>	<b>48</b>
21.1 Python . . . . .	48
21.1.1 Pip . . . . .	48
21.1.2 Poetry . . . . .	48
21.2 R . . . . .	48
21.2.1 Installering . . . . .	49
21.2.2 Avinstallering . . . . .	49
21.2.3 Oppgradere pakker . . . . .	49
<b>22 Lese inn filer</b>	<b>50</b>
22.1 sas7bdat . . . . .	50
22.2 Oracle . . . . .	50
22.3 Fame . . . . .	50
22.4 Tekstfiler . . . . .	50
22.5 Parquet . . . . .	50
<b>V Avansert</b>	<b>51</b>
<b>23 IDE'er</b>	<b>52</b>
23.1 RStudio . . . . .	52
23.2 VSCode . . . . .	52
23.3 Pycharm . . . . .	52
<b>24 Scheduling</b>	<b>53</b>
<b>25 Databaser</b>	<b>54</b>
25.1 BigQuery . . . . .	54
25.2 CloudSQL . . . . .	54
<b>Referanser</b>	<b>55</b>

# Velkommen

DAPLA står for dataplattform og er SSBs nye plattform for statistikkproduksjon. Arbeidet startet som et utviklingsprosjekt i 2018 i sammenheng med Skatteetatens prosjekt *Sirius*. Idag er plattformen mer moden og klar for å ta imot flere statistikker. Denne boken er ment som

DAPLA står for dataplattform og er SSBs nye plattform for statistikkproduksjon. Arbeidet startet som et utviklingsprosjekt i 2018 i sammenheng med Skatteetatens prosjekt *Sirius*. Idag er plattformen mer moden og klar for å ta imot flere statistikker. Denne boken er ment som

**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 klone [dette repoet](#), gjøre endringer i en branch, og sende en pull request til administratorene av repoet (Team Statistikktenester).

# Forord

Denne boken vil la SSB-ansatte ta i bruk grunnleggende funksjonalitet på DAPLA uten hjelp fra andre.

# **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**

## 5 Jupyterlab

## **6 Bakke vs. sky**

## **Part II**

# **Opprette Dapla-team**

Gå til [Dapla Start-veilederen](#) for å opprette et nytt Dapla-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**

## **9 Google Cloud Console**

## 10 Lagre data

## 11 Hente data

## **12 Fra bakke til sky**

## **13 Administrasjon av team**

## **Part III**

# **Beste-praksis for koding**



## 14 SSB-project

Produksjonsløp på **Dapla** kan med fordel følge noen helt klare retningslinjer for arbeidsprosesser og kode. Dette bør blant annet inkludere:

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 at publiserte tall skal være reproducerbare er SSB avhengig av at blant annet pakkeversjoner og versjon av Python/R lagres sammen med kode som er kjørt.

3. **Versjonshåndtering med Git**

Versjonshåndtering av kode er svært viktig for å kunne gjenskape og samarbeide om kode. [Git](#) er verdensstandarden for å gjøre dette, og derfor legges det opp til at all kode skal versjonshåndteres med Git i SSB.

4. **Lagre kode på Github**

På Dapla er det ingen fellesmappe som alle i SSB har tilgang til og hvor vi kan dele kode slik vi har gjort i bakkemiljøet tidligere. Kode som er versjonshåndtert med Git bruker som regel et remote repo<sup>1</sup> som er spesialsydd for Git og som skal deles med resten av verden hvis man ønsker. I SSB har vi valgt å bruke GitHub, der SSB har et eget område som heter [statisticsnorway](#).

[Team Statistikkjenester](#) har laget en CLI<sup>2</sup> som skal gjøre dette lett å implementere dette i kode. Den heter [ssb-project](#) og hjelper deg implementere det som til enhver tid er beste-praksis for koding.

Under vises det hvordan man bruker **ssb-project** til sette opp et prosjekt. Men programmet forutsetter at du har en GitHub-bruker som er knyttet opp mot [statisticsnorway](#). De første underkapitlene er derfor en beskrivelse av dette.

---

<sup>1</sup>Remote repo er en felle mappe som er lagret på en annen maskin. [Les mer her](#).

<sup>2</sup>CLI = Command-Line-Interface. Dvs. et program som er skrevet for å brukes terminalen ved hjelp av enkle kommandoer.

## 14.1 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 er 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.

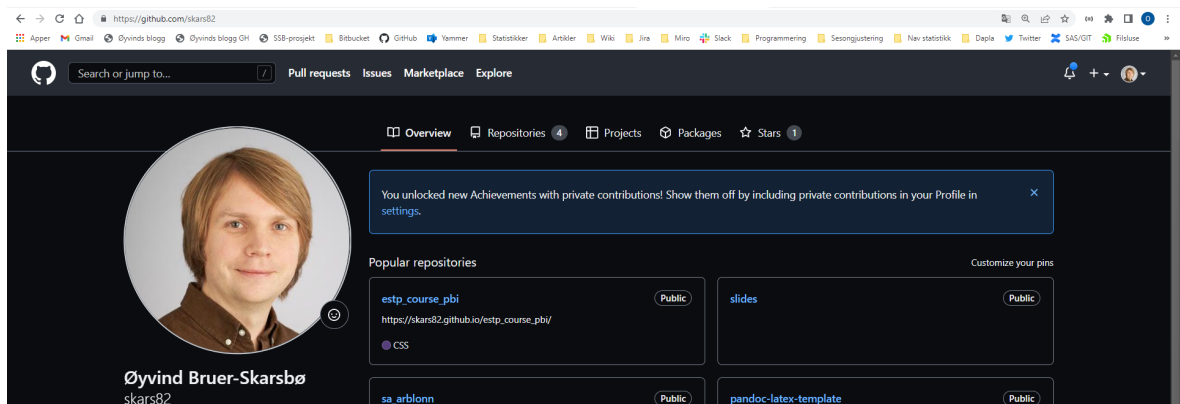
Slik gjør du det:

1. Gå til <https://github.com/>
2. Trykk **Sign up** øverst i høyre hjørne
3. Svar på spørsmålene du blir stilt.

Husk at du lager en personlig konto uavhengig av SSB. Brukernavnet kan være noe annet enn brukernavnet ditt i SSB. I neste steg skal vi knytte denne kontoen til din SSB-bruker.

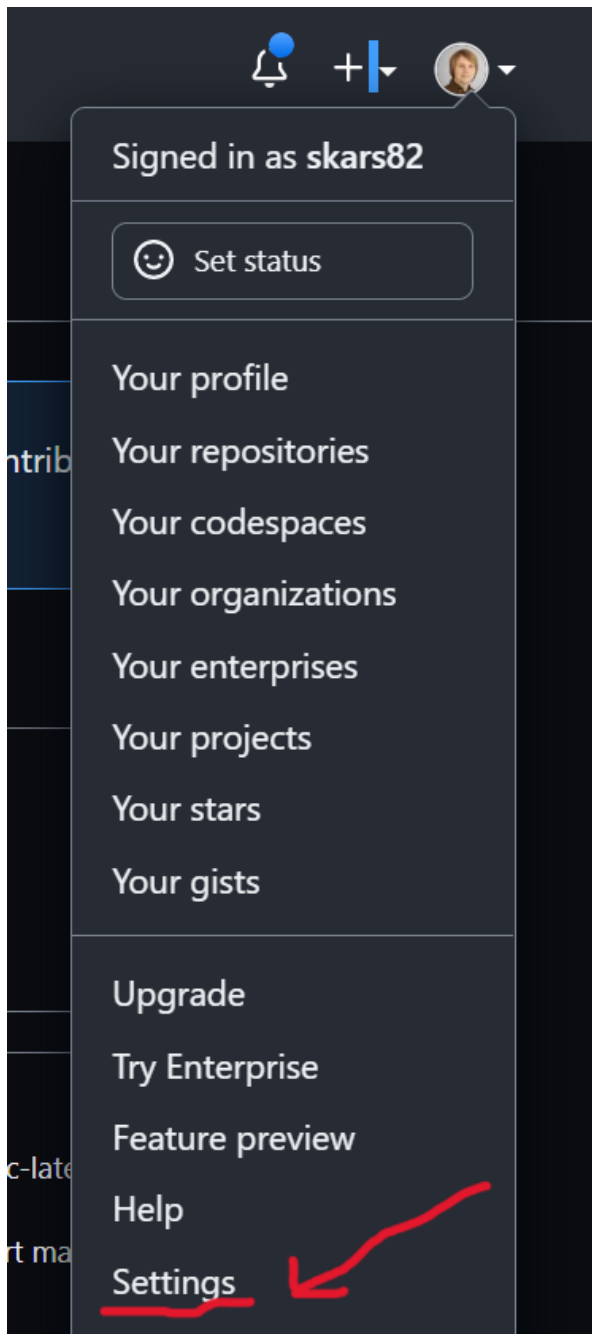
## 14.2 To-faktor autentifisering

Hvis du har fullført forrige steg så har du nå en GitHub-konto. Hvis du står på din profil-side så ser den slik ut:

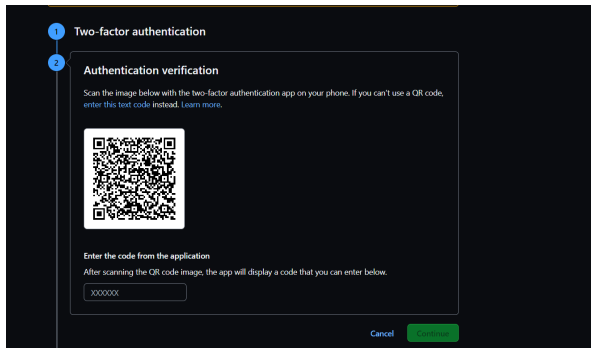


Det neste vi må gjøre er å aktivere 2-faktor autentifisering, siden det er dette sin benyttes i SSB. Hvis du står på siden i bildet over, så gjør du følgende for å aktivere 2-faktor autentifisering mot GitHub:

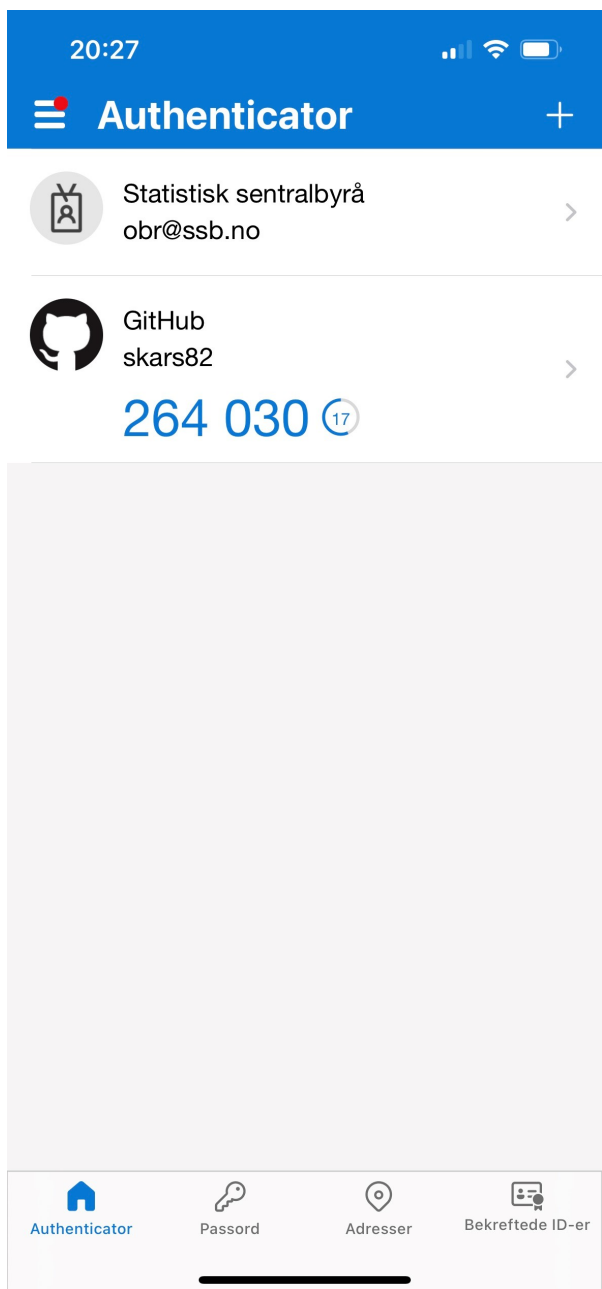
1. Trykk på den lille pilen øverst til høyre og velg **Settings**(se bilde til høyre).
2. Deretter velger du **Password and authentication** i menyen til venstre.



3. Under **Two-factor authentication** trykker du på **Add**. Da får du opp følgende bilde:



4. Strekkoden over skal skannes i din **Microsoft Authenticator**-app på mobilen. Åpne appen, trykk på **Bekreftede ID-er**, og til slutt trykk på **Skann QR-kode**. Deretter skanner du QR-koden fra punkt 3.
5. 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.
6. Til slutt lagrer du **Recovery-codes** et trygt sted på ditt hjemmeområdet.



Nå har vi aktivert 2-faktor autentifisering for GitHub og er klare til å knytte vår personlige konto til vår SSB-bruker på **statisticsnorway**.

## 14.3 Koble deg til SSB

I forrige steg aktiverte vi 2-faktor autentifisering for GitHub. Det neste vi må gjøre er å bruke denne autentifiseringen til koble oss til SSB sin bedriftskonto **statisticsnorway**. Det er dette som gjør at vi kan jobbe med SSB-kode som ligger lagret på GitHub.

1. Gå til profilsiden din og velg **Settings** slik du gjorde i punkt 1 i forrige delkapitel.
2. Trykk deretter på **Organizations** i menyen til venstre.
3. Trykk deretter på **New organization**.
4. Søk etter **statisticsnorway**.

## 14.4 Personal Access Token (PAT)

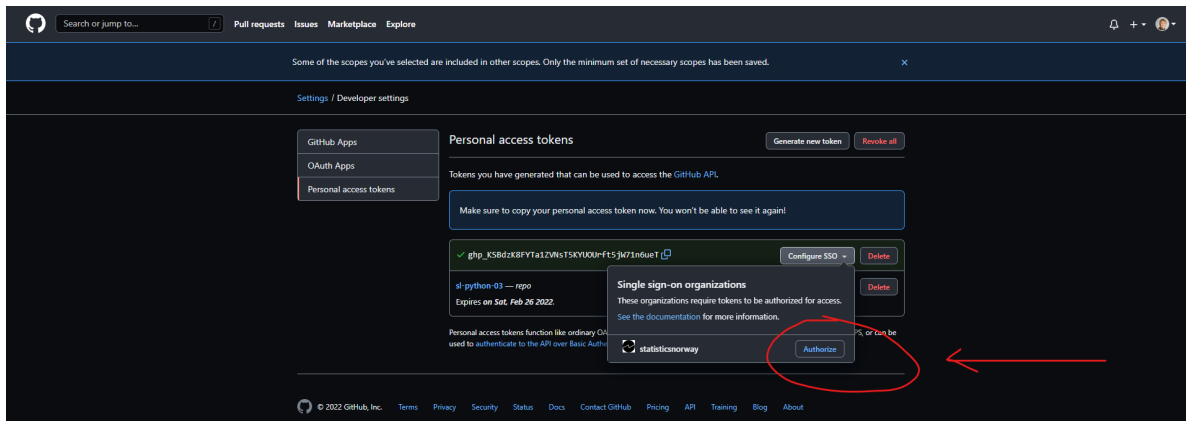
Når vi skal jobbe med SSB-kode som ligger lagret hos **statisticsnorway** på GitHub, så må vi autentifisere oss. Måten vi gjøre det på er ved å generere et **Personal Access Token** (ofte forkortet *PAT*) som vi oppgir når vi vil hente eller oppdatere kode på GitHub. Da sender vi med PAT for å autentifisere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

### 14.4.1 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 Section 14.2.
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 *prodson* 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. En grei mellomløsning kan være å velge 3 måneder. Når PAT går ut må du gjenta stegene i dette kapitlet.
6. Under **Select scopes** velger du **Repo** (se bilde under).





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 sendte med dette tokenet for å få lov til å jobbe med koden som ligger på **statisticsnorway** på GitHub.

## 14.4.2 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

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.

### 14.4.3 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 Section [14.4.1](#).
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 *staisticsnorway* på **GitHub**.

## 14.5 Opprett ssb-project

Programmet [ssb-project](#) skal gjøre det enklere å organiserer kode etter god praksis i en statistikkproduksjoner. Som nevnt i innledningen vil programmet gi deg en mappestruktur, virtuelt miljø og en jupyter-kernel. Hvis du ønsker kan det også opprette et GitHub-repo. Ønsker du alt 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** Deretter skriver du følgende i terminalen:

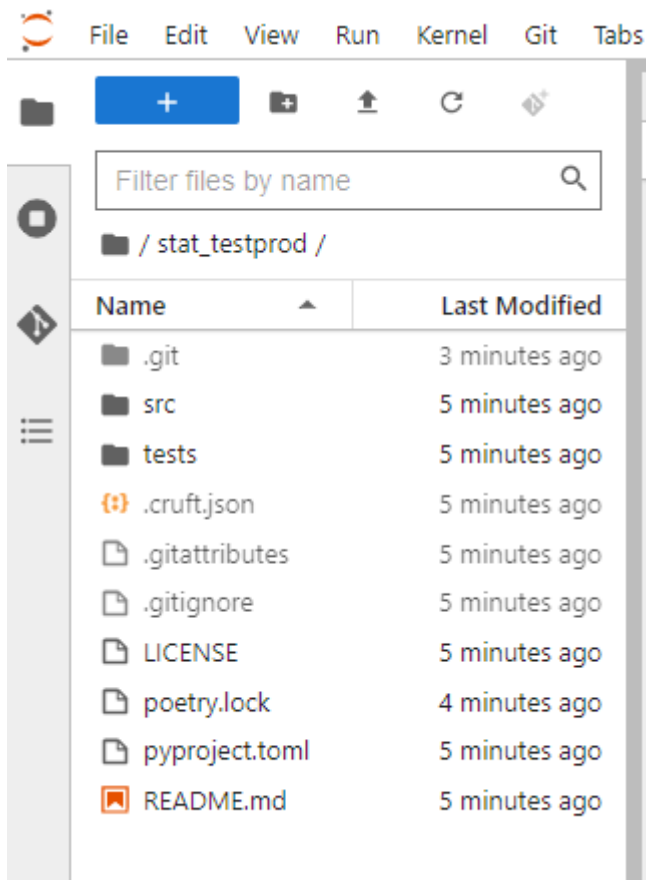
```
ssb-project create stat-testprod
```

Hvis du stod i hjemmemappen din på når du skrev inn kommandoen over i terminalen, så har du fått mappestrukturen som vises i bildet til høyre. <sup>3</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.

---

<sup>3</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.



Over så opprettet vi et ssb-project uten å opprette et GitHub-repo med samme navn. 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. Repoet i GitHub ser da slik ut:

[Pull requests](#)
[Issues](#)
[Marketplace](#)
[Explore](#)

[statisticsnorway/stat\\_testprod](#)
Private

[Watch](#)
[Fork](#)

[Code](#)
[Issues](#)
[Pull requests](#)
[Actions](#)
[Projects](#)
[Wiki](#)
[Security](#)
[Insights](#)
[Settings](#)

[main](#)
[1 branch](#)
[0 tags](#)

[Go to file](#)
[Add file](#)
[Code](#)

[About](#)

skars82

Initial commit

9e3e212

2 minutes ago

1 commit

src	Initial commit	2 minutes ago
tests	Initial commit	2 minutes ago
.craft.json	Initial commit	2 minutes ago
.gitattributes	Initial commit	2 minutes ago
.gitignore	Initial commit	2 minutes ago
LICENSE	Initial commit	2 minutes ago
README.md	Initial commit	2 minutes ago
pyproject.toml	Initial commit	2 minutes ago

README.md

stat\_testprod

Dette er en test for dapla-manual.

Opprettet av Øyvind Bruer-Skarsbø [obr@esb.no](#)

Dette er en test for dapla-manual.

[sub-project](#)

Readme

MIT license

0 stars

3 watching

0 forks

Releases

No releases published

[Create a new release](#)

Packages

No packages published

[Publish your first package](#)

Languages

Jupyter Notebook 65.4%

Python 34.6%

# 15 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 fungerer uten dette hjelpemiddelet. Forhåpentligvis vil det gjøre det lettere å håndtere mer kompliserte situasjoner som oppstår i arbeidshverdagen som statistikker.

## 15.1 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**.

### 15.1.1 Hva er Git?

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

### **15.1.2 Oppsett av Git**

Kommer snart. Scriptet til Kvakk.

### **15.1.3 Git og Notebooks**

Kommer snart. Jupyter og nbsripout. json.

### **15.1.4 Vanlige Git-operasjoner**

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

## **15.2 GitHub**

# 16 Virtuelle miljøer

## 16.1 Python

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

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

### 16.1.1 Anbefalning

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

Poetry setter opp virtuetl miljø, gjør det enkelt å oppdatere avhengigheter, sette versjons begrensninger 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.

## 16.2 R

## 17 Jupyter-kernels

# 18 Installere pakker

## 18.1 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](#).

### 18.1.1 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
```



### 18.1.2 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](#).

### 18.1.3 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
```

### 18.1.4 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
```

### 18.1.5 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.

### 18.1.6 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.

### 18.1.7 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?

## 18.2 R

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

### 18.2.1 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:

```
install.packages("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))
```

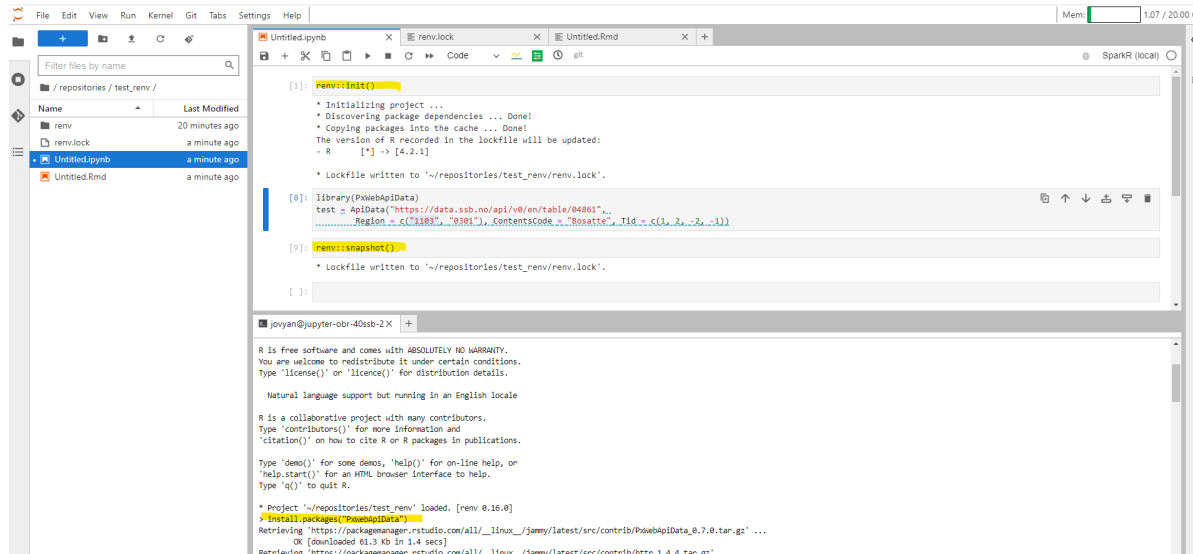
Når vi nå har brukt **PxWebApiData** i koden vår så kan vi kjøre en kommando som legger til den pakken i **renv.lock**. Men før vi kan gjøre det må vi være obs på at **renv** ikke klarer å gjenkjenne pakker som er i bruk Notebooks (ipynb-filer). Det er veldig upraktisk, men noe vi må forholde oss til 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("1801", "0301"), ContentsCode = "Roskilde", 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-40sdb-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('PxWebApiData')
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/http 1.4.4.tar.gz' ...
```

## 18.2.2 Avinstallering

## 18.2.3 Oppgradere pakker

## 19 Samarbeid

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

## 20 Vedlikehold

## **Part IV**

# **Jupyterlab på bakken**

# 21 Installere pakker

## 21.1 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.

### 21.1.1 Pip

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


### 21.1.2 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
```

## 21.2 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.



### 21.2.1 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](#).

### 21.2.2 Avinstallering

### 21.2.3 Oppgradere pakker

## **22 Lese inn filer**

Mer kommer.

### **22.1 sas7bdat**

### **22.2 Oracle**

### **22.3 Fame**

### **22.4 Tekstfiler**

### **22.5 Parquet**

**Part V**

**Avansert**

## **23 IDE'er**

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

### **23.1 RStudio**

### **23.2 VSCode**

### **23.3 Pycharm**

## 24 Scheduling

## **25 Databaser**

### **25.1 BigQuery**

### **25.2 CloudSQL**

## Referanser