Kobling B

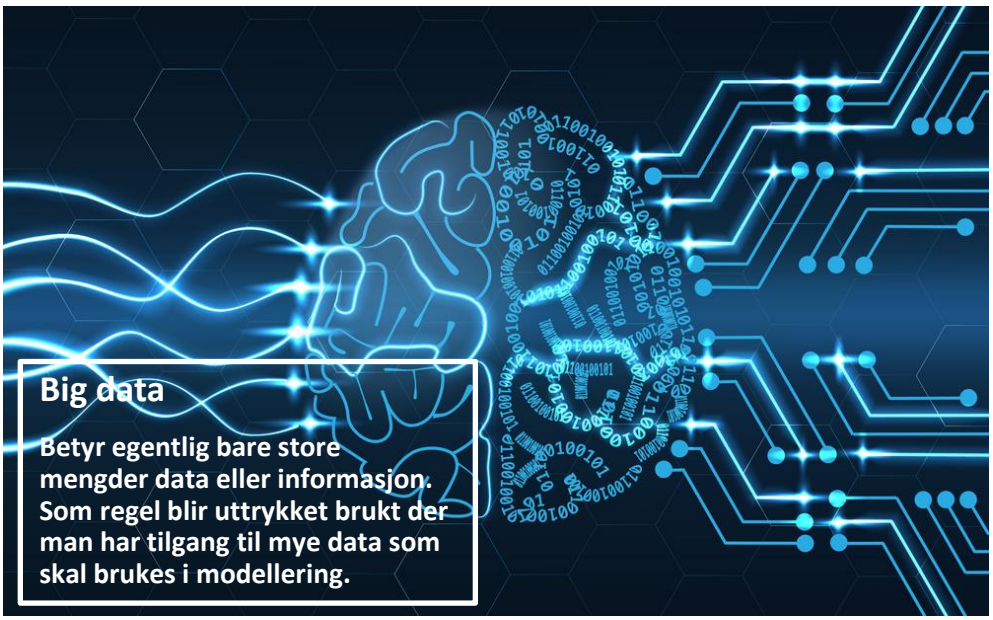
Programmering og skaperverksted for 8. - 10. klasse

Kapittel 13

Innhold

Kapittel 13 – Modellering med Python (for de spesielt interesserte)

Modellering og maskinlæring med Python



Maskinlæring – hva er det?

Maskinlæring er en del av fagområdet kunstig intelligens, men det er mindre avansert enn man skulle tro. Den mest grunnleggende formen for maskinlæring er å lage matematiske modeller ved hjelp av enkle regresjoner slik som dere tidligere har gjort i Geogebra.

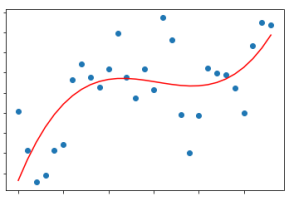
Etter hvert kan man bruke mer og mer avanserte modeller, og gjerne systemer som består av mange modeller som henger sammen med hverandre. Et eksempel på et slikt gigantisk system er værmodellering. Der hentes det inn allslags data om temperaturer, vindstyrke og lufttrykk fra mange steder i verden. Disse dataene brukes for å lage matematiske modeller, altså forskjellige matematiske funksjoner, slik dere har gjort tidligere. Videre kan meteorologene sette inn nye verdier i funksjonene sine, for å regne ut hvilket vær det er størst sannsynlighet for å bli i forskjellige områder på et senere tidspunkt. Hvordan henger dette sammen med gyldighetsområde og usikkerhet i målinger?

Mye modellering foregår uten at mennesker aktivt programmerer alle delene, programmene har «fått lov til» å lage sine egne matematiske modeller basert på forskjellige datasett. Dette kan gjøre det vanskeligere å vite hva som ligger bak avgjørelser som disse dataprogrammene gjør, da en del av algoritmene kan være vanskelig å få tilgang til. På en måte er det fint at maskiner kan gjøre dette for oss, på den andre siden kan det for eksempel kunne føre til problemer med diskriminering.

Diskriminering på grunn av maskinlæring har man sett flere eksempler på. Hvis et maskinlærings-program for ansiktsgjenkjenning får tilgang til datasett som nesten bare består av hvite menn, vil dette programmet bli mye dårligere på å kjenne igjen ansiktene til kvinner og ikke-hvite mennesker. Dette har man kunnet merke på enkelte mobilmerkers ansiktsgjenkjenning for å låse opp telefonen. Det er også funnet eksempler på diskriminering i ansettelser, for eksempel der et maskinlæringsprogram foreslo Amazon å heller ansette menn enn kvinner, basert på inndata om at det var veldig mange flere menn enn kvinner ansatt der i utgangspunktet. Les gjerne mer her:
https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G

Fra data

Via regresjon

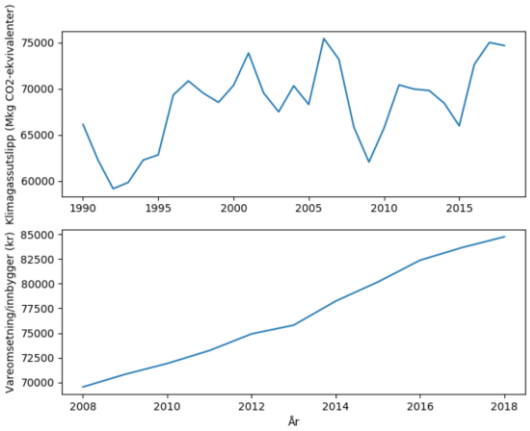


Til modeller

Lage graf med Python

Hvorfor Python?
Dere kan allerede tegne grafer med Geogebra, så hva er poenget med å bruke Python i stedet?

I Geogebra kan vi plotte en funksjon, eller legge inn punkter og få plottet en graf fra punktene. Siden vi må skrive inn alle punktene for hånd, kan det være tungvint dersom vi har veldig mange målinger. Dette gjelder spesielt om vi har ferdige filer med mange målinger. I Python kan man lage sub-plot, det vil si bilder med flere grafer som viser forskjellige målinger i samme bilde. Men hovedgrunnen til å bruke Python til plotting av grafer, er at det er begynnelsen på å bruke statistikk for å analysere data. Det er slik forskere jobber når de har samlet inn data!



I Python kan du bruke data som ligger i ferdige filer. Du slipper å skrive dem inn! Det som er viktig er at filene er riktig type. Filene kan bare inneholde tall, og lagres i tekstformat (.txt) for de eksemplene vi ser på her.

Plotting av grafer i Python består av seks deler:

1. Importere pylab-biblioteket i programmet
2. Importere data fra en .txt-fil
3. Legge dataene i en tabell (array) for x-verdiene og en for y-verdiene
4. Skrive hvilke data du skal plotte
5. Skrive inn tittel og merkelapper på aksene
6. Få plottet til å vises på skjermen

I tillegg kan du skrive inn hvordan grafen skal se ut med for eksempel farge, linjetykkelse og linjestil.

```
from pylab import *
```
→ Importerer det nødvendige biblioteket

```
plot(x, y)
```
→ Lager et plot av alle verdiene i tabellene x og y

```
show()
```
→ Viser figuren av plottet på skjermen

```
tabell = loadtxt("data.txt")
```
→ Importerer filen data.txt og legger den inn i en tabell ved navn tabell

```
x = tabell[:,0]
y = tabell[:,1]
```
→ Plukker alle x-verdiene fra tabellen og legger dem i en tabell som heter x. Tilsvarende for y-verdiene.

```
title("Tittel")
xlabel("x-verdier")
ylabel("y-verdier")
```
→ Lager tittel og merkelapper til aksene.

Oppgaver

1. Bruk kodelinjene på venstre side for å lage et program som plotter en graf fra en fil som heter data.txt.
2. Skriv en beskrivelse for hver del av programmet ditt uten å se på denne siden. Legg dem gjerne inn som kommentarer i koden din. Sammenlign så beskrivelsene dine med de på denne siden. Hvilke forskjeller fant du?
3. Finn et valgfritt datasett fra statistisk sentralbyrå, og bruk Python til å lage en graf av dataene. Hva må du endre i eksempelet på denne siden?

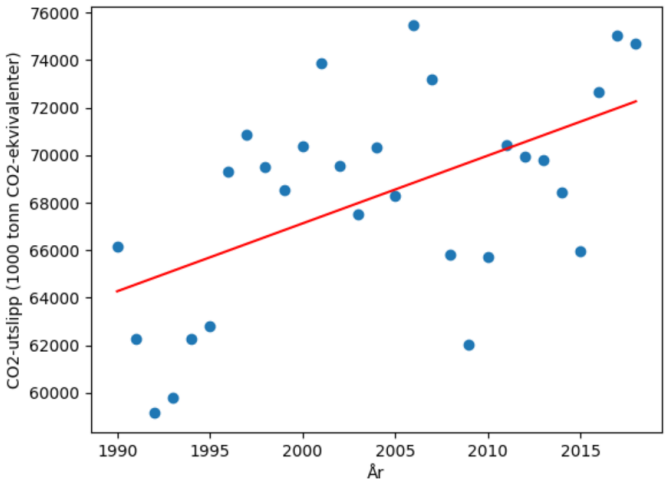
Lineær regresjon med Python

Hvorfor Python?

Dere har allerede utført mange lineære regresjoner med Geogebra, så hva er poenget med å bruke Python i stedet?

I Geogebra må vi skrive inn alle målingene våre for hånd, eller kopiere fra f.eks. Excel, og da kan det være vanskelig å gjøre en regresjon dersom vi har mye data. Tenk deg at du har 100 målinger, da tar det veldig lang tid å skrive inn alle målingene i regnearket i Geogebra. Det kan fort bli en feil eller to underveis også. Tenk om du har enda flere målinger, kanskje flere enn 1000! Da blir det ganske kjedelig å skrive inn alle tallene.

Samlet klimagassutslipp for Norge 1990 – 2018



I Python kan du bruke data som ligger i ferdige filer. Du slipper å skrive dem inn! Det som er viktig er at filene er riktig type. Slik som med plotting av grafer i Python.

En annen fordel med å bruke Python, er at det er et programmeringsspråk som blir brukt av forskere som jobber med modellering. Andre forskere kan bruke andre programmeringsspråk som er spesialtilpasset akkurat den modelleringen de jobber med, men de følger de samme prinsippene som Python. Så dersom du kan modellere med Python, er det enklere å modellere i et annet programmeringsspråk enn om du bare har brukt Geogebra tidligere.

Regresjon i Python består av seks deler:

1. Importere de bibliotekene som trengs
2. Importere data fra en .txt-fil
3. Legge dataene i en tabell for x-verdiene og en for y-verdiene
4. Lage en programmeringsfunksjon som sier at vi skal bruke en lineær modell i regresjonen
5. Utføre regresjonen, og vise på skjermen hva den matematiske modellen blir, stigningstallet (a) og skjæringspunktet med y-aksen (b)
6. Tegne grafen med resultatet – tilpass koden for plotting av graf
   - Plotte punktene med et scatterplot
   - Plotte regresjonsgrafen – hvordan få programmet til å regne ut y-verdiene for modellen?

Puslespilloppgave

Sorter kodelinjene etter numrene i tekstboksen over, om regresjon i Python. Da vil du få den riktige rekkefølgen i programmet ditt.

1.
```
co2data = loadtxt("co2utslipp.txt")
```

2.
```
from pylab import *
from scipy.optimize import curve_fit
```

3.
```
scatter(x_verdier, y_verdier)
plot(x_verdier, funksjon(x_verdier, a, b), color="red")
title("Samlet klimagassutslipp for Norge 1990 - 2018")
xlabel("År")
ylabel("CO2-utslipp (1000 tonn CO2-ekvivalenter)")
show()
```

4.
```
koeffisienter, kovarianser = curve_fit(funksjon, x_verdier, y_verdier)
a, b = koeffisienter
print(a,b)
```

5.
```
x_verdier = co2data[:,0]
y_verdier = co2data[:,1]
```

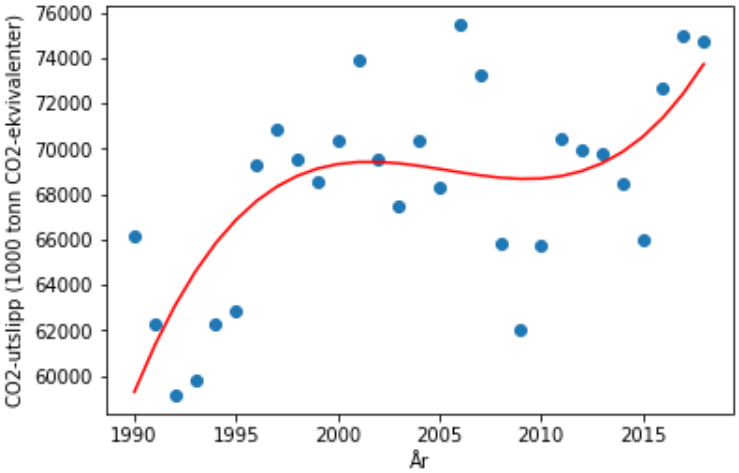
6.
```
def funksjon(x, a, b):
    return (a * x) + b
```

Ikke-lineær regresjon med Python

Forskjell på lineær og ikke-lineær regresjon i Python

Det er liten forskjell på hvordan programmene for lineær og ikke-lineær regresjon ser ut. Den viktigste forskjellen er å forandre på hvilken funksjonstype datasettet skal tilpasses. Det står en oversikt over noen funksjonstyper, og hvordan man skriver dem i python på de to neste sidene.

Samlet klimagassutslipp for Norge 1990 – 2018



Men hvordan skal vi vite hvilken funksjonstype som passer til datasettet vårt? Går det an å finne det ut på forhånd? Svaret på det spørsmålet avhenger av datasettet ditt. Du kan lage et scatterplot av datasettet ditt, og se om du kjenner igjen en av funksjonstypene. Som regel er det likevel vanskelig å gjette seg til en funsjonstype på forhånd, og vi kan prøve oss fram med forskjellige funksjonstyper.

Programmeringen er veldig lik for lineær og ikke-lineær regresjon. Vi må bare endre den lineære funksjonen til en ikke-lineær funksjon. Da vil modellen vår tilpasse seg datasettet vårt best mulig, og vi får vite koeffisientene som bestemmer funksjonen for modellen vår. I tillegg må vi endre til riktig antall koeffisienter i programmet vårt. Se eksempelet under på hva som må endres.

```
def funksjon(x, a, b):
    return (a * x) + b
```

→

```
def funksjon(x, a, b, c, d):
    return (a*x**3)+(b*x**2)+(c*x)+d
```

```
koeffisienter, kovarianser = curve_fit(funksjon, x_verdier, y_verdier)
a, b = koeffisienter
print(a,b)
```

↓

```
koeffisienter, kovarianser = curve_fit(funksjon, x_verdier, y_verdier)
a, b, c, d = koeffisienter
print(a,b,c,d)
```

```
scatter(x_verdier, y_verdier)
plot(x_verdier, funksjon(x_verdier, a, b), color="red")
title("Samlet klimagassutslipp for Norge 1990 - 2018")
xlabel("År")
ylabel("CO2-utslipp (1000 tonn CO2-ekvivalenter)")
show()
```

↓

```
scatter(x_verdier, y_verdier)
plot(x_verdier, funksjon(x_verdier, a, b, c, d), color="red")
title("Samlet klimagassutslipp for Norge 1990 - 2018")
xlabel("År")
ylabel("CO2-utslipp (1000 tonn CO2-ekvivalenter)")
show()
```

Vi bruker data fra statistisk sentralbyrå over klimagassutslipp i Norge for årene 1990 – 2018, og ønsker å lage en matematisk modell over utviklingen over tid. Vi prøver ut forskjellige funksjonstyper i regresjonsprogrammet, og ser hvilke modeller vi får.

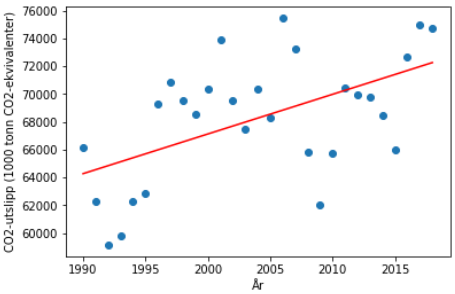
Vi bruker en lineær modell i programmet vårt

```
def funksjon(x, a, b):
    return (a * x) + b
```

Programmet gir oss den lineære funksjonen

$$f(x) = 286x - 504059$$

som den beste tilpasningen til datasettet vårt.



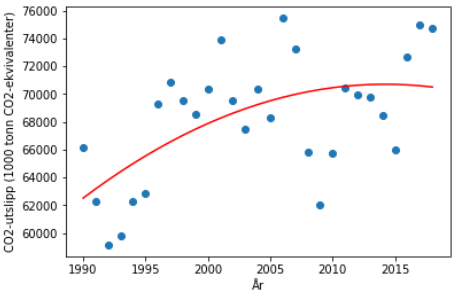
Vi bruker en andregradsmodell i programmet vårt

```
return (a*x**2)+(b*x)+c
```

Programmet gir oss andregradsfunksjonen

$$f(x) = -14x^2 + 56298 - 56627608$$

som den beste tilpasningen til datasettet vårt.



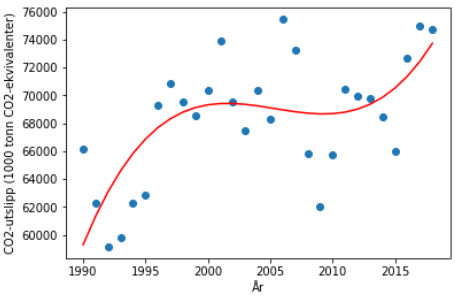
Vi bruker en tredjegradsmodell i programmet vårt

```
return (a*x**3)+(b*x**2)+(c*x)+d
```

Programmet gir oss tredjegradsfunksjonen

$$f(x) = 3x^3 - 19753x^2 + 39612328x - 26479502215$$

som den beste tilpasningen til datasettet vårt.



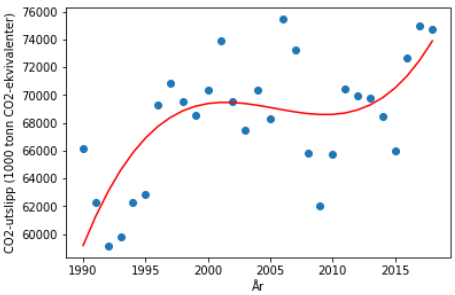
Vi bruker en fjerdegradsmodell i programmet vårt

```
return (a*x**4)+(b*x**3)+(c*x**2)+(d*x)+e
```

Programmet gir oss fjerdegradsfunksjonen

$$f(x) = 0{,}004x^4 - 30x^3 + 79206x^2 - 92029574x + 39188643912$$

som den beste tilpasningen til datasettet vårt.



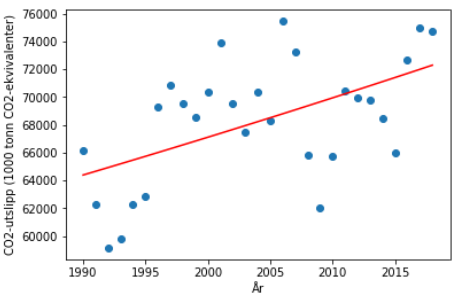
Vi bruker en eksponentiell modell i programmet vårt

```
return a * (b**x)
```

Programmet gir oss eksponentialfunksjonen

$$f(x) = 17 \cdot 1{,}004^x$$

som den beste tilpasningen til datasettet vårt.



Vi bruker en potensmodell i programmet vårt

```
return a*(x**b)
```

```
RuntimeError: Optimal parameters not
found: Number of calls to function has
reached maxfev = 600.
```

Når vi prøver å lage en potens-modell for CO_2-dataene våre, får vi det ikke til. For noen datasett vil noen funksjonstyper passe veldig dårlig. Da kan det hende at vi får en feilmelding i programmet, om at vi har brutt en tidsbegrensing. Da har det tatt programmet for lang tid å tilpasse modellen for oss, og vi får ikke noe resultat. Det kan være et tegn på at modellen (funksjonstypen) vi har valgt passer dårlig med datasettet vårt.

Vi ser på noen andre eksempler med et datasett for kumulativt antall Covid 19-smittede i Norge for uke 8 til 32 i 2020.

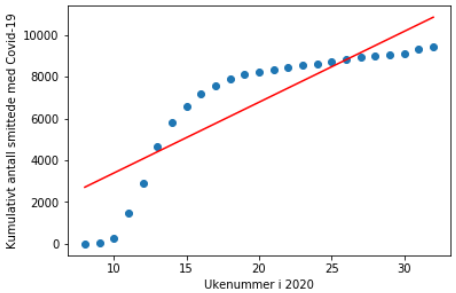
Vi bruker en potensmodell i programmet vårt

```
return a*(x**b)
```

Programmet gir oss potensfunksjonen

$$f(x) = 0{,}800 \cdot x^{1{,}00}$$

som den beste tilpasningen til datasettet vårt.



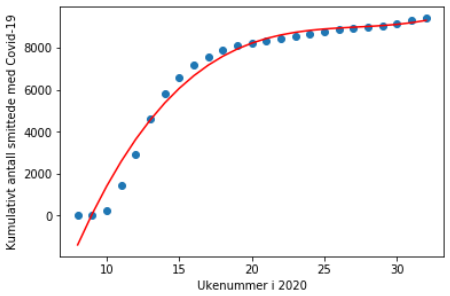
Vi bruker en tredjegradsmodell i programmet vårt

```
return (a * x**3) + (b * x**2) + (c * x) + d
```

Programmet gir oss tredjegradsfunksjonen

$$f(x) = 1{,}37x^3 - 112x^2 + 3080x - 19578$$

som den beste tilpasningen til datasettet vårt.



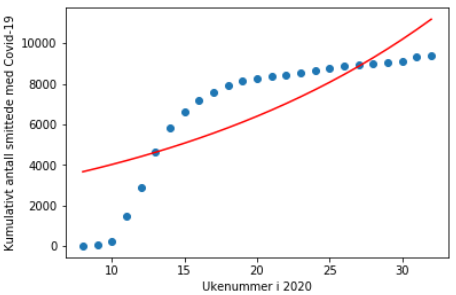
Vi bruker en eksponentialmodell i programmet vårt

```
return a * (b**x)
```

Programmet gir oss eksponentialfunksjonen

$$f(x) = 2521 \cdot 1{,}048^x$$

som den beste tilpasningen til datasettet vårt.



Nå vet vi hvordan vi kan bruke data for å tilpasse mange forskjellige funksjonstyper. Men hvordan skal vi vite hvilken modell som passer best med datasettet vårt? Må vi bare se hvilken funksjonstype som ser ut til å passe best med datasettet vårt, eller kan vi gjøre det på en mer objektiv måte? Bla om til neste side for å finne ut mer.

Korrelasjon i Python

Når vi skal beregne korrelasjonen, er det som regel Pearsons korrelasjonskoeffisient vi mener, og den kalles R. Det er også den vi skal bruke her. Vi er mest interessert i denne korrelasjonskoeffisienten opphøyd i andre, altså R^2, fordi den verdien sier hvor stor andel av variasjonen i målingene/datasettet vårt som henger direkte sammen. Verdien kan være mellom null og en, og desto høyere verdi, desto større del av variasjonen i y kan forklares av variasjonen i x. R^2 oppgis som et desimaltall, men multipliserer vi det med 100 %, får vi svaret i prosent.

Den enkleste formen for korrelasjon, er mellom x og y-verdiene i et datasett. Da finner vi hvor stor del av variasjonen i y som kan forklares av variasjonen i x. Dette gjøres ved at Python foretar en lineær regresjon, slik at R^2 viser hvor god den lineære sammenhengen mellom x og y-verdiene er. Da er R^2 et mål på hvor godt den lineære funksjonen passer med datasettet vårt.

Det går også an å gjøre tilsvarende for ikke-lineære regresjoner. Da får vi et objektivt mål på hvilken modell som passer best med vårt datasett. Dersom man vil finne R^2 for en regresjon der f(x) er en eksponentialfunksjon, begynner vi med å utføre selve regresjonen. Når den er ferdig, vet vi hva koeffisientene i eksponentialfunksjonen er, altså vet vi hva f(x) er. Da bruker vi den estimerte funksjonen til å putte inn x-verdiene fra datasettet vårt, for så å regne ut de tilsvarende f(x) verdiene. For å finne R^2 bruker vi y-verdiene fra datasettet vårt og f(x)-verdiene som vi regnet ut ved å bruke den eksponentielle modellen vi fant. Da gjelder fortsatt at den modellen som gir en verdi nærmest R^2 = 1,00 er den beste modellen.

Korrelasjon i Python består av fem deler:

1. Importere bibliotekene som trengs
2. Importere data fra en .txt-fil
3. Legge dataene i en tabell (array) for x-verdiene og en for y-verdiene
4. Beregne korrelasjonskoeffisienten
5. Vise korrelasjonskoeffisienten på skjermen

Datasettene vi bruker kan vi for eksempel hente fra statistisk sentralbyrå. Hvordan man gjør dette står i opplegg 26. Eller kan man bruke en micro:bit for å samle inn data. Dersom vi har mange målinger, er det fint å lagre det som en fil på micro:biten. Hvordan dette gjøres står i kapittel 12.

```
from pylab import *
from scipy.stats import pearsonr

data = loadtxt("filnavn.txt")

x = data[:,0]
y = data[:,1]

R = pearsonr(x, y)[0]

print(R)
print(R**2)
```

Importerer de nødvendige bibliotekene.

Legger inn data fra fil i en tabell.

Legger x-verdiene fra første kolonne inn i en tabell og y-verdiene fra den andre kolonnen i en tabell.

Regner ut korrelasjonskoeffisienten.

Skriver korrelasjonskoeffisienten og R^2 til skjermen. R^2 tilsvarer R^2-verdien i Geogebra.

Oppgaver

1. Finn et valgfritt datasett fra statistisk sentralbyrå, og bruk Python til å finne korrelasjonen for datasettet. Hva må du endre i eksempelet på denne siden?
2. Hvordan kan man beregne korrelasjonen mellom to forskjellige datasett? Hvilke kriterier må være oppfylt for at det skal kunne gå an?

Valg av regresjonsmodell i Python

Vi kan ikke *bare* se på R^2 når vi skal velge modell (velge mellom de ulike funksjonstypene). Vi bør nemlig ha en idé utfra teori, om hvilken type modell som burde fungere best. Det er for eksempel slik at en polynomfunksjon vil passe bedre og bedre, desto høyere grad den er. Det vil si at en fjerdegradsfunksjon vil alltid passe like godt eller bedre enn en tredjegradsfunksjon. Eller at en andregradsfunksjon vil alltid passe like godt eller bedre enn en førstegradsfunksjon, som er det samme som en lineær funksjon.

Regresjon med korrelasjon i Python består av sju deler:

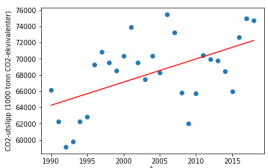
1. Importere bibliotekene som trengs
2. Importere data fra en .txt-fil
3. Legge dataene i en tabell for x-verdiene og en for y-verdiene
4. Lage en funksjon som sier hvilken modell som skal brukes i regresjonen
5. Utføre regresjonen, og vise på skjermen hva den matematiske modellen blir
6. Tegne grafen med resultatet
7. Regne ut R og R^2 og skrive dem ut til skjermen

```
from pylab import *
from scipy.optimize import curve_fit
from scipy.stats import pearsonr
```

```
R = pearsonr(y_verdier,funksjon(x_verdier,a,b,c))[0]
print(R)
print(R**2)
```

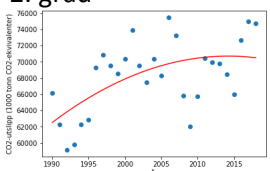
Nå kan vi sjekke hvilken matematisk modell som beskriver datasettene våre best. Vi sjekker hvilken modell som passer best for både CO_2-utslippene og kumulativt antall covid-19 smittede. Sjekk om du får samme verdier for de ulike modellene.

Lineær



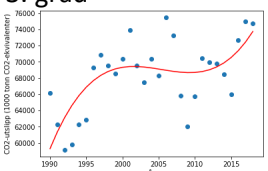
R = 0,5428
R^2 = 0,2947

2. grad



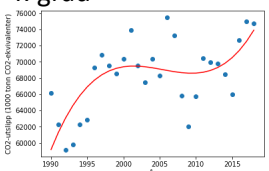
R = 0,5780
R^2 = 0,3340

3. grad



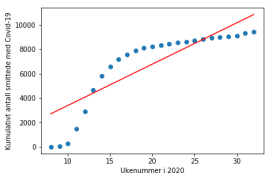
R = 0,6695
R^2 = 0,4483

4. grad



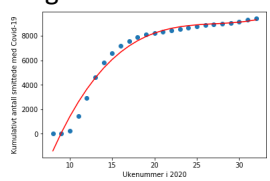
R = 0,6710
R^2 = 0,4502

Potens



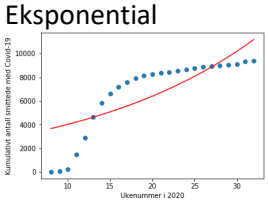
R = 0,8730
R^2 = 0,7620

3. grad



R = 0,9869
R^2 = 0,9740

Eksponential



R = 0,8004
R^2 = 0,6407

Diskusjonsoppgaver

1. Hvilken modell beskriver CO_2-utslippene best? Begrunn svaret.
2. Hvilken modell beskriver kumulativt antall covid-19 smittede best? Begrunn svaret.
3. Kan det potensielt finnes andre modeller som beskriver datasettene våre bedre?
4. Dersom du har laget et program som allerede utfører en regresjon, hva må du legge inn for at det også skal finne og skrive ut R og R^2?