Python_programmering_for_data_science

January 21, 2026

1 Python-programmering for datavitenskap

For å følge med: - JupyterHub ved UiO: https://www.uio.no/tjenester/it/utdanning/jupyter/ - GitHub-repositorium: https://github.uio.no/milenpa/IN-STK1050

1.1 Læringsmål

- Sette opp og administrere virtuelle miljøer
- Jobbe effektivt i Jupyter Notebooks
- Gjennomgå nyttige Python-funksjoner for datavitenskap
- Utføre numeriske operasjoner med NumPy-biblioteket
- Lage visualiseringer med Matplotlib og Seaborn

1.2 Husk

Kursinnholdet kan noen ganger avvike fra beste praksis innen programvareutvikling. Hovedmålet med kodeeksemplene er å illustrere statistiske konsepter og bygge intuisjon gjennom simuleringer, heller enn å følge strenge ingeniørstandarder.

1.3 Hva er en Jupyter Notebook?

- Interaktivt Python-miljø for å kjøre kode.
- Kombinerer kode, output og dokumentasjon på ett sted.
- Støtter rik output: tekst, bilder, plott og interaktive visualiseringer.
- Ideelt for live koding, testing og rask tilbakemelding.
- Tillater eksportering og deling av notebooks som HTML, PDF eller .ipynb-filer.

1.4 Hvordan kjøre Jupyter Notebooks?

- UiO JupyterHub: https://www.uio.no/tjenester/it/utdanning/jupyter/
- Hostede Jupyter-tjenester:
  - Binder: https://mybinder.org/
  - Google Colab: https://colab.research.google.com/
  - Kaggle: https://www.kaggle.com/code
- Lokalt oppsett: anbefalt med et virtuelt miljø, som uv, Conda eller venv.

1.5 Hva er uv?

- En pakke- og prosjektbehandler.
- Hjelper med å administrere avhengigheter og biblioteker i isolerte miljøer.

- Holder prosjekter organisert ved å unngå versjonskonflikter.
- Tillater forskjellige prosjekter å bruke forskjellige versjoner av Python eller biblioteker.

For å sette opp miljøet på din lokale maskin, følg disse stegene:

1. Klon GitHub-repositoriet for kurset:

```
git clone https://github.uio.no/milenpa/IN-STK1050.git
```

Alternativt, bruk lenken ovenfor for å klone repositoriet direkte via din IDE.

2. Installer uv og opprett miljøet med alle nødvendige pakker:

```
curl -LsSf https://astral.sh/uv/install.sh | sh
uv sync
```

For å kjøre Python-konsoll:

```
(in-stk1050) user@computer ~ % python
Python 3.12.11 (main, Sep 18 2025, 19:41:45) [Clang 20.1.4 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> np.__version__
'2.4.1'
```

Andre miljøbehandlere: - venv (https://docs.python.org/3/library/venv.html) - conda (https://www.anaconda.com/docs/getting-started/main) - mer om uv: https://docs.astral.sh/uv/

1.6 Nyttige Python-funksjoner for datavitenskap

I denne seksjonen vil vi dekke: - f-strenger, - gjennomgå lister og ordbøker, - liste- og ordbok-komprehensjon, - funksjoner, - typing.

1.7 F-strenger

F-strenger gir en enkel og effektiv måte å formatere strenger i Python. De tillater innbygging av uttrykk inne i strengliteraler, ved bruk av krøllparenteser {}.

[ ]:
```python
# Klassisk Hello World-eksempel:
print("Hello World!")
```

[ ]:
```python
# Bruk av f-strenger:
navn = "Maria"
poeng = 90
print(f"{navn} fikk {poeng} på testen.")  # Output: Maria fikk 90 på testen.
```

[ ]:
```python
# Komplekst uttrykk inne i f-streng:
print(f"Halvparten av {poeng} er {poeng / 2}.")
```

1.8 Jobbe med lister

Lister er ordnede, mutérbare samlinger som brukes til å lagre en sekvens av elementer. De tillater operasjoner som å legge til, aksessere og dele opp.

[ ]:
```python
# Opprette og legge til verdier i listen:
min_liste = [10, 20, 30, 40, 50]
print(min_liste)  # Output: [10, 20, 30, 40, 50]
min_liste.append(60)
print(min_liste)  # Output: [10, 20, 30, 40, 50, 60]

# Slette femte element
min_liste.pop(4)
print(min_liste)  # Output: [10, 20, 30, 40, 60]
# Slette et spesifikt element fra listen:
min_liste.remove(60)
print(min_liste) # Output: [10, 20, 30, 40]
```

[ ]:
```python
# Aksessere elementer og slicing:
print(min_liste)
print(min_liste[1])  # Output: 20 (andre element)
print(min_liste[-1])  # Output: 40 (siste element)
print(min_liste[:3])  # Output: [10, 20, 30] (tre første elementer)
print(min_liste[-2:])  # Output: [30, 40] (to siste elementer)
print(min_liste[1:3])  # Output: [20, 30] (andre og tredje element)
```

1.9 Jobbe med ordbøker

Ordbøker lagrer nøkkel-verdi-par. De er uordnede og mutérbare, noe som gjør dem ideelle for å kartlegge (bygge en oversikt over) relasjoner (som navn og poengsummer).

[ ]:
```python
# Opprette og aksessere en ordbok:
student_poeng = {"Alice": 85, "Bob": 92, "Charlie": 78}

print(student_poeng)  # Output: {'Alice': 85, 'Bob': 92, 'Charlie': 78}
print(student_poeng["Bob"])  # Output: 92 (Bobs poengsum)
print(student_poeng.get("Charlie"))  # Output: 78 (Charlies poengsum)
print(student_poeng.values()) # Output: dict_values([85, 92, 78])
print(student_poeng.keys()) # Output: dict_keys(['Alice', 'Bob', 'Charlie'])
```

1.9.1 Oppdatere og iterere over ordbøker

[ ]:
```python
# Legge til et nytt nøkkel-verdi-par:
student_poeng["David"] = 90
print(student_poeng)  # Output: {'Alice': 85, 'Bob': 92, 'Charlie': 78, 'David':
 ↪ 90}

# Oppdatere en eksisterende verdi:
student_poeng["Charlie"] = 80
print(student_poeng)  # Output: {'Alice': 85, 'Bob': 92, 'Charlie': 80, 'David':
 ↪ 90}
```

```
# Slette et nøkkel-verdi-par:
del student_poeng["Alice"]
print(student_poeng)  # Output: {'Bob': 92, 'Charlie': 80, 'David': 90}
```

[ ]:
```
# Iterere over en ordbok med for-løkke:
for nokkel, verdi in student_poeng.items():
    print(f"{nokkel} fikk {verdi}")
```

1.10 Listeforståelse

Listeforståelse tilbyr en konsis måte å lage lister i Python. De gir et elegant og effektivt alternativ til å bruke løkker for å generere lister. De brukes ofte til å anvende et uttrykk på hvert element i en itererbar (som en liste eller range) eller for å filtrere elementer basert på en betingelse.

[ ]:
```
min_liste = list(range(1, 6))
print(min_liste)

# finn kvadrater av alle elementer i listen:
kvadrater = []
for x in min_liste:
    kvadrater.append(x**2)

print(kvadrater)
```

[ ]:
```
# det samme med listeforståelse:
kvadrater = [x**2 for x in min_liste]
print(kvadrater)
```

1.10.1 Filtrere en liste

[ ]:
```
# filtrere en liste i en for-løkke:
partall = []
for x in min_liste:
    if x % 2 == 0:
        partall.append(x)

print(partall)

# listeforståelse for filtrering:
partall = [x for x in min_liste if x % 2 == 0]
print(partall)
```

1.10.2 Betinget prosessering av elementer i listen

```
[ ]: # Betinget prosessering av elementer i listen:
partallskvadrat = []
for x in min_liste:
    if x % 2 == 0:
        partallskvadrat.append(x**2)
print(partallskvadrat)

# Listeforståelse med betingelse:
partallskvadrat = [x**2 for x in range(1, 6) if x % 2 == 0]
print(partallskvadrat)  # Output: [4, 16]
```

1.10.3 Øvelse: studentkarakterer med lister

Gitt en liste med studentpoengsummer, filtrer først ut de under 60, og konverter deretter de gjenværende poengsummene til karakterer på følgende måte: for poengsummer over 90 er karakteren "A"; for poengsummer over 75 er karakteren "B", og for poengsummer over 60 er karakteren "C".

```
[ ]: poengsummer = [55, 81, 31, 78, 93, 61]

# TODO: fyll inn dette
scores_bestått = None
karakterer = None

print(scores_bestått)
print(karakterer)
```

1.11 Ordbokforståelse

- Ordbokforståelse gir en konsis måte å lage ordbøker i Python, akkurat som listeforståelse gjør for lister.
- De lar deg generere ordbøker fra en itererbar ved å spesifisere både nøkler og verdier i en enkelt, lesbar linje med kode.
- Du kan også anvende betingelser for å filtrere hvilke nøkkel-verdi-par som legges til i ordboken.

```
[ ]: # Lag en ordbok der nøkler er tall 1 til 5, og verdier er deres kvadrater:
kvadrat_ordbok = {}
for x in range(1, 6):
    kvadrat_ordbok[x] = x**2

print(kvadrat_ordbok)

# Ordbokforståelse:
kvadrat_ordbok = {x: x**2 for x in range(1, 6)}
print(kvadrat_ordbok)
```

1.11.1 Ordbokforståelse med betingelse

```
[ ]: # Samme oppgave med bare partall i det gitte området:
partallskvadrat_ordbok = {}
for x in range(1, 6):
    if x % 2 == 0:
        partallskvadrat_ordbok[x] = x**2

print(partallskvadrat_ordbok)

# Ordbokforståelse med betingelse:
partallskvadrat_ordbok = {x: x**2 for x in range(1, 6) if x % 2 == 0}
print(partallskvadrat_ordbok)
```

1.11.2 Bytte nøkler og verdier i en ordbok

```
[ ]: # Bytt nøkler og verdier i en eksisterende ordbok
original_ordbok = {'a': 1, 'b': 2, 'c': 3}
byttet_ordbok = {v: k for k, v in original_ordbok.items()}
print(f"Original dict: {original_ordbok}")
print(f"Swapped dict: {byttet_ordbok}")  # Output: {1: 'a', 2: 'b', 3: 'c'}
```

1.11.3 Øvelse: studentkarakterer med ordbøker

Gitt en ordbok med studentnavn og poengsummer, lag en ny ordbok kun med studenter som har bestått eksamen (poengsum > 60) og oversett hver students poengsum til en karakter (poengsum > 90 -> "A", poengsum > 75 -> "B", poengsum > 60 -> "C").

```
[ ]: poengsummer = {"Alice": 55, "Bob": 81, "Charlie": 31, "Diana": 78, "Eve": 93,␣
  ↪"Helen": 61}

# TODO: implementer dette
bestått_scores = None
karakterer = None

print(bestått_scores)
print(karakterer)
```

1.11.4 Øvelse: studentkarakterer med nestede ordbøker

Med utgangspunkt i poengsummene, lag en ordbok som har informasjon for hver student om de bestod/strøk og poengsummen de fikk (altså hver student er representert med en indre ordbok som sier om bestått (True/False) og poengsummen)

```
[ ]: # TODO: implementer dette
karakterer = None
```

```python
print(karakterer)
print(karakterer['Eve'])
```

1.12 Funksjoner

- Vi er vant til å tenke på funksjoner som gjenbrukbare kodeblokker som kan ta imot input (argumenter), prosessere det, og returnere output. De hjelper med å organisere og forenkle koden din.
- Python-funksjoner støtter standardargumenter, flere argumenter, flere returverdier og lambda-funksjoner (anonyme funksjoner).
- Funksjoner i matematikk (og statistikk) er en måte å oversette (mappe) en verdi til en annen (tenk på f(x)) - dette er lignende ordbøker i Python, bortsett fra at matematiske funksjoner kan oversette et uendelig antall verdier.
- Python-funksjoner (og funksjoner i programmering) oversetter også en verdi (sett av argumenter) til en annen verdi (returverdi). De kan til og med oversette et uendelig antall mulige inputverdier (ulikt en ordbok).

[ ]:
```python
# Programmeringsfunksjon (ikke en typisk matematisk funksjon):
def hils(navn):
    return f"Hallo, {navn}!"

print(hils("Alice"))  # Output: Hallo, Alice!

# en ordbok er en oversetting (mapping) og dermed en slags f(x)
kvadrater = {
    1: 1,
    2: 4,
    3: 9
}

# det samme uten begrensning på mulige inputer
def kvadrat(x):
    return x**2

print(kvadrat(2))
```

1.13 Typehint

- Python er et dynamisk typet språk: typer sjekkes ved kjøretid
- **Typehint** lar deg annotere forventede typer, selv om de annoterte typene ikke håndheves ved kjøretid
- Typehint forbedrer lesbarhet og dokumentasjon av koden, spesielt i større kodebaser
- Vi vil her bruke det (og oppfordrer deg til å bruke det) for å være presis om naturen til ulike statistiske konsepter

[ ]:
```python
# argumentet navn forventes å være en streng og funksjonen returnerer en streng
def hils(navn: str) -> str:
```

```
    return f"Hello, {navn}!"

returverdi = hils("Alice")
print(returverdi)
print(type(returverdi))
```

[ ]:
```
# Funksjoner med standardargumenter:
def hils(navn: str = "Alice"):
    return f"Hello, {navn}!"

print(hils())  # Output: Hello, Alice!
print(hils("Bob"))  # Output: Hello, Bob!
```

[ ]:
```
# Funksjoner med flere argumenter:
def hils(navn: str, melding: str):
    return f"{melding}, {navn}!"

print(hils("Alice", "Good Morning"))  # Output: Good Morning, Alice!
```

1.13.1 Øvelse: kvadratfunksjon med typehint

Skriv kvadratfunksjonen med typehint for å oversette fra float til float.

[ ]:
```

```

1.14 Pakking og utpakking med funksjoner

[ ]:
```
def hent_student_info():
    navn = "Alice"
    poeng = 85
    return navn, poeng

student_navn, student_poeng = hent_student_info()
print(f"{student_navn} fikk {student_poeng}")  # Output: Alice fikk 85
student_info = hent_student_info()
print(f"{student_info[0]} fikk {student_info[1]}")  # Output: Alice fikk 85
```

1.14.1 Øvelse: bytt to verdier

Skriv **én linje** for å bytte de to verdiene.

[ ]:
```
x = 1
y = 2

# TODO: bytt verdiene

print(x, y)
```

1.15 Lambda-funksjoner

Lambda-funksjoner er små anonyme funksjoner uten navn, ofte brukt for korte operasjoner.

[ ]:
```
# Lambda-funksjoner:
multipliser = lambda x, y: x * y
print(multipliser(5, 4))  # Output: 20

# Bruke lambda-funksjon med sortering
punkter = [(1, 2), (3, 1), (5, 4), (2, 3)]
sorterte_punkter = sorted(punkter, nokkel=lambda punkt: punkt[0])  # Sorter␣
 ↪etter x-koordinat
print(sorterte_punkter)
```

2 Biblioteker for datavitenskap

I denne seksjonen vil du lære om Python-biblioteker brukt i datavitenskap, som numpy, matplotlib og seaborn.

2.1 Importere biblioteker

- Å importere biblioteker lar deg bruke ekstra funksjonalitet fra eksterne moduler og pakker.
- Python har et rikt økosystem av biblioteker for ulike oppgaver som datamanipulering, visualisering og vitenskapelig databehandling.
- Du kan importere hele biblioteker, spesifikke funksjoner, eller gi aliaser for enklere bruk.

[ ]:
```
# Importere biblioteker:
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

2.2 Tilfeldige tall og operasjoner i Python (random-biblioteket)

- Random-modulen i Python gir funksjoner for å generere tilfeldige tall og velge tilfeldige elementer. Den brukes mye i simuleringer, randomisert testing og opprettelse av tilfeldige datasett for dataanalyse.
- Du kan generere tilfeldige tall, velge tilfeldige elementer fra lister og kontrollere reproduserbarhet med et frø.

[ ]:
```
# Du kan sette et frø for å generere de samme tilfeldige tallene:
random.seed(0)

# Generere et tilfeldig desimaltall:
print(random.random())  # Output: Tilfeldig desimaltall mellom 0 og 1
print(random.uniform(1, 10))  # Output: Tilfeldig desimaltall mellom 1 og 10
```

[ ]:
```
# Generere et tilfeldig heltall:
print(random.randint(1, 10))  # Output: Tilfeldig heltall mellom 1 og 10
```

2.2.1 Trekke fra en liste

[ ]:
```
# Generere et tilfeldig utvalg fra en liste:
min_liste = [10, 20, 30, 40, 50]

# Trekking uten tilbakelegging: allerede trukket element er ikke tilgjengelig␣
 ↪for å trekkes igjen i samme funksjonskall
print(random.sample(min_liste, 2))  # Output: Tilfeldig utvalg av 2 elementer␣
 ↪[uten tilbakelegging]

print(random.choice(min_liste))  # Output: Tilfeldig valg fra listen

# Trekking med tilbakelegging: velger fra hele listen for hvert av de k␣
 ↪elementene
print(random.choices(min_liste, k=2))  # Output: Tilfeldig valg av 2 elementer␣
 ↪[med tilbakelegging]
```

2.2.2 Øvelse: daglige temperaturer

Skriv en funksjon for å simulere daglig temperatur gitt basistemperaturen. Anta at alle temperaturer +/- 5 grader rundt basistemperaturen er like sannsynlige.

[ ]:
```

```

2.3 Introduksjon til NumPy

NumPy (Numerical Python) er et kraftig bibliotek for numeriske beregninger. Det gir støtte for flerdimensjonale arrayer og funksjoner for numeriske operasjoner på disse arrayene.

2.3.1 Opprette NumPy-arrayer

Numpy-arrayer er mer effektive enn Python-lister for numeriske operasjoner.

[ ]:
```
# Opprette Numpy-arrayer:
# 1D-array (vektor)
tabell_1d = np.array([1, 2, 3, 4, 5])
print("1D array:\n", tabell_1d)

# 2D-array (matrise)
tabell_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("\n2D array:\n", tabell_2d)

# 3D-array (tensor)
tabell_3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
```

```
print("\n3D array:\n", tabell_3d)
```

2.3.2 Egenskaper ved arrayer

[ ]:
```
# Sjekke typen til arrayene
print(type(tabell_1d))  # Output: <class 'numpy.ndarray'>
print(type(tabell_2d))  # Output: <class 'numpy.ndarray'>
```

[ ]:
```
# Form på arrayene
print("Shape of 1D array:", tabell_1d.shape)  # Output: (5,)
print("Shape of 2D array:", tabell_2d.shape)  # Output: (3, 3)
print("Shape of 3D array:", tabell_3d.shape)  # Output: (2, 2, 2)
```

[ ]:
```
# Antall dimensjoner (ndim) for arrayene
print("Dimensions of 1D array:", tabell_1d.ndim)  # Output: 1
print("Dimensions of 2D array:", tabell_2d.ndim)  # Output: 2
print("Dimensions of 3D array:", tabell_3d.ndim)  # Output: 3
```

2.3.3 Aksessere elementer i arrayer

[ ]:
```
# Aksessere elementer i en 1D-array
print(f"Element at index 2 in 1D array: {tabell_1d[2]}")  # Output: 3

# Aksessere elementer i en 2D-array
print(f"\nElement at row 1, col 2 in 2D array: {tabell_2d[1, 2]}")  # Output: 6

# Slicing av 1D-array
print(f"\nSlicing 1D array [1:4]: {tabell_1d[1:4]}")  # Output: [2, 3, 4]

# Slicing av 2D-array
print(f"\nSlicing 2D array [0:2, 1:3]:\n{tabell_2d[0:2, 1:3]}")
```

2.4 Vektorisering med NumPy

Vektorisering refererer til å utføre operasjoner på hele arrayer eller matriser (vektorer) uten eksplisitte løkker. Det lar deg utføre elementvise operasjoner i bulk, noe som gjør koden mer effektiv og raskere sammenlignet med tradisjonelle løkker.

[ ]:
```
# Vektorisering
# Uten vektorisering (bruker en Python-løkke)
tabell = np.array([1, 2, 3, 4, 5])
kvadrert = np.zeros_like(tabell)

for i in range(len(tabell)):
    kvadrert[i] = tabell[i] ** 2
```

```
print(kvadrert)  # Output: [ 1  4  9 16 25]

# Med vektorisering (bruker NumPys array-operasjoner)
kvadrert_vektorisert = tabell ** 2
print(kvadrert_vektorisert)  # Output: [ 1  4  9 16 25]
```

2.4.1 Grunnleggende operasjoner på NumPy-arrayer

[ ]:
```
# Grunnleggende operasjoner på Numpy-arrayer:

# Elementvis addisjon
tabell_sum = tabell_1d + 2  # Legger til 2 på hvert element
print(f"1D array after addition:\n{tabell_sum}")

# Merk forskjellen fra Python-lister:
liste1 = [1, 2, 3]
liste2 = [5]
liste_sum = liste1 + liste2
print(f"\n\"Sum\" of two lists: {liste_sum}")
```

[ ]:
```
# Elementvis multiplikasjon
tabell_produkt = tabell_2d * 2  # Multipliserer hvert element med 2
print(f"2D array after multiplication:\n{tabell_produkt}")
```

2.5 Nyttige NumPy-funksjoner

[ ]:
```
# Array med jevnt fordelte verdier (som en range)
tabell_rekke = np.arange(0, 10, 2)
print(f"Array with range 0 to 10 with step 2: {tabell_rekke}")

# Array med tilfeldige verdier
np.random.seed(0)
tabell_tilfeldig = np.random.random((2, 3))
print(f"Array of random values:\n{tabell_tilfeldig}")
```

[ ]:
```
# Numpy-funksjoner for numerisk analyse
tabell = np.array([1, 2, 3, 4, 5])

print(np.mean(tabell))  # Output: 3.0 (gjennomsnitt)
print(np.std(tabell))   # Output: 1.414... (standardavvik)
print(np.sum(tabell))   # Output: 15 (sum av alle elementer)
print(np.max(tabell))   # Output: 5 (maksverdi)
```

2.5.1 Øvelse: oppsummer temperaturer i en uke

Daglige temperaturer for en uke er gitt. Skriv funksjoner for å beregne følgende oppsummering: gjennomsnittstemperaturen, laveste og høyeste temperatur, og antall dager med temperaturer over og under det historiske daglige gjennomsnittet.

[ ]:
```
temperaturer = np.array([-1, 1, -4, -1, -2, 2, 1])
historisk_daglig_snitt = -1

# TODO: implementer dette
def oppsummer_uke(temperaturer: np.ndarray, historisk_daglig_snitt: int) ->␣
 ↪dict:
    return {}

oppsummer_uke(temperaturer, historisk_daglig_snitt)
```

2.6 Introduksjon til datavisualisering med Matplotlib og Seaborn

Matplotlib og Seaborn er biblioteker brukt for å lage datavisualiseringer i Python. De brukes mye i datavitenskap for å lage plott som spredningsplott, stolpediagrammer og linjediagrammer. Matplotlib gir lavnivå-kontroll over plottilpasning, mens Seaborn forenkler opprettelsen av vanlige plott ved å bygge på Matplotlib.

[ ]:
```
import seaborn as sns
```

[ ]:
```
data = {'Name': ['Alice', 'Tom', 'Bob', 'John', 'Kate', 'Selena', 'Adam',␣
 ↪'Nancy', 'Mary', 'Andrew', 'Jennifer', 'Robert', 'Matthew', 'Mark',␣
 ↪'Anthony', 'Steven', 'Audrey', 'Julia', 'Lisa', 'Richard'],
        'Age': [22, 21, 20, 33, 30, 27, 28, 21, 24, 35, 32, 30, 21, 31, 22, 29,␣
 ↪31, 26, 27, 30],
        'Score': [85, 90, 88, 70, 95, 90, 98, 60, 95, 60, 55, 60, 92, 88, 80,␣
 ↪85, 90, 92, 65, 78],
        'Hours_Studied': [4, 5, 4, 3, 6, 5, 7, 3, 7, 2, 3, 3, 7, 5, 4, 4, 6, 6,␣
 ↪3, 4]}
```

2.6.1 Spredningsplott

[ ]:
```
sns.scatterplot(x=data["Hours_Studied"], y=data["Score"])
plt.title("Studentpoengsummer etter navn (spredningsplott)")
plt.show()
```

2.6.2 Legge til farge og størrelse basert på en annen variabel

[ ]:
```
sns.scatterplot(
    x=data["Hours_Studied"],
    y=data["Score"],
    size=data["Age"],
```

```
    hue=data["Age"]
)

plt.title("Studentpoengsummer og studietimer")
plt.legend().remove()
plt.show()
```

2.6.3 Stolpediagram

[ ]:
```
sns.barplot(x=data["Name"], y=data["Score"])
plt.title("Studentpoengsummer etter navn (stolpediagram)")
plt.xticks(rotation=45)
plt.show()
```

2.6.4 Tilpasse stolpediagram

[ ]:
```
sns.barplot(
    x=data["Name"],
    y=data["Score"],
    hue=data["Hours_Studied"]
)

plt.title("Studentpoengsummer etter navn (sortert stolpediagram)")
plt.xticks(rotation=45)
plt.legend(
    bbox_to_anchor=(1.05, 1),
    loc='upper left',
)
plt.show()
```

2.6.5 Histogram

[ ]:
```
sns.histplot(x=data["Score"], bins=5)
plt.title("Fordeling av studentpoengsummer (histogram)")
plt.show()
```

2.6.6 Linjediagram

[ ]:
```
tids_data = {'Week': [1, 2, 3, 4, 5],
             'Score': [75, 80, 82, 85, 90]}

sns.lineplot(x=tids_data["Week"], y=tids_data["Score"], marker="o")
plt.title("Poengtrend over tid")
plt.show()
```

2.6.7 Tilpasse linjediagrammet

[ ]:
```
sns.lineplot(x=tids_data["Week"], y=tids_data["Score"], marker="o",␣
 ↪color="green")
plt.title("Studentpoengtrend over tid")
plt.xlabel("Ukenummer")
plt.ylabel("Poengverdi")
plt.grid(True)
# Lagre plottet til en jpg-fil
plt.savefig("student_poengsummer.jpg")
plt.show()
```

2.6.8 Varmekart

[ ]:
```
poengsummer = [
    [80, 85, 78],   # Alice
    [90, 88, 92],   # Bob
    [70, 75, 82]    # Charlie
]

sns.heatmap(
    poengsummer,
    annot=True,
    cmap="viridis",
    xticklabels=["Matte", "Naturfag", "Engelsk"],
    yticklabels=["Alice", "Bob", "Charlie"]
)

plt.title("Varmekart over studentpoengsummer")
plt.show()
```

2.6.9 Øvelse: utforsk temperatur- og regndata

Gitt temperatur- og regndata per dag i uken: - lag et linje-, spredningss- eller stolpediagram for å visualisere temperaturendringer over tid - hvilken visualisering er mest informativ for dette tilfellet? - legg til en "regnet"-nøkkel med True/False-verdier i ordboken og plott to temperaturhistogrammer: ett for regnfulle dager og ett for dager uten regn.

[ ]:
```
vaer_ordbok = {
    "day": [1, 2, 3, 4, 5, 6, 7],
    "temperature": [18, 19, 21, 23, 22, 20, 19],
    "rain_mm": [2, 0, 1, 0, 5, 3, 0]
}
```

[ ]:
```

```

3 Fullstendig eksempel

Denne oppgaven kombinerer lister, ordbøker, funksjoner, Numpy, Pandas og Seaborn/Matplotlib.

Problemstilling: - Generer 10 tilfeldige studentpoengsummer. - Lag en funksjon for å beregne gjennomsnittspoengsummen. - Lagre studentnavn og poengsummer i en Pandas DataFrame. - Filtrer studenter som skåret over gjennomsnittet. - Visualiser dataene med Seaborn/Matplotlib.

[ ]:
```
import numpy as np
from random import randint
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Generer tilfeldige studentpoengsummer
studenter = ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank', 'Grace',␣
 ↪'Hannah', 'Ivy', 'Jack']
poengsummer = [randint(50, 100) for _ in range(len(studenter))]
print(studenter)
print(poengsummer)


# 2. Funksjon for å beregne gjennomsnittspoengsum
def beregn_gjennomsnitt(poengsummer):
    return np.mean(poengsummer)


gj_snitt_poeng = beregn_gjennomsnitt(poengsummer)
print(f"Gjennomsnittspoengsum: {gj_snitt_poeng}")

# 3. Filtrer studenter med poengsummer over gjennomsnittet
over_gj_snitt_indekser = [i for i in range(len(studenter)) if poengsummer[i] >␣
 ↪gj_snitt_poeng]
over_gj_snitt = {'Name': [studenter[x] for x in over_gj_snitt_indekser],
                 'Score': [poengsummer[x] for x in over_gj_snitt_indekser]}
print(over_gj_snitt)

over_gj_snitt_indekser = np.array(poengsummer) > gj_snitt_poeng
print(over_gj_snitt_indekser)
over_gj_snitt = {'Name': np.array(studenter)[over_gj_snitt_indekser],
                 'Score': np.array(poengsummer)[over_gj_snitt_indekser]}
print(over_gj_snitt)

# 4. Visualiser resultater med Seaborn/Matplotlib
sns.barplot(x=over_gj_snitt['Name'], y=over_gj_snitt['Score'])
plt.title('Studenter som skåret over gjennomsnittet')
plt.xlabel('Name')
plt.ylabel('Score')
plt.xticks(rotation=45)
```

4 Øvelser

I denne seksjonen finner du øvelser for å praktisere konseptene dekket i denne notatboken. Du kan prøve å løse disse øvelsene på egen hånd eller diskutere dem med medstudenter. Løsningene er gitt nedenfor.

5 Aksjemarkedssimulering

Simuler tilfeldige prissvingninger for en aksje over tid.

Problemstilling:

- Bruk random.uniform(-0.05, 0.05) for å generere tilfeldige daglige prosentendringer (mellom -5% og +5%) i 100 dager.
- Anta en startaksjepris på $100.
- Beregn de daglige prisendringene basert på de tilfeldige prosentendringene.
- Visualiser aksjeprisene over tid med Seaborn/Matplotlib.
- Du kan også simulere flere aksjer og sammenligne deres tilfeldige prisendringer i én enkelt visualisering.

[ ]:

6 Simulering av tilfeldige daglige temperaturer

Simuler daglige temperaturendringer for et år med tilfeldighet.

Problemstilling: - Bruk random.uniform(-5, 5) for å simulere daglige temperaturavvik rundt en gjennomsnittsverdi (f.eks. 25°C). - Legg til sesongeffekter ved å variere gjennomsnittstemperaturen basert på måneden (f.eks. kaldere om vinteren, varmere om sommeren). - Lagre dataene i en ordbok med nøklene: "Dato" og "Temperatur". - Beregn den gjennomsnittlige månedlige temperaturen. - Bruk Seaborn/Matplotlib for å visualisere den daglige temperaturen som en tidsserie og et stolpediagram for månedlige gjennomsnitt.

[ ]:

7 Løsninger på øvelsene

7.0.1 Øvelse: studentkarakterer med lister

[ ]:
```
poengsummer = [55, 81, 31, 78, 93, 61]

scores_bestått = [el for el in poengsummer if el > 60]
karakterer = ["A" if el > 90 else "B" if el > 75 else "C" for el in␣
 ↪scores_bestått]

print(scores_bestått)
print(karakterer)
```

7.0.2 Øvelse: studentkarakterer med ordbøker

[ ]:
```
poengsummer = {"Alice": 55, "Bob": 81, "Charlie": 31, "Diana": 78, "Eve": 93,␣
 ↪"Helen": 61}

bestått_scores = {navn: poeng for navn, poeng in poengsummer.items() if poeng >␣
 ↪60}
karakterer = {navn: "A" if poeng > 90 else "B" if poeng > 75 else "C"
          for navn, poeng in bestått_scores.items()}

print(bestått_scores)
print(karakterer)
```

7.0.3 Øvelse: studentkarakterer med nestede ordbøker

[ ]:
```
poengsummer = {"Alice": 55, "Bob": 81, "Charlie": 31, "Diana": 78, "Eve": 93,␣
 ↪"Helen": 61}

karakterer = {navn: {"status": "bestått" if poeng > 60 else "stryk", "poeng":␣
 ↪poeng}
          for navn, poeng in poengsummer.items()}

print(karakterer)
print(karakterer['Eve'])
```

7.0.4 Øvelse: kvadratfunksjon med typehint

[ ]:
```
def kvadrat(x: float) -> float:
    return x**2

print(kvadrat(4.5))
```

7.0.5 Øvelse: bytt to verdier

[ ]:
```
x = 1
y = 2

x, y = y, x

print(x, y)
```

7.0.6 Øvelse: daglig temperatur

[ ]:
```
def simuler_daglig_temperatur(basis_temp: int = 20) -> int:
    return basis_temp + random.randint(-5, 5)

simuler_daglig_temperatur()
```

7.0.7 Øvelse: oppsummer temperaturer i en uke

[ ]:
```
temperaturer = np.array([-1, 1, -4, -1, -2, 2, 1])
historisk_daglig_snitt = -1

def oppsummer_uke(temperaturer: np.ndarray, historisk_daglig_snitt: int) ->␣
 ↪dict:
    return {
        "hoeyeste_temperatur": int(np.max(temperaturer)),
        "laveste_temperatur": int(np.min(temperaturer)),
        'gjennomsnittstemperatur': float(round(np.mean(temperaturer), 2)),
        'dager_over_snitt': int(np.sum(temperaturer > historisk_daglig_snitt)),
        'dager_under_snitt': int(np.sum(temperaturer < historisk_daglig_snitt))
    }

oppsummer_uke(temperaturer, historisk_daglig_snitt)
```

7.0.8 Øvelse: prosesser studenteksamendata

[ ]:
```
import numpy as np

student_eksamen_data = {'Name': ['Alice', 'Tom', 'Bob', 'John', 'Kate',␣
 ↪'Selena', 'Adam', 'Nancy', 'Mary', 'Andrew', 'Jennifer', 'Robert',␣
 ↪'Matthew', 'Mark', 'Anthony', 'Steven', 'Audrey', 'Julia', 'Lisa',␣
 ↪'Richard'],
                    'Age': [22, 21, 20, 33, 30, 27, 28, 21, 24, 35, 32, 30,␣
 ↪21, 31, 22, 29, 31, 26, 27, 30],
                    'Score': [85, 90, 88, 70, 95, 90, 98, 60, 95, 60, 55, 60,␣
 ↪92, 88, 80, 85, 90, 92, 65, 78],
                    'Hours_Studied': [4, 5, 4, 3, 6, 5, 7, 3, 7, 2, 3, 3, 7,␣
 ↪5, 4, 4, 6, 6, 3, 4]}

student_eksamen_data['Passed'] = [el > 60 for el in␣
 ↪student_eksamen_data['Score']]
print(student_eksamen_data)

average_hours_bestått = np.mean(np.
 ↪array(student_eksamen_data['Hours_Studied'])[student_eksamen_data['Passed']])
```

```
print(f"\nBestått eksamen: {sum(student_eksamen_data['Passed'])}/
 ↪{len(student_eksamen_data['Name'])}")
print(f"Gjennomsnittlig studietimer for studenter som bestod eksamen:␣
 ↪{average_hours_bestått}")
```

7.0.9 Øvelse: utforsk temperatur- og regndata

[ ]:
```
import seaborn as sns
from matplotlib import pyplot as plt

vaer_ordbok = {
    "day": [1, 2, 3, 4, 5, 6, 7],
    "temperature": [16, 20, 19, 23, 17, 19, 25],
    "rain_mm": [2, 0, 1, 0, 5, 3, 0]
}

sns.lineplot(x=vaer_ordbok["day"], y=vaer_ordbok["temperature"], marker="o")
plt.title("Temperatur i løpet av uken")
plt.xlabel("Day")
plt.ylabel("Temperature")
plt.show()

sns.scatterplot(x=vaer_ordbok["day"], y=vaer_ordbok["temperature"])
plt.title("Temperatur i løpet av uken")
plt.xlabel("Day")
plt.ylabel("Temperature")
plt.show()

sns.barplot(x=vaer_ordbok["day"], y=vaer_ordbok["temperature"])
plt.title("Temperatur i løpet av uken")
plt.xlabel("Day")
plt.ylabel("Temperature")
plt.show()
```

[ ]:
```
temp_regn = [vaer_ordbok["temperature"][i] for i in␣
 ↪range(len(vaer_ordbok['day'])) if vaer_ordbok['rain_mm'][i]>0]
sns.histplot(temp_regn, bins=5)
plt.title("Temperaturfordeling (regnfulle dager)")
plt.xlabel("Temperature")
plt.show()
```

7.0.10 Øvelse: Aksjemarkedssimulering

[ ]:
```python
def simuler_aksjepriser(aksje_navn, dager=100, start_pris=100):
    """
    Simulates random stock price changes for a given number of dager.
    :param aksje_navn: The navn of the stock.
    :param dager: The number of dager to simulate.
    :param start_pris: The starting price of the stock.
    :return: A dictionary with the simulated stock priser and dager.
    """
    tilfeldige_prosentendringer = [random.uniform(-0.05, 0.05) for _ in␣
 ↪range(dager)]
    priser = [start_pris]

    for change in tilfeldige_prosentendringer:
        priser.append(priser[-1] * (1 + change))

    dager_liste = list(range(dager + 1))

    return {
        "Stock": aksje_navn,
        "Day": dager_liste,
        "Price": priser
    }

aksjer = ["Stock_A", "Stock_B", "Stock_C"]
aksje_data = [simuler_aksjepriser(stock, dager=100) for stock in aksjer]

for stock in aksje_data:
    sns.lineplot(
        x=stock["Day"],
        y=stock["Price"],
        label=stock["Stock"]
    )

plt.title("Simulerte aksjepriser over 100 dager for flere aksjer")
plt.xlabel("Day")
plt.ylabel("Price")
plt.legend(title="Stock")
plt.grid(True)
plt.show()
```

7.0.11 Øvelse: Simulering av tilfeldige daglige temperaturer

[ ]:
```python
def simuler_daglige_temperaturer(dager=365, basis_temperatur=20) -> dict:
    """
    Simulates daily temperature changes for a period of time (supposedly a␣
 ↪year).
    :param dager: number of dager to simulate
    :param basis_temperatur: average temperature around which daily deviations␣
 ↪occur
    :return: a dict with daily temperaturer
    """
    temperaturer = []
    for day in range(dager):
        # Bestem måneden (grovt delt 365 dager i 12 måneder)
        maaned = (day // 30.5) + 1
        # Bruk sesongeffekter (kaldere om vinteren, varmere om sommeren)
        if maaned in [12, 1, 2]:  # Vintermåneder
            gj_snitt_temp = basis_temperatur - 10
        elif maaned in [6, 7, 8]:  # Sommermåneder
            gj_snitt_temp = basis_temperatur + 10
        else:  # Vår/høstmåneder
            gj_snitt_temp = basis_temperatur

        # Simuler daglig temperaturavvik
        daglig_temp = gj_snitt_temp + random.uniform(-5, 5)
        temperaturer.append(daglig_temp)

    dager_rekke = list(range(1, dager + 1))

    data = {
        'Day': np.array(dager_rekke),
        'Temperature': np.array(temperaturer)
    }

    return data

data = simuler_daglige_temperaturer()

data['Month'] = data['Day'] // 30.5 + 1  # oversett dager til måneder
maanedlig_gj_snitt_temp = {int(maaned): float(np.
 ↪mean(data['Temperature'][data['Month'] == maaned])) for maaned in␣
 ↪set(data['Month'])}
print(maanedlig_gj_snitt_temp)

sns.lineplot(x=data["Day"], y=data["Temperature"])
plt.title("Daglig temperatur over ett år")
plt.xlabel("Day")
```

```
plt.ylabel("Temperatur (°C)")
plt.grid(True)
plt.show()

sns.barplot(x=list(maanedlig_gj_snitt_temp.keys()),␣
 ↪y=list(maanedlig_gj_snitt_temp.values()))
plt.title("Gjennomsnittlig månedlig temperatur")
plt.xlabel("Month")
plt.ylabel("Average Temperatur (°C)")
plt.show()
```

7.1 Valgfritt: Introduksjon til Pandas

Pandas er et kraftig Python-bibliotek for datamanipulering og analyse. Det gir **DataFrame**, en 2D-merket datastruktur som gjør det enkelt å organisere, filtrere og manipulere strukturerte data, lignende et Excel-regneark eller SQL-tabell.

[ ]:
```
import pandas as pd

# Opprette en Pandas DataFrame
data = {
    "Name": ["Alice", "Bob", "Charlie", "David"],
    "Age": [25, 30, 35, 40],
    "Score": [85, 90, 88, 92]
}
df = pd.DataFrame(data)
print(df)
```

[ ]:
```
# Legge til en ny kolonne
df["Hours_Studied"] = [5, 10, 7, 8]
print(df)
```

[ ]:
```
# Aksessere en kolonne og hente verdier
timer_studert = df['Hours_Studied']
print(f"Column values:\n{timer_studert}\ntype: {type(timer_studert)}\n")

# For å få det som en numpy-array:
print(f"Getting it as a numpy array:\n{df['Hours_Studied'].values}\ntype:␣
 ↪{type(df['Hours_Studied'].values)}")
```

[ ]:
```
# Lese en CSV-fil
df = pd.read_csv("week_1_data.csv")
# Vise de første radene i DataFrame
print(df.head())
```

[ ]:
```
# Vise kolonnene i DataFrame
print(df.columns)
```

[ ]:
```
# Vise formen på DataFrame
print(df.shape)
```

[ ]:
```
# DataFrame-operasjoner
# Aksessere kolonner
print(df[["Name", "Age"]])
```

[ ]:
```
# Filtrere rader basert på en betingelse
print(df[df["Age"] > 30])
```

7.2 Lagre og eksportere dataframes

CSV er et veldig vanlig format for å lagre og dele tabelldata.

Avhengig av dine behov kan du også eksportere din DataFrame til Excel- eller JSON-formater, noe som gir deg fleksibilitet når du samhandler med forskjellige verktøy eller deler data.

[ ]:
```
# Lagre DataFrame til en CSV-fil
df.to_csv("week_1_data_tmp.csv", index=False)
```

[ ]:
```

```