

Kom i gang med DAPLA

Øyvind Bruer-Skarsbø

10/9/2022

Innhold

Velkommen	6
Forord	7
I Introduksjon	8
1 Hva er Dapla?	10
2 Hvorfor Dapla?	11
3 Arkitektur	12
4 Innlogging	13
4.1 Dapla	13
4.1.1 Prod	14
4.1.2 Staging	15
4.2 Bakkemiljøet	15
4.2.1 Prod	15
4.2.2 Staging	15
5 Jupyterlab	16
5.1 Hva er Jupyterlab?	16
5.2 Terminalen	16
5.3 Pakkeinstallasjoner	16
5.4 Extensions	16
5.5 Tips & triks	16
6 Bakke vs. sky	17
II Opprette Dapla-team	18
7 Hva er Dapla-team?	20
8 Opprette Dapla-team	21

9 Bøtter	22
9.1 Bøttenavn i Produksjonsmiljøet	22
9.2 Andre miljøer	23
9.3 Fellesbøtter	23
10 Google Cloud Console	24
11 Lagre data	25
12 Hente data	26
13 Fra bakke til sky	27
14 Administrasjon av team	28
 III Beste-praksis for koding	 29
15 SSB-project	30
15.1 Opprett GitHub-bruker	31
15.2 To-faktor autentifisering	31
15.3 Koble deg til SSB	35
15.4 Personal Access Token (PAT)	35
15.4.1 Opprette PAT	35
15.4.2 Lagre PAT	37
15.4.3 Oppdater PAT	38
15.5 Opprett ssb-project	38
16 Git og Github	41
16.1 Git	41
16.1.1 Hva er Git?	41
16.1.2 Oppsett av Git	42
16.1.3 Git og Notebooks	42
16.1.4 Vanlige Git-operasjoner	42
16.2 GitHub	42
17 Virtuelle miljøer	43
17.1 Python	43
17.1.1 Anbefaling	43
17.2 R	43
18 Jupyter-kernels	44

19	Installere pakker	45
19.1	Python	45
19.1.1	Poetry prosjekt eksempel	45
19.1.2	Installering	46
19.1.3	Avinstallering	46
19.1.4	Oppgradere pakker	46
19.1.5	Legge til kernel for poetry	46
19.1.6	Fjerne kernel	47
19.1.7	Sikkerhet	47
19.2	R	47
19.2.1	Installering	47
19.2.2	Avinstallering	49
19.2.3	Oppgradere pakker	49
20	Samarbeid	50
21	Vedlikehold	51
IV	Jupyterlab på bakken	52
22	Installere pakker	53
22.1	Python	53
22.1.1	Pip	53
22.1.2	Poetry	53
22.2	R	53
22.2.1	Installering	54
22.2.2	Avinstallering	54
22.2.3	Oppgradere pakker	54
23	Lese inn filer	55
23.1	sas7bdat	55
23.2	Oracle	55
23.3	Fame	55
23.4	Tekstfiler	55
23.5	Parquet	55
V	Avansert	56
24	Lese filer fra bønne	57
24.1	Eksempler	57
24.2	Vanlige problemer	57
24.2.1	Feil miljø	57

24.2.2 Omstart av Jupyter	58
24.2.3 Opprett TMS sak	58
25 IDE'er	59
25.1 RStudio	59
25.2 VSCode	59
25.3 Pycharm	59
26 Scheduling	60
27 Databaser	61
27.1 BigQuery	61
27.2 CloudSQL	61
Referanser	62

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](#) 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 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

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 nettverk¹. For å gjøre det enda enklere har vi laget en fast snarvei til denne nettadressen [på vårt intranett/Byrånettet](#)(se Figure 4.1).

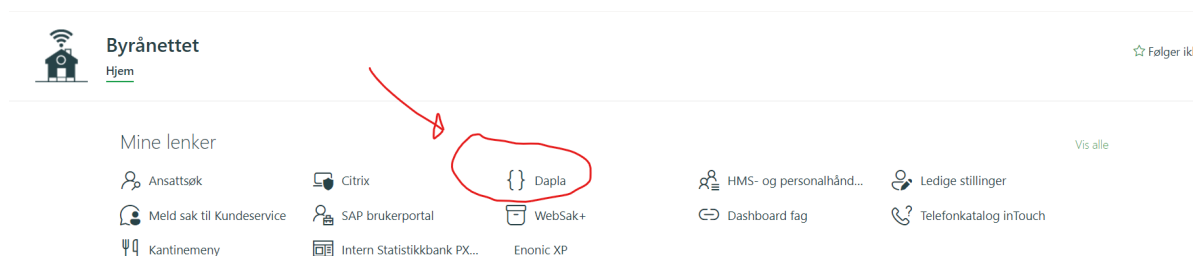


Figure 4.1: Snarvei til Dapla fra intranett

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. Figure 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 statistikker 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

¹Å 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.

Bakkemiljø	Skymiljø/Dapla
PROD	PROD
STAGING	STAGING

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

som vi kaller **staging-området**. Årsaken er at vi ønsker å beskytte prod-miljøet fra software 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 Figure 4.1.
2. Deretter blir man spurt om man godtar at SSB-profilen din blir brukt av Google for innlogging. Trykk **Tillat**.
3. Deretter lander man på en side som lar deg avgjøre hvor mye maskinkraft som skal holdes av til deg:
 - a. **Lite ressurser**
Betyr at man får...kommer snart
 - b. **Mer ressurser**
Betyr at man får...kommer snart
 - c. **Mye ressurser**
Betyr at man får...kommer snart

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-image²](#) 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 Figure 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/>.

²Hva 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

Dette kapitlet beskriver hva et Dapla-team er og hvordan man går frem for å opprette et nytt team på Dapla.

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.

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 Lagre data

12 Hente data

13 Fra bakke til sky

14 Administrasjon av team

Part III

Beste-praksis for koding

15 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 repo¹ 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 Statistikktenester](#) har laget en CLI² 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 forutsettet at du har en GitHub-bruker som er knyttet opp mot [statisticsnorway](#). De første underkapitlene er derfor en beskrivelse av dette.

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

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

15.1 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.

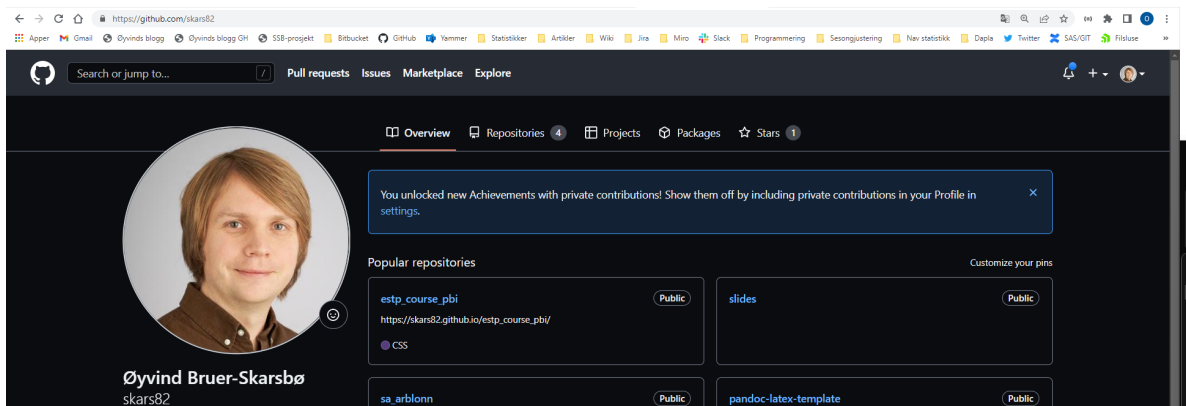
Slik gjør du det:

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.

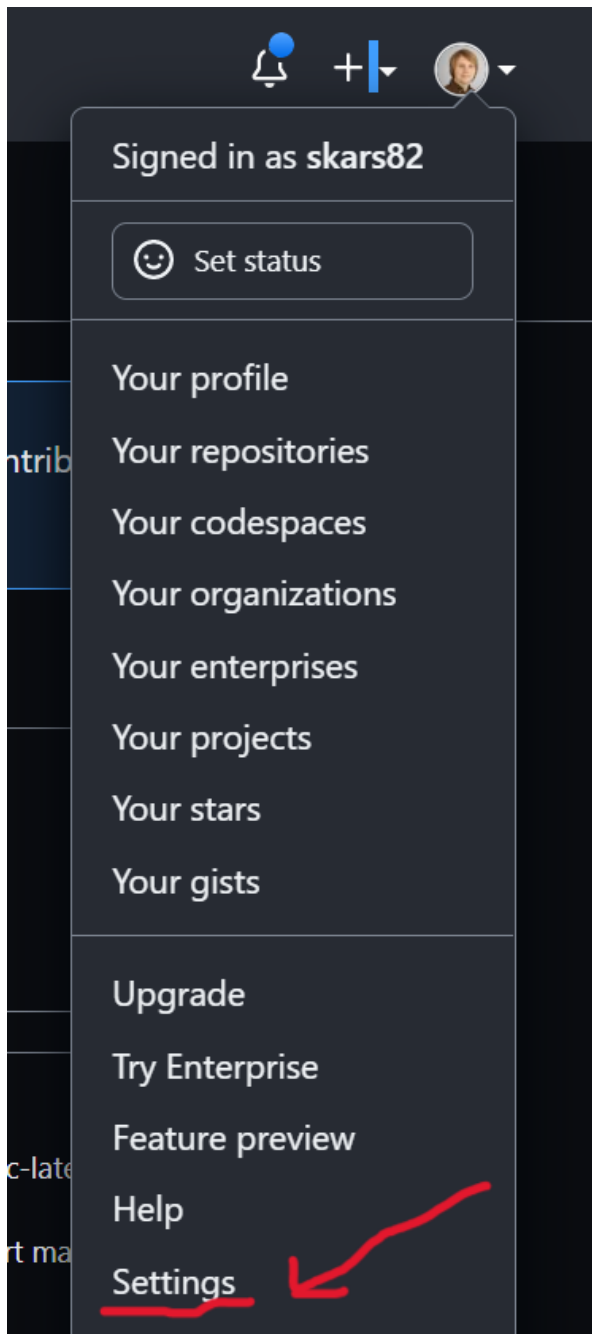
15.2 To-faktor autentifisering

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:

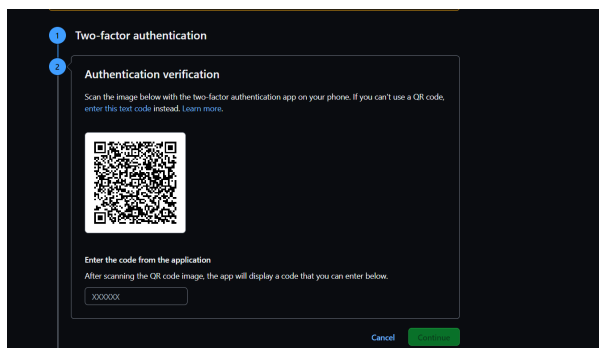


Det neste vi må gjøre er å aktivere 2-faktor autentifisering, siden det er dette sin benyttes i SSB. Hvis du står på siden i bildet over, så gjør du følgende for å aktivere 2-faktor autentifisering mot GitHub:

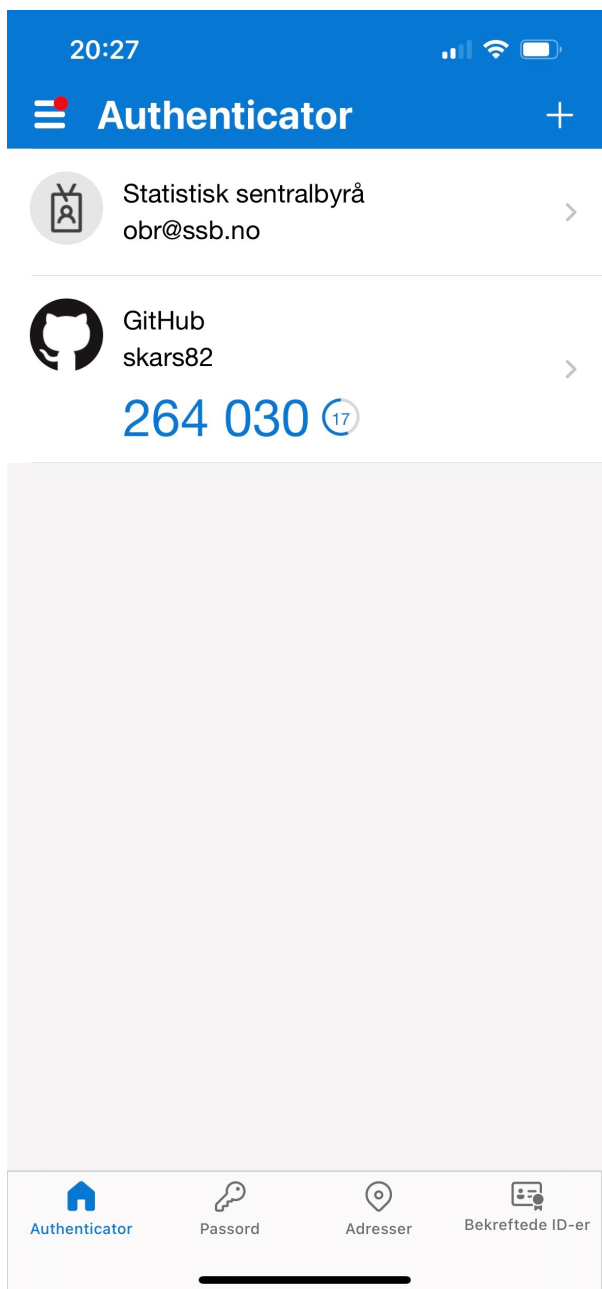
1. Trykk på den lille pilen øverst til høyre og velg **Settings**(se bilde til høyre).
2. Deretter velger du **Password and authentication** i menyen til venstre.



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**.

15.3 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**.

15.4 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.

15.4.1 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 [15.2](#).
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).

Settings / Developer settings

GitHub Apps
OAuth Apps
Personal access tokens
Fine-grained tokens (Beta)
Tokens (classic)

New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note
dapla

What's this token for?

Expiration *
90 days The token will expire on Sat, Jan 28 2023

Select scopes
Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events
<input type="checkbox"/> workflow	Update GitHub Action workflows

8. Trykk på **Generate token** nederst på siden og du får opp noe som ser ut som dette:

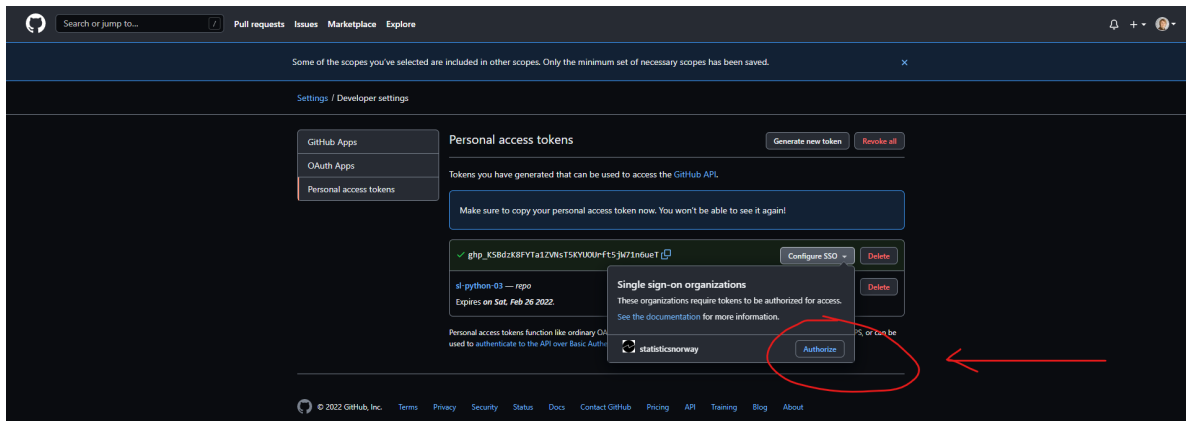
Personal access tokens (classic) Generate new token ▼ Revoke all

Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your personal access token now. You won't be able to see it again!

✓ ghp_4pPvt6gTSjlbn60EKC9fU164UmSFfu01I8Lo Configure SSO ▼ Delete

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](#). Svar deretter på spørsmålene som dukker opp.



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.

15.4.2 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.

15.4.3 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 [15.4.1](#).
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**.

15.5 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:

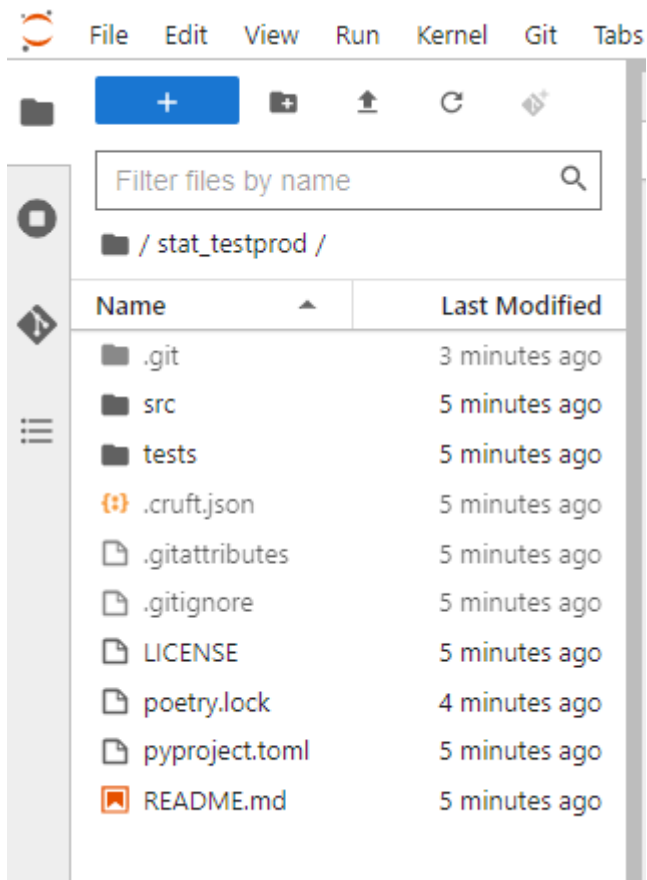
1. Åpne en terminal. De fleste vil gjøre dette i Jupyterlab på bakke eller sky og da kan de bare trykke på det blå +-tegnet i Jupyterlab og velge **Terminal**.
2. Før vi kjører programmet må vi være obs på at **ssb-project** vil opprette en ny mappe der vi står. Gå derfor til den mappen du ønsker å ha 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. ³. 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.

³Filer 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%

16 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.

16.1 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**.

16.1.1 Hva er Git?

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

16.1.2 Oppsett av Git

Kommer snart. Scriptet til Kvakk.

16.1.3 Git og Notebooks

Kommer snart. Jupyter og nbsripout. json.

16.1.4 Vanlige Git-operasjoner

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

16.2 GitHub

17 Virtuelle miljøer

17.1 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](#).

17.1.1 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.

17.2 R

18 Jupyter-kernels

19 Installere pakker

19.1 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](#).

19.1.1 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
```

19.1.2 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](#).

19.1.3 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
```

19.1.4 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
```

19.1.5 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.

19.1.6 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.

19.1.7 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?

19.2 R

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

19.2.1 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))
```

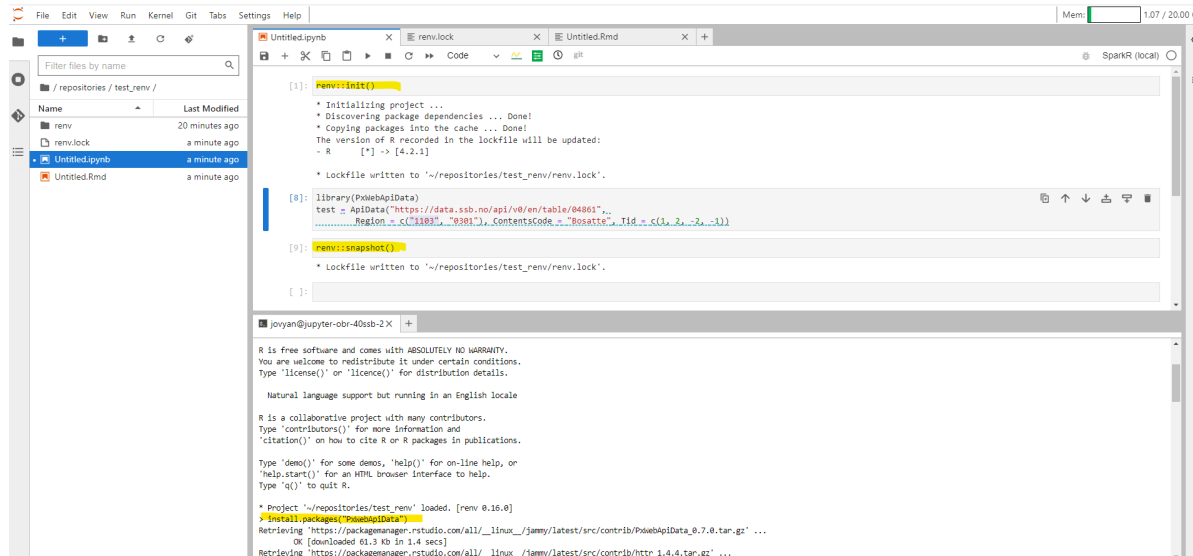
Når vi nå har brukt **PxWebApiData** i koden vår så kan vi kjøre en kommando som legger til den pakken i **renv.lock**. Men før vi kan gjøre det må vi være obs på at **renv** ikke klarer å gjenkjenne pakker som er i bruk Notebooks (ipynb-filer). Det er veldig upraktisk, men noe vi må forholde oss til 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.



```
[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 = "Boskille", Tid = c(1, 2, 3, 4))

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

[ ]:
```

```
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 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' ...
```

19.2.2 Avinstallering

19.2.3 Oppgradere pakker

20 Samarbeid

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

21 Vedlikehold

Part IV

Jupyterlab på bakken

22 Installere pakker

22.1 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.

22.1.1 Pip

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


22.1.2 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
```

22.2 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.

22.2.1 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](#).

22.2.2 Avinstallering

22.2.3 Oppgradere pakker

23 Lese inn filer

Mer kommer.

23.1 sas7bdat

23.2 Oracle

23.3 Fame

23.4 Tekstfiler

23.5 Parquet


Part V

Avansert

24 Lese filer fra bømte

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

24.1 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/")
```

24.2 Vanlige problemer

24.2.1 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 bømtenes 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.

24.2.2 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”.

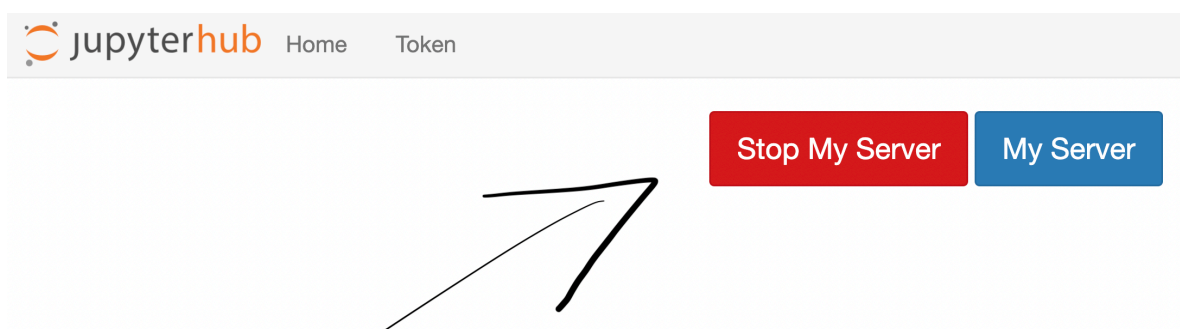


Figure 24.1: Stop My Server

24.2.3 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.

25 IDE'er

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

25.1 RStudio

25.2 VSCode

25.3 Pycharm

26 Scheduling

27 Databaser

27.1 BigQuery

27.2 CloudSQL

Referanser