

Kom i gang med DAPLA

Øyvind Bruer-Skarsbø

10/9/2022

Innhold

Velkommen	5
Forord	6
I Introduksjon	7
1 Hva er Dapla?	9
2 Hvorfor Dapla?	10
3 Arkitektur	11
4 Innlogging	12
5 Jupyterlab	13
6 Bakke vs. sky	14
II Opprette Dapla-team	15
7 Hva er Dapla-team?	16
8 Opprette Dapla-team	17
9 Google Cloud Console	18
10 Lagre data	19
11 Hente data	20
12 Fra bakke til sky	21
13 Administrasjon av team	22

III Beste-praksis for koding	23
14 SSB-project	24
14.1 Opprett GitHub-bruker	25
14.2 Koble seg til SSB	25
14.3 Autentifisering	25
14.4 ssb-project-cli	25
15 Git og Github	26
16 Virtuelle miljøer	27
16.1 Python	27
16.1.1 Anbefaling	27
16.2 R	27
17 Jupyter-kernels	28
18 Installere pakker	29
18.1 Python	29
18.1.1 Poetry prosjekt eksempel	29
18.1.2 Installering	30
18.1.3 Avinstallering	30
18.1.4 Oppgradere pakker	30
18.1.5 Legge til kernel for poetry	30
18.1.6 Fjerne kernel	31
18.1.7 Sikkerhet	31
18.2 R	31
18.2.1 Installering	31
18.2.2 Avinstallering	31
18.2.3 Oppgradere pakker	31
19 Samarbeid	32
20 Vedlikehold	33
IV Jupyterlab på bakken	34
21 Installere pakker	35
21.1 Python	35
21.1.1 Pip	35
21.1.2 Poetry	35
21.2 R	36
21.2.1 Installering	36

21.2.2	Avinstallering	36
21.2.3	Oppgradere pakker	36
22	Lese inn filer	37
22.1	sas7bdat	37
22.2	Oracle	37
22.3	Fame	37
22.4	Tekstfiler	37
22.5	Parquet	37
V	Avansert	38
23	IDE'er	39
23.1	RStudio	39
23.2	VSCode	39
23.3	Pycharm	39
24	Schedulering	40
25	Databaser	41
25.1	BigQuery	41
25.2	CloudSQL	41
	Referanser	42

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?

2 Hvorfor Dapla?

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

7 Hva er Dapla-team?

Mer kommer

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

Fremtidens produksjonsløp på **Dapla** bør 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 Statistikkjenester](#) 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 forutsetter 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.

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 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 Koble seg til SSB

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

14.3 Autentifisering

14.4 `ssb-project-cli`

15 Git og 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 virtuelt 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

18.2.1 Installering

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 fjerner nexus kilden ved å kjøre

```
poetry source remove nexus
```

21.2 R

21.2.1 Installering

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