



Slektstre med NetworkX

En komplett guide til genealogi og grafteori

En omfattende lærebok som kombinerer slektstrær (genealogi) og grafteori med praktiske eksempler og øvelser. Perfekt for både nybegynnere og erfarne brukere som vil forstå hvordan NetworkX kan brukes til å bygge og analysere familie-trær.



6 omfattende kapitler



Tospråklig



Praktiske øvelser



Visualiseringer



Slektsanalyse



Eksterne databaser

Av Arvid Lundervold

(etter en idé fra Helena Lundervold Pedersen)

12 Oktober 2025

Versjon 1.01



Innholdsfortegnelse

Introduksjon

- Slektstre med NetworkX - Introduksjon **1**
Hva er prosjektet, installasjon, hovedfunksjoner og rask start

Teori og grunnleggende konsepter

- Slektstrær og Grafer - En Introduksjon **15**
Kombinert introduksjon til genealogi og grafteori med praktiske øvelser

Praktisk bruk

- Oversikt og grunnleggende konsepter **45**
Detaljert oversikt over slektstre-prosjektet og dets komponenter
- Bygge slektstrær programmatisk **65**
Steg-for-steg guide til å bygge slektstrær med Python
- Import og eksport av data **85**
Håndtering av forskjellige dataformater (YAML, JSON, CSV, GEDCOM)
- Visualisering av slektstrær **105**
Alle visualiseringsalternativer og hvordan de brukes
- Integrasjon med eksterne databaser **125**
Kobling til FamilySearch, Digitalarkivet og andre genealogi-tjenester

Vedlegg

- Stikkordregister **145**
Alfabetisk oversikt over viktige begreper og deres definisjoner
- Bok-redigering og vedlikehold **155**
Instruksjoner for å redigere boken og legge til nye kapitler


Slektstre med NetworkX


Et Python-bibliotek for å bygge, administrere og visualisere familie-trær ved hjelp av NetworkX og Jupyter notebooks.




Podcast / Lydinnhold


Slektstre med Python og Grafteori - Slik Analyserer du Din Familie

 **Last ned podcast-filer:** - [MP3 \(universell kompatibilitet\)](#) - 20MB - [M4A \(høy kvalitet\)](#) - 40MB

 **For å spille av:** 1. Klikk på lenken ovenfor for å laste ned 2. Åpne med din foretrukne lydspiller 3. Eller høre direkte i nettleseren (støtter MP3/M4A)

 **Tips:** Du kan også høre på podcasten direkte i Jupyter Notebook ved å bruke:

```
import IPython.display as ipd
# Fra repository-rot:
ipd.Audio('podcast/Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.mp3',
          embed=False)
# Eller fra notebooks/-mappen:
# ipd.Audio('../podcast/Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.mp3',
#           embed=False)
```

 **Eller åpne [notebook 00_slektstraer_og_grafer.ipynb](#) som har en forbedret audio-spiller med:** - Automatisk path-deteksjon (fungerer både lokalt og i Google Colab) - HTML5 audio som fallback - **Enhanced cleanup** for å unngå store notebook-filer (fjerner embedded audio/bilder) - Bilingual instruksjoner og feedback

Enhanced Notebook Cleanup






Notebooks kan bli store på grunn av embedded audio/bilde-data. Vi har laget en forbedret cleanup-løsning:

Automatisk cleanup i notebook: - Kjør cleanup-cellen i notebooken før commit/push - Fjerner automatisk embedded audio og bilder - Reduserer filstørrelse med opptil 99% (fra 27MB til 43KB)

Manuell cleanup fra kommandolinje:

```
# Bruk det forbedrede cleanup-scriptet
python scripts/enhanced_notebook_cleanup.py
      notebooks/00_slektstraer_og_grafer.ipynb

# Eller bruk bash-scriptet
./scripts/cleanup_notebook.sh notebooks/00_slektstraer_og_grafer.ipynb
```

Hva cleanup gjør: -  Fjerner alle cell outputs -  Fjerner embedded base64 audio-data -  Fjerner embedded base64 bilder -  Beholder notebook-strukturen intakt -  Audio-spilleren fungerer fortsatt når notebooken kjøres på nytt

🇳🇴 Norsk / 🇬🇧 English

Norsk (Hovedspråk)

Slektstre-prosjektet lar deg bygge komplekse familie-trær med rike metadata, importere/eksportere data i flere formater, og visualisere slektskap på forskjellige måter. Prosjektet støtter både norsk og engelsk språk.

Hovedfunksjoner

- 🇮🇹 **Rike metadata:** Fødselsdatoer, steder, bilder, historier og notater
- 🔄 **Fleksibel datainput:** Manuell programmatisk opprettelse og fil-basert import
- 📈 **Avanserte visualiseringer:** Hierarkisk, vifte-diagram, interaktiv og timeglass-visning
- 🌐 **Tospråklig støtte:** Norsk (primær) og engelsk
- 📁 **Flere dataformater:** YAML, JSON, CSV og GEDCOM
- 🔍 **Slektsanalyse:** Beregning av slektskap, generasjonsnivåer og statistikk
- 🌐 **Eksterne databaser:** Integrasjon med FamilySearch, Digitalarkivet og Wikipedia API

Installasjon

```
# Opprett conda-miljø (anbefalt)
conda env create -f environment.yml
conda activate slektstre

# Eller installer pakker direkte
pip install -r requirements.txt

# For kun bok-generering
pip install -r requirements-book.txt
```

Rask start

```
# Importer modulene direkte fra src-mappen
import sys
sys.path.append('src')

from models import Person, Gender
from tree import Slektstre
from datetime import date

# Opprett personer
person = Person(
    fornavn="Arvid",
    etternavn="Lundervold",
    kjønn=Gender.MALE,
    fødselsdato=date(1985, 12, 10),
    fødested="Bergen"
)

# Opprett slektstre
slektstre = Slektstre()
slektstre.add_person(person)


# Visualiser
from visualization import plot_hierarchical_tree
import matplotlib.pyplot as plt
```

```
fig = plot_hierarchical_tree(slektstre)
plt.show()
```

Jupyter Notebooks







Prosjektet inkluderer seks omfattende notebooks:


0. **00_slektstraer_og_grafer.ipynb** - Introduksjon til slektstrær og grafteori
1. **01_introduksjon.ipynb** - Oversikt og grunnleggende konsepter
2. **02_bygg_tre_manuelt.ipynb** - Bygge slektstreet programmatisk
3. **03_importer_data.ipynb** - Import/eksport av data
4. **04_visualisering.ipynb** - Alle visualiseringsalternativer
5. **05_eksterne_databaser.ipynb** - Integrasjon med genealogi-databaser og API-er

 **Nytt: 00_slektstraer_og_grafer.ipynb** er en omfattende introduksjonsnotebook som kobler sammen genealogi og grafteori. Den dekker grunnleggende konsepter, praktiske øvelser, og viser hvordan NetworkX brukes til å bygge og analysere slektstrær. Perfekt for nybegynnere som vil forstå både slektstrær og den underliggende matematikken.

Åpne i Google Colab / Open in Google Colab

Alle notebooks kan kjøres direkte i Google Colab uten installasjon:

-  [Open in Colab](#) **00_slektstraer_og_grafer.ipynb** - Introduksjon til slektstrær og grafteori
-  [Open in Colab](#) **01_introduksjon.ipynb** - Oversikt og grunnleggende konsepter
-  [Open in Colab](#) **02_bygg_tre_manuelt.ipynb** - Bygge slektstreet programmatisk
-  [Open in Colab](#) **03_importer_data.ipynb** - Import/eksport av data
-  [Open in Colab](#) **04_visualisering.ipynb** - Alle visualiseringsalternativer
-  [Open in Colab](#) **05_eksterne_databaser.ipynb** - Integrasjon med genealogi-databaser og API-er

 **Tips:** Klikk på Colab-badgen for å åpne notebooken direkte i Google Colab. Alle avhengigheter installeres automatisk!

Dataformat

Slektstreet støtter flere formater:

YAML (anbefalt):

```
personer:
- id: "p1"
  fornavn: "Arvid"
  etternavn: "Lundervold"
  kjønn: "male"
  fødselsdato: "1985-12-10"
  fødested: "Bergen"
ekteskap:
- partner1_id: "p1"
  partner2_id: "p2"
  ekteskapsdato: "2010-06-18"
```

JSON, CSV og GEDCOM støttes også.

Visualiseringer

- **Hierarkisk slektstre:** Tradisjonell tre-struktur
- **Vifte-diagram:** Sirkulær visning av forfedre
- **Interaktiv visning:** Plotly-basert med hover-info

- **Timeglass-visning:** Fokuspersion i midten
- **Statistikk-diagrammer:** Kjønnfordeling, aldersfordeling, etc.

Eksterne databaser

- **FamilySearch API:** Verdens største genealogi-database (gratis)
- **Digitalarkivet:** Norske historiske kilder og arkiver (gratis)
- **Wikipedia API:** Biografisk informasjon om kjente personer (gratis)
- **Data-konvertering:** Automatisk konvertering fra eksterne formater
- **Eksport:** Til forskjellige formater for deling med andre

English

The Slektstre project allows you to build complex family trees with rich metadata, import/export data in multiple formats, and visualize relationships in various ways. The project supports both Norwegian and English languages.



Podcast / Audio Content

Slektstre med Python og Grafteori - Slik Analyserer du Din Familie



Note: Since this is a private repository, you need to download the podcast files locally to play them.



Download podcast files: - [MP3 \(universal compatibility\)](#) - 20MB - [M4A \(high quality\)](#) - 40MB



Note: Podcast files are not included in the Git repository due to size. You need to download them separately or generate them locally.



To play: 1. Download the file to your computer 2. Open with your preferred audio player 3. Or use an online audio player that supports local files



Tip: You can also listen to the podcast directly in Jupyter Notebook using:

```
import IPython.display as ipd
ipd.Audio('podcast/Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.mp3')
```

Key Features

- **Rich metadata:** Birth dates, places, photos, stories and notes
- **Flexible data input:** Manual programmatic creation and file-based import
- **Advanced visualizations:** Hierarchical, fan chart, interactive and hourglass views
- **Bilingual support:** Norwegian (primary) and English
- **Multiple data formats:** YAML, JSON, CSV and GEDCOM
- **Family analysis:** Relationship calculation, generation levels and statistics
- **External databases:** Integration with FamilySearch, Digitalarkivet and Wikipedia API

Installation

```
# Create conda environment (recommended)
```

```
conda env create -f environment.yml
```

```
conda activate slektstre
```

```
# Or install packages directly
```

```
pip install -r requirements.txt
```

```
# For book generation only
```

```
pip install -r requirements-book.txt
```

Quick Start

```
# Import modules directly from src folder
```

```
import sys
sys.path.append('src')
```

```
from models import Person, Gender
from tree import Slektstre
from datetime import date
```

```
# Create person
```

```
person = Person(
    fornavn="Arvid",
    etternavn="Lundervold",
    kjønn=Gender.MALE,
    fødselsdato=date(1985, 12, 10),
    fødested="Bergen"
)
```

```
# Create family tree
```

```
slektstre = Slektstre()
slektstre.add_person(person)
```

```
# Visualize
```


```
from visualization import plot_hierarchical_tree
import matplotlib.pyplot as plt
```

```
fig = plot_hierarchical_tree(slektstre)
plt.show()
```

Jupyter Notebooks

The project includes six comprehensive notebooks:

0. **00_slektstraer_og_grafer.ipynb** - Introduction to family trees and graph theory
1. **01_introduksjon.ipynb** - Overview and basic concepts
2. **02_bygg_tre_manuelt.ipynb** - Building family trees programmatically
3. **03_importer_data.ipynb** - Data import/export
4. **04_visualisering.ipynb** - All visualization options
5. **05_eksterne_databaser.ipynb** - Integration with genealogy databases and APIs

 **New: 00_slektstraer_og_grafer.ipynb** is a comprehensive introductory notebook that bridges genealogy and graph theory. It covers fundamental concepts, practical exercises, and shows how NetworkX is used to build and analyze family trees. Perfect for beginners who want to understand both family trees and the underlying mathematics.

Data Format

The family tree supports multiple formats:

YAML (recommended):

```
personer:
- id: "p1"
  fornavn: "Arvid"
  etternavn: "Lundervold"
  kjønn: "male"
```



```
fødselsdato: "1985-12-10"
fødested: "Bergen"
ekteskap:
  - partner1_id: "p1"
    partner2_id: "p2"
    ekteskapsdato: "2010-06-18"
```

JSON, CSV and GEDCOM are also supported.

Visualizations

- **Hierarchical family tree:** Traditional tree structure
- **Fan chart:** Circular ancestor view
- **Interactive view:** Plotly-based with hover info
- **Hourglass view:** Focus person in center
- **Statistics charts:** Gender distribution, age distribution, etc.

External Databases

- **FamilySearch API:** World's largest genealogy database (free)
- **Digitalarkivet:** Norwegian historical sources and archives (free)
- **Wikipedia API:** Biographical information about notable people (free)
- **Data conversion:** Automatic conversion from external formats
- **Export:** To various formats for sharing with others

Teknisk informasjon / Technical Information

Avhengigheter / Dependencies

Core slektstre packages: - Python 3.12+ - NetworkX 3.0+ - Matplotlib 3.7+ - Plotly 5.0+ - Pydantic 2.0+ - PyYAML 6.0+ - Pandas 2.0+ - Jupyter 1.0+

Book generation packages: - Jupyter Book 0.15+ - Sphinx 5.0+ - nbconvert[webpdf] 7.0+ - Playwright 1.55+ - PyPDF2 3.0+ - Pandoc 3.0+

Prosjektstruktur / Project Structure

```
slektstre/
├── src/                                # Kildekode / Source code
│   ├── __init__.py
│   ├── models.py                      # Pydantic modeller / Models
│   ├── tree.py                        # Slekststre-klasse / Main class
│   ├── family_io.py                  # Import/eksport / I/O functions
│   ├── visualization.py              # Visualisering / Visualization
│   └── localization.py              # Lokalisering / Localization
├── notebooks/                         # Jupyter notebooks
├── data/                              # Eksempeldata / Sample data
├── assets/                            # Bilder og media / Images and media
├── environment.yml                   # Conda miljø / Conda environment
├── requirements.txt                  # Python pakker / Python packages
└── README.md                        # Dokumentasjon / Documentation
```

Lisens / License

MIT License - se LICENSE fil for detaljer / see LICENSE file for details.

Håndtering av sensitive data / Handling Sensitive Data

For detaljert veiledning om å håndtere private familie-data, se [SENSITIVE_DATA_GUIDE.md](#).


For detailed guidance on managing private family data, see [SENSITIVE_DATA_GUIDE.md](#).


En privat repository template er tilgjengelig i `templates/private-repo/`.

A private repository template is available in `templates/private-repo/`.

Public Repository / Offentlig Repository

Dette er nå et **public repository** som alle kan se og bruke. Dette betyr:

 **Fordeler:** - Alle kan teste notebooks i Google Colab uten autentisering - Bedre muligheter for å dele kunnskap og lære sammen - Open source-utvikling og bidrag fra samfunnet - Enklere å finne og bruke for nye brukere

 **Viktig:** - Ingen sensitive familie-data er inkludert i dette repositoryet - Kun syntetiske eksempler og læringsressurser - For ekte familie-data, bruk private repository-malen

Bidrag / Contributing

Bidrag er velkommen! Se [DEVELOPER.md](#) for detaljerte instruksjoner.

Hvordan bidra: 1. Fork repositoryet 2. Lag en feature branch (`git checkout -b feature/ny-funksjon`) 3. Test endringene dine i Google Colab 4. Send en pull request

Contributions are welcome! See [DEVELOPER.md](#) for detailed instructions.

How to contribute: 1. Fork the repository 2. Create a feature branch (`git checkout -b feature/new-feature`) 3. Test your changes in Google Colab 4. Submit a pull request

Generer PDF-bok / Generate PDF Book

Du kan generere en komplett PDF-bok av hele prosjektet:

You can generate a complete PDF book of the entire project:

```
# Enkel metode / Simple method
make book
```

```
# Eller manuelt / Or manually
pip install -r requirements-book.txt
jupyter-book build . --builder pdfhtml
```

Boken vil inkludere README.md som introduksjon, fulgt av alle notebooks som separate kapitler.

The book will include README.md as introduction, followed by all notebooks as separate chapters.

Bok-redigering / Book Editing

For detaljerte instruksjoner om hvordan du redigerer boken, legger til kapitler, eller setter opp automatisk oppdatering, se [BOK-REDIGERING.md](#).

For detailed instructions on how to edit the book, add chapters, or set up automatic updates, see [BOK-REDIGERING.md](#).

Rask start for bok-redigering / Quick start for book editing

```
# Valider eksisterende bok / Validate existing book  
make validate
```

```
# Start automatisk overvåkning / Start automatic monitoring  
make watch
```

```
# Generer bok på nytt / Regenerate book  
make book
```

Utviklingsverktøy / Development Tools

For detaljerte instruksjoner om utvikling og repository-vedlikehold, se [DEVELOPER.md](#).

For detailed instructions on development and repository maintenance, see [DEVELOPER.md](#).

Hurtigreferanse / Quick reference:

```
# Fjern store filer / Remove large files  
pip install git-filter-repo  
git filter-repo --path large_file.ipynb --invert-paths --force  
  
# Rydd notebook outputs / Clean notebook outputs  
jupyter nbconvert --clear-output --inplace notebook.ipynb
```

Kontakt / Contact

Arvid Lundervold - [GitHub](#)

00_slektstraer_og_grafer

October 12, 2025

```
[17]: # =====
# GOOGLE COLAB SETUP / GOOGLE COLAB SETUP
# =====

# Sjekk om vi kjører i Google Colab
try:
    import google.colab
    IN_COLAB = True
    print(" Kjører i Google Colab - installerer avhengigheter...")
    print(" Running in Google Colab - installing dependencies...")

    # Installer nødvendige pakker
    import subprocess
    import sys
    try:
        subprocess.check_call([sys.executable, "-m", "pip", "install", "-q",
                                "networkx", "matplotlib", "plotly", "pydantic",
                                "pyyaml", "pandas", "ipywidgets", "pillow", ↵
↵"kaleido"])
        print(" Pakker installert")
    except Exception as e:
        print(f" Pip install feilet: {e}")

    # Fjern eksisterende slektstre-mappe hvis den finnes
    import shutil
    import os
    if os.path.exists('/content/slektstre'):
        shutil.rmtree('/content/slektstre')
        print(" Fjernet eksisterende slektstre-mappe")

    # Klon repository
    try:
        subprocess.check_call(['git', 'clone', 'https://github.com/arvidl/
↵slektstre.git'])
        print(" Repository klonet")
    except Exception as e:
        print(f" Git clone feilet: {e}")
```

```

# Legg til src-mappen til Python path og importer direkte
sys.path.insert(0, '/content/slektstre/src')
print(" Path lagt til")

# Importer slektstre-modulene direkte for å unngå navnekonflikt
import importlib.util
import types

# Først, fjern konfliktende moduler fra sys.modules
modules_to_remove = ['tree', 'models', 'localization']
for module_name in modules_to_remove:
    if module_name in sys.modules:
        del sys.modules[module_name]

# Last inn models.py først
try:
    spec = importlib.util.spec_from_file_location("slektstre_models", "/
↪content/slektstre/src/models.py")
    slektstre_models = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_models)

    # Opprett midlertidig models modul
    temp_models_module = types.ModuleType('models')
    temp_models_module.Person = slektstre_models.Person
    temp_models_module.Gender = slektstre_models.Gender
    temp_models_module.Ekteskap = slektstre_models.Ekteskap
    temp_models_module.FamilieData = slektstre_models.FamilieData
    sys.modules['models'] = temp_models_module

    print(" models.py lastet")
except Exception as e:
    print(f" models.py feilet: {e}")

# Last inn localization.py
try:
    spec = importlib.util.spec_from_file_location("slektstre_localization",
↪"/content/slektstre/src/localization.py")
    slektstre_localization = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_localization)

    # Opprett midlertidig localization modul
    temp_localization_module = types.ModuleType('localization')
    temp_localization_module.t = slektstre_localization.t
    sys.modules['localization'] = temp_localization_module

    print(" localization.py lastet")

```

```

except Exception as e:
    print(f" localization.py feilet: {e}")

# Last inn tree.py som slektstre_tree
try:
    spec = importlib.util.spec_from_file_location("slektstre_tree", "/
↪content/slektstre/src/tree.py")
    slektstre_tree = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_tree)

    # Opprett midlertidig tree modul
    temp_tree_module = types.ModuleType('tree')
    temp_tree_module.Slektstre = slektstre_tree.Slektstre
    sys.modules['tree'] = temp_tree_module

    print(" tree.py lastet")
except Exception as e:
    print(f" tree.py feilet: {e}")

# Last inn family_io.py
try:
    spec = importlib.util.spec_from_file_location("slektstre_io", "/content/
↪slektstre/src/family_io.py")
    slektstre_io = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_io)
    print(" family_io.py lastet")
except Exception as e:
    print(f" family_io.py feilet: {e}")

# Last inn visualization.py
try:
    spec = importlib.util.spec_from_file_location("slektstre_viz", "/
↪content/slektstre/src/visualization.py")
    slektstre_viz = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_viz)
    print(" visualization.py lastet")
except Exception as e:
    print(f" visualization.py feilet: {e}")

print(" Slektstre-moduler lastet inn i Colab")

except ImportError:
    IN_COLAB = False
    print(" Kjører lokalt / Running locally")
    import sys
    sys.path.append('../src')
except Exception as e:

```

```

print(f" Colab setup feilet: {e}")
IN_COLAB = False
print(" Fallback til lokal modus / Fallback to local mode")
import sys
sys.path.append('../src')

print(f" Miljø: {'Google Colab' if IN_COLAB else 'Lokal'}")
print(f" Environment: {'Google Colab' if IN_COLAB else 'Local'}")

```

Kjører lokalt / Running locally
Miljø: Lokal
Environment: Local

```

[18]: # Spill podcast direkte i notebook (uten å embedde data)
import IPython.display as ipd
import os
import gc

# Sjekk om IN_COLAB er definert, hvis ikke, sett til False
try:
    IN_COLAB
except NameError:
    IN_COLAB = False

# Prøv flere mulige paths for podcast-filen
possible_paths = []
if IN_COLAB:
    possible_paths = [
        "/content/slektstre/podcast/
↳Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.mp3",
        "/content/slektstre/podcast/
↳Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.m4a"
    ]
else:
    possible_paths = [
        "podcast/
↳Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.mp3",
        "podcast/
↳Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.m4a",
        "../podcast/
↳Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.mp3",
        "../podcast/
↳Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.m4a",
        "../podcast/
↳Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.mp3",
        "../podcast/
↳Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.m4a"
    ]

```

```

]

# Finn første eksisterende fil
podcast_path = None
for path in possible_paths:
    if os.path.exists(path):
        podcast_path = path
        break

if podcast_path:
    print(" Spiller podcast: Slektstre med Python og Grafteori")
    print(" Playing podcast: Family Trees with Python and Graph Theory")
    print(f" Fil: {podcast_path}")

    # Initialiser audio_widget variabel
    audio_widget = None

    # Prøv IPython.display.Audio først (uten embed=False for lokale filer)
    try:
        audio_widget = ipd.Audio(podcast_path)
        display(audio_widget)
        print(" Audio-spiller opprettet")
    except Exception as e:
        print(f" Audio-spiller feilet: {e}")
        print(" Prøv å åpne filen manuelt i nettleseren")

else:
    print(" Podcast-fil ikke funnet. Last ned fra GitHub repository.")
    print(" Podcast file not found. Download from GitHub repository.")
    print(f" Søkte etter følgende paths:")
    for path in possible_paths:
        print(f" - {path}")
    print(f" Current directory: {os.getcwd()}")

```

```

Spiller podcast: Slektstre med Python og Grafteori
Playing podcast: Family Trees with Python and Graph Theory
Fil:
../podcast/Slektstre_med_Python_og_Grafteori__Slik_Analyserer_du_Din_Famil.mp3
<IPython.lib.display.Audio object>

```

Audio-spiller opprettet

```

[19]: # AUTOMATISK CLEANUP - Kjør denne cellen før commit/push (kun lokalt)
# AUTOMATIC CLEANUP - Run this cell before commit/push (local only)

import os
import subprocess

```



```

import sys
import json

# Sjekk om vi kjører i Colab
try:
    IN_COLAB
except NameError:
    IN_COLAB = False

if IN_COLAB:
    print(" Cleanup-cellen hoppes over i Google Colab")
    print(" Cleanup cell skipped in Google Colab")
    print(" I Colab trenger du ikke å rydde opp før commit")
    print(" In Colab you don't need to cleanup before commit")
    print(" Colab-sessioner er midlertidige og påvirker ikke repository")
    print(" Colab sessions are temporary and don't affect repository")
else:
    def enhanced_cleanup_notebook():
        """Rydd opp i notebook-outputs med spesialhåndtering for embedded audio/
        ↪bilder"""

        # Prøv flere mulige paths for notebook
        possible_paths = [
            "notebooks/00_slektstraer_og_grafer.ipynb",
            "../notebooks/00_slektstraer_og_grafer.ipynb",
            "./notebooks/00_slektstre_og_grafer.ipynb",
            "00_slektstraer_og_grafer.ipynb"
        ]

        notebook_path = None
        for path in possible_paths:
            if os.path.exists(path):
                notebook_path = path
                break

        if not notebook_path:
            print(f" Notebook ikke funnet. Prøvde følgende paths:")
            for path in possible_paths:
                print(f"   - {path}")
            print(f" Current directory: {os.getcwd()}")
            return False

        print(" Starter automatisk cleanup av notebook...")
        print(" Starting automatic notebook cleanup...")
        print(f" Bruker notebook: {notebook_path}")
        print(f" Using notebook: {notebook_path}")

```

```

# Få størrelse før cleanup
size_before = os.path.getsize(notebook_path)
print(f" Størrelse før cleanup: {size_before / (1024*1024):.1f} MB")
print(f" Size before cleanup: {size_before / (1024*1024):.1f} MB")

try:
    # Last inn notebook og fjern outputs manuelt
    with open(notebook_path, 'r', encoding='utf-8') as f:
        notebook = json.load(f)

    cells_modified = 0
    audio_removed = 0
    images_removed = 0

    # Prosesser hver celle
    for cell in notebook.get('cells', []):
        if cell.get('cell_type') == 'code' and 'outputs' in cell:
            original_outputs = cell['outputs']
            cell['outputs'] = []

            # Sjekk om vi fjernet noe betydningsfullt
            if original_outputs:
                cells_modified += 1

            # Tell hva vi fjerner
            for output in original_outputs:
                if 'data' in output:
                    data = output['data']

                    # Sjekk for embedded audio
                    for key in data.keys():
                        if isinstance(data[key], list):
                            for item in data[key]:
                                if isinstance(item, str) and 'data:
↳audio' in item:
                                    audio_removed += 1
                                    break

                    # Sjekk for embedded bilder
                    for key in data.keys():
                        if isinstance(data[key], list):
                            for item in data[key]:
                                if isinstance(item, str) and 'data:
↳image' in item:
                                    images_removed += 1
                                    break

```

```

# Lagre den ryddede notebooken
with open(notebook_path, 'w', encoding='utf-8') as f:
    json.dump(notebook, f, ensure_ascii=False, separators=(',', '':
↪'))

# Få størrelse etter cleanup
size_after = os.path.getsize(notebook_path)
reduction = size_before - size_after

print(f" Enhanced cleanup fullført!")
print(f" Enhanced cleanup completed!")
print(f" Celler modifisert: {cells_modified}")
print(f" Cells modified: {cells_modified}")
print(f" Audio embeddings fjernet: {audio_removed}")
print(f" Audio embeddings removed: {audio_removed}")
print(f" Bilde embeddings fjernet: {images_removed}")
print(f" Image embeddings removed: {images_removed}")
print(f" Størrelse etter cleanup: {size_after / (1024*1024):.1f}MB")
↪

print(f" Size after cleanup: {size_after / (1024*1024):.1f} MB")
print(f" Reduksjon: {reduction / (1024*1024):.1f} MB")
print(f" Reduction: {reduction / (1024*1024):.1f} MB")

if reduction > 1024 * 1024: # Mer enn 1MB reduksjon
    print(" Betydelig størrelsesreduksjon oppnådd!")
    print(" Significant size reduction achieved!")
elif reduction > 0:
    print(" Noe cleanup utført")
    print(" Some cleanup performed")
else:
    print(" Ingen betydelig cleanup nødvendig")
    print(" No significant cleanup needed")

print(" Notebook er nå klar for commit/push!")
print(" Notebook is now ready for commit/push!")

return True

except Exception as e:
    print(f" Cleanup feilet: {e}")
    return False

# Kjør enhanced cleanup automatisk
cleanup_success = enhanced_cleanup_notebook()

if cleanup_success:
    print(" Tips:")

```

```

print("- Notebook er nå ryddet og klar for commit")
print("- Du kan trygt pushe til GitHub")
print("- Audio-spillere vil fortsatt fungere når notebooken kjøres på
↳nytt")
print("- Enhanced cleanup fjerner embedded audio/bilder automatisk")
else:
print("\n Cleanup feilet. Kjør manuelt:")
print(" Cleanup failed. Run manually:")
print("python scripts/enhanced_notebook_cleanup.py notebooks/
↳00_slektstraer_og_grafer.ipynb")

```

Starter automatisk cleanup av notebook...

Starting automatic notebook cleanup...

Bruker notebook: ../notebooks/00_slektstraer_og_grafer.ipynb

Using notebook: ../notebooks/00_slektstraer_og_grafer.ipynb

Størrelse før cleanup: 27.7 MB

Size before cleanup: 27.7 MB

Enhanced cleanup fullført!

Enhanced cleanup completed!

Celler modifisert: 16

Cells modified: 16

Audio embeddings fjernet: 1

Audio embeddings removed: 1

Bilde embeddings fjernet: 0

Image embeddings removed: 0

Størrelse etter cleanup: 0.1 MB

Size after cleanup: 0.1 MB

Reduksjon: 27.6 MB

Reduction: 27.6 MB

Betydelig størrelsesreduksjon oppnådd!

Significant size reduction achieved!

Notebook er nå klar for commit/push!

Notebook is now ready for commit/push!

Tips:

- Notebook er nå ryddet og klar for commit
- Du kan trygt pushe til GitHub
- Audio-spillere vil fortsatt fungere når notebooken kjøres på nytt
- Enhanced cleanup fjerner embedded audio/bilder automatisk

1 Slektstrær og Grafer - En Introduksjon

2 Family Trees and Graphs - An Introduction

Norsk | English

2.1 Velkommen til slektstre-prosjektet! / Welcome to the Family Tree Project!

Denne notebooken gir deg en grundig introduksjon til både slektstrær (genealogi) og grafteori, og hvordan disse to fagområdene kobles sammen i dette prosjektet.

This notebook provides you with a comprehensive introduction to both family trees (genealogy) and graph theory, and how these two fields are connected in this project.

2.1.1 Hva vil du lære? / What will you learn?

Norsk: - Hva slektstrær er og hvorfor de er viktige - Grunnleggende grafteori og hvordan den gjelder for slektstrær - Praktiske øvelser med både refleksjon og programmering - Hvordan NetworkX brukes til å bygge og analysere slektstrær

English: - What family trees are and why they matter - Basic graph theory and how it applies to family trees - Practical exercises with both reflection and programming - How NetworkX is used to build and analyze family trees

2.1.2 Forutsetninger / Prerequisites

Norsk: - Grunnleggende Python-kunnskap (anbefalt) - Ingen forkunnskap om grafteori nødvendig - Åpenhet for å lære nye konsepter

English: - Basic Python knowledge (recommended) - No prior graph theory knowledge required - Openness to learning new concepts

2.2 Del 1: Hva er slektstrær? / Part 1: What are Family Trees?

2.2.1 Slektrær - En historisk oversikt

Et **slektstre** (også kalt **stamtre** eller **genealogi**) er en visuell representasjon av slektskap og familierelasjoner. Slektrær har eksistert i tusenvis av år og har spilt en viktig rolle i menneskelig kultur og historie.

Hvorfor er slektrær viktige?

1. **Personlig identitet:** Hjelper oss å forstå vår egen bakgrunn
2. **Historisk bevaring:** Bevarer familiens historie for fremtidige generasjoner
3. **Medisinsk informasjon:** Kan gi viktig informasjon om arvelige sykdommer
4. **Kulturell betydning:** Kobler oss til vår kulturelle og etniske bakgrunn
5. **Historisk forskning:** Hjelper historikere å forstå befolkningsbevegelser og sosiale strukturer

Typer slektrær 1. **Stamtavle (Pedigree Chart)** - Viser forfedre til en bestemt person - Tradisjonell tre-struktur - Brukes ofte i medisinsk kontekst

2. **Etterkommertre (Descendant Tree)** - Viser alle etterkommere fra en bestemt forfader - Nyttig for å se familiens utbredelse

3. **Timeglass-visning (Hourglass Chart)** - Viser både forfedre og etterkommere - Fokuspersone i midten - Gir komplett oversikt

4. Vifte-diagram (Fan Chart) - Sirkulær visning av forfedre - Estetisk tiltalende - Populær i moderne genealogi-programmer

2.2.2 Family Trees - A Historical Overview

A **family tree** (also called **pedigree** or **genealogy**) is a visual representation of kinship and family relationships. Family trees have existed for thousands of years and have played an important role in human culture and history.

Why are family trees important?

1. **Personal identity:** Helps us understand our own background
2. **Historical preservation:** Preserves family history for future generations
3. **Medical information:** Can provide important information about hereditary diseases
4. **Cultural significance:** Connects us to our cultural and ethnic background
5. **Historical research:** Helps historians understand population movements and social structures

Types of family trees 1. **Pedigree Chart** - Shows ancestors of a specific person - Traditional tree structure - Often used in medical context

2. **Descendant Tree** - Shows all descendants from a specific ancestor - Useful for seeing family spread

3. **Hourglass Chart** - Shows both ancestors and descendants - Focus person in the center - Provides complete overview

4. **Fan Chart** - Circular view of ancestors - Aesthetically pleasing - Popular in modern genealogy software

2.2.3 Refleksjonsoppgaver / Reflection Questions

Norsk: 1. Hvilke typer slektskap kjenner du til i din egen familie? 2. Hvor langt tilbake i tid kan du spore din familie? 3. Hvilke historier eller tradisjoner har blitt videreført i din familie? 4. Hvorfor tror du slektstrær er viktige for mennesker?

English: 1. What types of relationships do you know about in your own family? 2. How far back in time can you trace your family? 3. What stories or traditions have been passed down in your family? 4. Why do you think family trees are important to people?

2.3 Del 2: Grunnleggende grafteori / Part 2: Basic Graph Theory

2.3.1 Hva er en graf? / What is a Graph?

En **graf** i matematikk og informatikk er en samling av **noder** (også kalt **hjørner** eller **vertices**) som er koblet sammen med **kanter** (edges). Dette er et kraftig konsept som brukes i mange områder, inkludert slektstrær!

A graph** in mathematics and computer science is a collection of **nodes** (also called **vertices**) that are connected by **edges**. This is a powerful concept used in many areas, including family trees!**

Grunnleggende begreper / Basic Concepts **Norsk:** - **Node/Hjørne:** Et punkt i grafen (f.eks. en person) - **Kant:** En linje som kobler to noder (f.eks. en slektsrelasjon) - **Retted graf:** Kanter har retning ($A \rightarrow B$) - **Ikke-retted graf:** Kanter har ingen retning ($A \rightarrow B$) - **Tre:** En spesiell type graf uten sykler

English: - **Node/Vertex:** A point in the graph (e.g., a person) - **Edge:** A line connecting two nodes (e.g., a family relationship) - **Directed graph:** Edges have direction ($A \rightarrow B$) - **Undirected graph:** Edges have no direction ($A \rightarrow B$) - **Tree:** A special type of graph without cycles

2.3.2 Eksempler med NetworkX / Examples with NetworkX

La oss se på noen enkle eksempler:

```
[20]: # Importer nødvendige biblioteker
import networkx as nx
import matplotlib.pyplot as plt
from datetime import date

# Importer slektstre-moduler (fungerer både lokalt og i Colab)
if IN_COLAB:
    # Bruk de modulene vi lastet inn i Colab-setup
    Person = slektstre_models.Person
    Gender = slektstre_models.Gender
    Slekststre = slektstre_tree.Slekststre
    load_from_yaml = slektstre_io.load_from_yaml
else:
    # Lokale imports
    from models import Person, Gender
    from tree import Slekststre
    from family_io import load_from_yaml

print(" Biblioteker importert!")
print(" Libraries imported!")
```

Biblioteker importert!

Libraries imported!

2.4 Eksempel 4: Bygg et enkelt slektstre som graf

2.5 Example 4: Build a simple family tree as graph

I dette eksemplet skal vi bygge et enkelt slektstre ved å bruke NetworkX-grafbiblioteket. Vi vil opprette noder for familiemedlemmer og knytte dem sammen med kanter som representerer familierelasjoner.

Læringsmål / Learning objectives: - Forstå hvordan slektstre kan representeres som grafer -

Lære å bruke NetworkX for å bygge grafer - Se forskjellen mellom rettede og ikke-rettede kanter - Forstå hvordan familierelasjoner kan modelleres

Hva vi skal gjøre / What we will do: 1. Opprette en rettet graf med NetworkX 2. Legge til noder for familiemedlemmer 3. Koble sammen familierelasjoner med kanter 4. Visualisere slektstreet

2.6 Nabomatrise (Adjacency Matrix)

En nabomatrise er en matematisk representasjon av en graf som viser hvilke noder som er koblet sammen. I en nabomatrise A er $A[i,j] = 1$ hvis det finnes en kant fra node i til node j , og 0 hvis ikke.

Viktige egenskaper / Important properties: - **Symmetrisk for ikke-rettede grafer:** $A[i,j] = A[j,i]$ - **Ikke-symmetrisk for rettede grafer:** $A[i,j]$ kan være forskjellig fra $A[j,i]$ - **Blandet graf:** Kombinasjon av rettede og ikke-rettede kanter - **Diagonal:** $A[i,i] = 0$ (ingen node kobler til seg selv)

I slektstre / In family trees: - Rettede kanter: Forelder \rightarrow Barn (asymmetrisk) - Ikke-rettede kanter: Ekteskap (symmetrisk) - Resultatet blir en ikke-symmetrisk nabomatrise

2.7 Node-grader i blandede grafer

I blandede grafer (som slektstre) har vi både rettede og ikke-rettede kanter. Dette gir oss tre typer grader for hver node:

Matematiske definisjoner / Mathematical definitions: - **Grad (Degree):** Totalt antall kanter som er koblet til noden - **Inn-grad (In-degree):** Antall rettede kanter som peker INN til noden - **Ut-grad (Out-degree):** Antall rettede kanter som peker UT fra noden

I slektstre / In family trees: - **Inn-grad:** Antall foreldre (vanligvis 0 eller 2) - **Ut-grad:** Antall barn - **Grad:** Antall foreldre + antall barn + antall ektefeller

Eksempel / Example: - En person med 2 foreldre, 3 barn og 1 ektefelle har: - Inn-grad: 2 - Ut-grad: 3 - Grad: $2 + 3 + 1 = 6$

```
[21]: # Eksempel 1: Enkel urettet graf / Simple undirected graph
print("  Eksempel 1: Enkel urettet graf")
print("  Example 1: Simple undirected graph")

# Opprett en enkel graf
G = nx.Graph()

# Legg til noder (5 totalt)
G.add_node("A", name="Alice")
G.add_node("B", name="Bob")
G.add_node("C", name="Charlie")
G.add_node("D", name="David")
G.add_node("E", name="Emma")
```



```

# Legg til kanter
G.add_edge("A", "B")
G.add_edge("B", "C")
G.add_edge("C", "D")
G.add_edge("D", "E")
G.add_edge("E", "A") # Lager en sykel for å vise forskjell fra tre

# Visualiser med fiksert layout
plt.figure(figsize=(10, 8))

# Definer posisjoner manuelt for konsistent layout
pos = {
    "A": (0, 1),      # Alice øverst i midten
    "B": (-1, 0),     # Bob til venstre
    "C": (-0.5, -1),  # Charlie nederst til venstre
    "D": (0.5, -1),   # David nederst til høyre
    "E": (1, 0)       # Emma til høyre
}

nx.draw(G, pos, with_labels=True, node_color='lightblue',
        node_size=1000, font_size=16, font_weight='bold')
plt.title("Enkel urettet graf / Simple undirected graph")
plt.show()

print(f"Antall noder: {G.number_of_nodes()}")
print(f"Antall kanter: {G.number_of_edges()}")
print(f"Number of nodes: {G.number_of_nodes()}")
print(f"Number of edges: {G.number_of_edges()}")

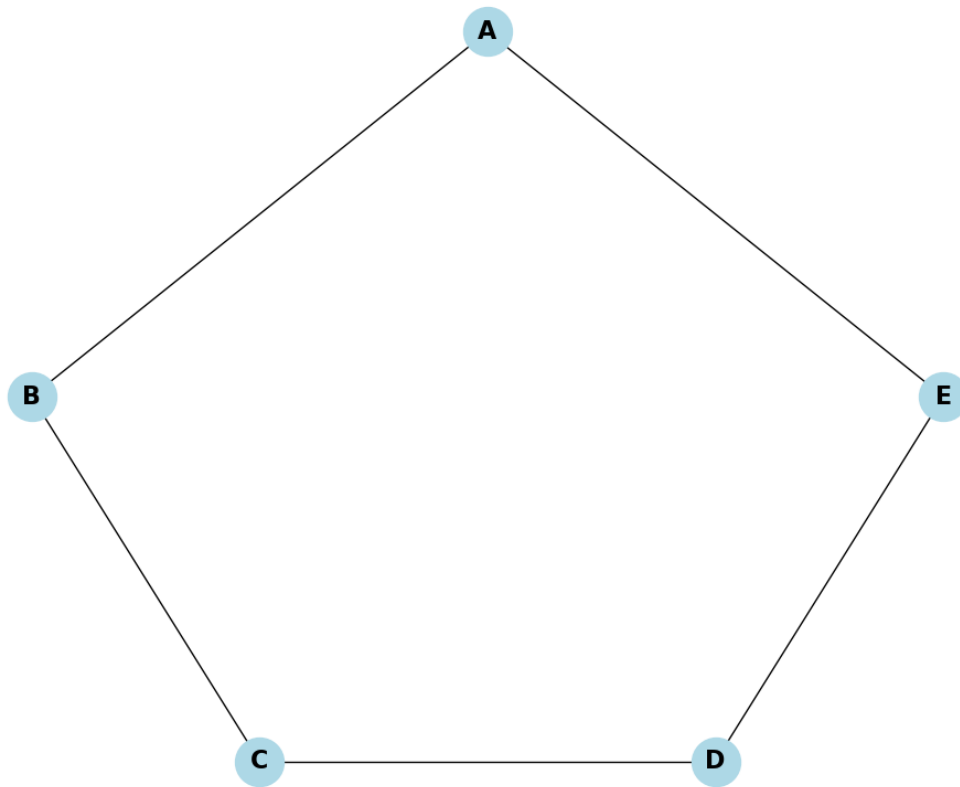
# Vis alle noder og deres navn
print(f"\nNoder i grafen:")
print(f"Nodes in the graph:")
for node in G.nodes():
    print(f"    {node}: {G.nodes[node]['name']}")

```

Eksempel 1: Enkel urettet graf

Example 1: Simple undirected graph

Enkel urettet graf / Simple undirected graph



Antall noder: 5
Antall kanter: 5
Number of nodes: 5
Number of edges: 5

Noder i grafen:
Nodes in the graph:
A: Alice
B: Bob
C: Charlie
D: David
E: Emma

2.8 Avstands-matrise (Distance Matrix)

En avstands-matrise viser den korteste stien mellom alle par av noder i en graf. Avstanden er antall kanter man må gå for å komme fra en node til en annen.

Viktige egenskaper / Important properties: - **Avstand:** Antall kanter i korteste sti mellom to noder - **Diagonal:** Avstand fra node til seg selv er alltid 0 - **Symmetrisk:** Avstanden fra A til

B er samme som fra B til A - **Uendelig**: Hvis det ikke finnes sti mellom to noder

I slektstre / In family trees: - **Biologisk beslektet**: Forelder-barn, søsken (deler blod) - **Ikke-biologisk beslektet**: Ektefeller, svigerfamilie - **Avstand 1**: Direkte relasjon (forelder-barn, ektefelle) - **Avstand 2**: Besteforeldre-barn, svigerforeldre

Eksempel / Example: - Far til barn: avstand 1 (direkte forelder-barn) - Bestefar til barnebarn: avstand 2 (via forelder)

```
[22]: # Eksempel 2: Rettet graf / Directed graph
print("  Eksempel 2: Rettet graf")
print("  Example 2: Directed graph")

# Opprett en rettet graf
D = nx.DiGraph()

# Legg til noder
D.add_node("Parent", name="Forelder")
D.add_node("Child1", name="Barn1")
D.add_node("Child2", name="Barn2")

# Legg til rettede kanter (forelder → barn)
D.add_edge("Parent", "Child1")
D.add_edge("Parent", "Child2")

# Visualiser med manuell posisjonering
plt.figure(figsize=(8, 6))

# Definer posisjoner manuelt: Forelder øverst, Barn1 og Barn2 på samme linje
# ↪ nedenfor
pos = {
    "Parent": (0, 1),      # Forelder øverst i midten
    "Child1": (-0.5, 0),   # Barn1 til venstre nedenfor
    "Child2": (0.5, 0)     # Barn2 til høyre nedenfor
}

# Tegn noder
nx.draw_networkx_nodes(D, pos, node_color='lightgreen',
                       node_size=1000, alpha=0.7)

# Tegn kanter
nx.draw_networkx_edges(D, pos, arrows=True, arrowsize=20,
                       edge_color='black', width=2)

# Legg til etiketter med norske navn
labels = {node: D.nodes[node]['name'] for node in D.nodes()}
nx.draw_networkx_labels(D, pos, labels, font_size=16, font_weight='bold')
```

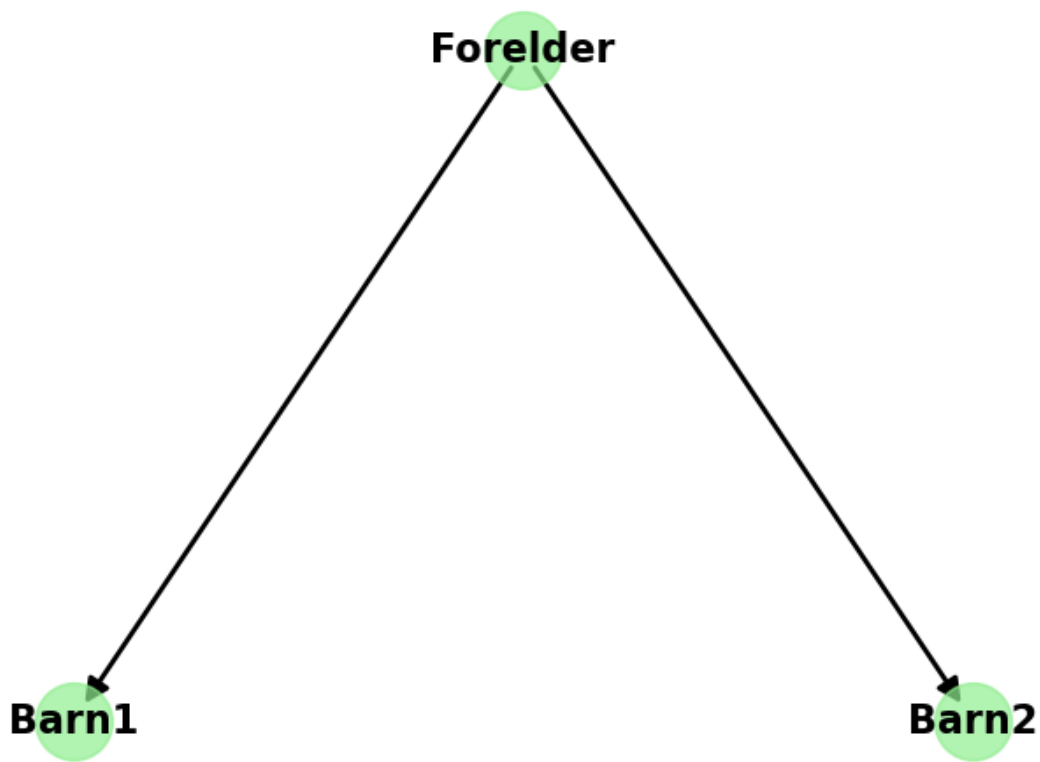
```
plt.title("Rettet graf (forelder-barn) / Directed graph (parent-child)")
plt.axis('off')
plt.show()

print(f"Antall noder: {D.number_of_nodes()}")
print(f"Antall kanter: {D.number_of_edges()}")
print(f"Number of nodes: {D.number_of_nodes()}")
print(f"Number of edges: {D.number_of_edges()}")
```

Eksempel 2: Rettet graf

Example 2: Directed graph

Rettet graf (forelder-barn) / Directed graph (parent-child)



```
Antall noder: 3
Antall kanter: 2
Number of nodes: 3
Number of edges: 2
```

2.9 Similaritetsmål i slektstre

Similaritetsmål måler hvor like to personer er basert på forskjellige attributter. Dette er forskjellig fra avstandsmatriser som bare måler strukturell avstand.

Typersimilaritetsmål / Types of similarity measures: - **Kjønn:** Like kjønn = høy similarity, forskjellig kjønn = lav similarity - **Alder:** Mindre aldersforskjell = høyere similarity - **Geografisk:** Samme bosted = høy similarity - **Yrke:** Samme yrke = høy similarity - **Navn:** Likhets i navn (Levenshtein distance)

Kombinert similarity / Combined similarity: Vi kan kombinere flere similarity-mål ved å gi dem vekt og beregne et vektet gjennomsnitt.

Anvendelse / Applications: - Finne personer med lik bakgrunn - Identifisere potensielle slektninger - Gruppere familiemedlemmer etter likhet - Analysere familiemønstre

2.9.1 Spesielle typer grafer / Special Types of Graphs

1. Tre (Tree) Et **tre** er en spesiell type graf som: - Har ingen sykler (ikke kan gå i sirkel) - Er sammenkoblet (alle noder kan nås fra hverandre) - Har nøyaktig $n-1$ kanter for n noder

A `tree**` is a special type of graph that:** - Has no cycles (cannot go in circles) - Is connected (all nodes can be reached from each other) - Has exactly $n-1$ edges for n nodes

2. Sykler (Cycles) En **sykel** er når du kan gå fra en node og komme tilbake til samme node. I slektstrær vil dette være umulig (en person kan ikke være sin egen forfader).

A `cycle**` is when you can go from a node and return to the same node. In family trees this would be impossible (a person cannot be their own ancestor).**

3. Grad (Degree) **Grad** er antall kanter som er koblet til en node: - **Inngrad:** Antall kanter som kommer inn til noden - **Utgrad:** Antall kanter som går ut fra noden

Degree is the number of edges connected to a node: - In-degree: **Number of edges coming into the node** - Out-degree**: Number of edges going out from the node

2.10 Slektstre med attributter

I denne visualiseringen viser vi slektstreet med alle attributtene vi har definert. Dette gir oss en komplett oversikt over både strukturen og egenskapene til familiemedlemmene.

Visualiseringselementer / Visualization elements: - **Node-farger:** Blå for menn, rød for kvinner - **Node-størrelse:** Proporsjonal med alder (eldre = større) - **Node-labels:** Fornavn for bedre lesbarhet - **Attributt-bokser:** Viser kjønn, alder, bosted og yrke - **Kant-typer:** Heltrukne for forelder-barn, stiplete for ekteskap - **Legend:** Forklarer alle elementer

Insights vi kan få / Insights we can gain: - Se aldersfordeling i familien - Identifisere geografiske mønstre - Se yrkesfordeling - Forstå familierelasjoner visuelt

```
[23]: # Eksempel 3: Tre vs. Graf med sykler / Tree vs. Graph with cycles
print(" Eksempel 3: Tre vs. Graf med sykler")
print(" Example 3: Tree vs. Graph with cycles")

# Opprett et tre
tree = nx.Graph()
tree.add_edges_from([(1, 2), (1, 3), (2, 4), (2, 5)])
```

```

# Opprett en graf med sykler
cycle_graph = nx.Graph()
cycle_graph.add_edges_from([(1, 2), (2, 3), (3, 4), (4, 1)])

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

# Tegn treet med fiksert layout
pos1 = {
    1: (0, 1),      # Node 1 øverst i midten
    2: (0, 0),      # Node 2 i midten
    3: (-0.8, 0.5), # Node 3 til venstre
    4: (-0.5, -0.8), # Node 4 nederst til venstre
    5: (0.5, -0.8)  # Node 5 nederst til høyre
}
nx.draw(tree, pos1, with_labels=True, node_color='lightblue',
        node_size=1000, font_size=16, font_weight='bold', ax=ax1)
ax1.set_title("Tre (ingen sykler) / Tree (no cycles)")

# Tegn grafen med sykler med fiksert layout
pos2 = {
    1: (0, 1),      # Node 1 øverst
    2: (1, 0),      # Node 2 til høyre
    3: (0, -1),     # Node 3 nederst
    4: (-1, 0)      # Node 4 til venstre
}
nx.draw(cycle_graph, pos2, with_labels=True, node_color='lightcoral',
        node_size=1000, font_size=16, font_weight='bold', ax=ax2)
ax2.set_title("Graf med sykler / Graph with cycles")

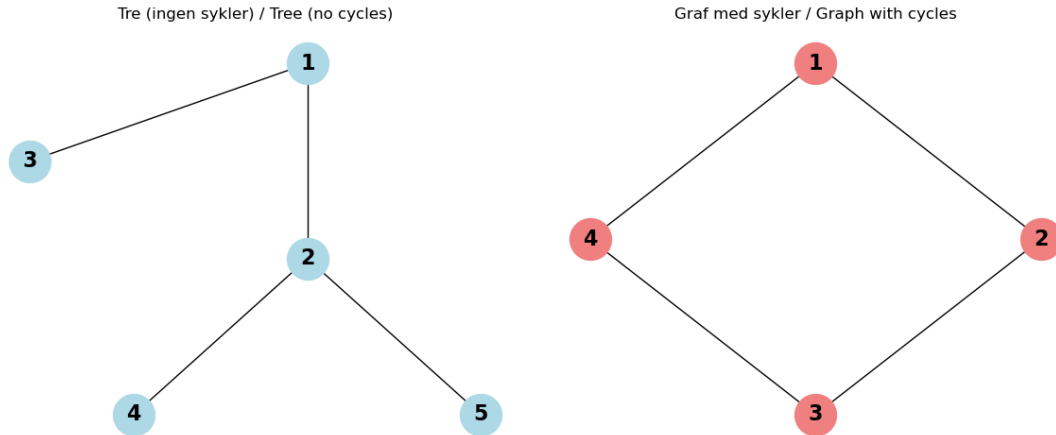
plt.tight_layout()
plt.show()

print(f"Tre - Er tre: {nx.is_tree(tree)}")
print(f"Sykelgraf - Er tre: {nx.is_tree(cycle_graph)}")
print(f"Tree - Is tree: {nx.is_tree(tree)}")
print(f"Cycle graph - Is tree: {nx.is_tree(cycle_graph)}")

```

Eksempel 3: Tre vs. Graf med sykler

Example 3: Tree vs. Graph with cycles



Tre - Er tre: True
 Sykelgraf - Er tre: False
 Tree - Is tree: True
 Cycle graph - Is tree: False

2.11 Visualisering av alle similarity-matriser

I denne seksjonen visualiserer vi alle similarity-matrisene vi har beregnet som heatmaps. Dette gir oss en omfattende oversikt over hvordan familiemedlemmene er like hverandre på forskjellige måter.

Hva vi visualiserer / What we visualize: - **Gender Similarity:** Kjønnsbasert similarity - **Age Similarity:** Aldersbasert similarity
 - **Location Similarity:** Geografisk similarity - **Profession Similarity:** Yrkesbasert similarity - **Name Similarity:** Navnebasert similarity - **Combined Similarity:** Kombinert similarity-score

Analytiske insights / Analytical insights: - Identifiser par med høyest similarity - Se korrelasjoner mellom forskjellige similarity-mål - Forstå hvilke attributter som påvirker similarity mest - Finne interessante mønstre i familien

Heatmap-tolkning / Heatmap interpretation: - **Lys farge:** Høy similarity (mer lik) - **Mørk farge:** Lav similarity (mindre lik) - **Diagonal:** Alltid 1.0 (person lik seg selv)

2.12 Del 3: Slegtstrær SOM grafer / Part 3: Family Trees AS Graphs

2.12.1 Kobling mellom slektstrær og grafteori / Connection between Family Trees and Graph Theory

Nå skal vi se hvordan slektstrær kan representeres som grafer:

Now we'll see how family trees can be represented as graphs:

Mapping / Kartlegging

Slektstre / Family Tree	Graf / Graph
Person / Person	Node / Node
Slektsrelasjon / Family relationship	Kant / Edge
Forelder-barn / Parent-child	Retted kant / Directed edge
Ekteskap / Marriage	Ikke-retted kant / Undirected edge
Generasjon / Generation	Avstand fra rot / Distance from root
Slektskap / Kinship	Sti mellom noder / Path between nodes

Hvorfor NetworkX er perfekt for slektstrær / Why NetworkX is perfect for family trees

1. **Kraftige algoritmer:** NetworkX har innebygde algoritmer for å finne stier, beregne avstander, og analysere grafer
2. **Visualisering:** Enkelt å lage visuelle representasjoner
3. **Fleksibilitet:** Støtter både rettede og ikke-rettede grafer
4. **Skalerbarhet:** Kan håndtere store slektstrær
5. **Analyse:** Kan beregne komplekse slektskap automatisk

English: 1. **Powerful algorithms:** NetworkX has built-in algorithms for finding paths, calculating distances, and analyzing graphs 2. **Visualization:** Easy to create visual representations 3.

Flexibility: Supports both directed and undirected graphs 4. **Scalability:** Can handle large family trees 5. **Analysis:** Can calculate complex relationships automatically

```
[24]: # Eksempel 4: Bygg et enkelt slektstre som graf / Build a simple family tree as graph
      ↪graph
print(" Eksempel 4: Bygg et enkelt slektstre som graf")
print(" Example 4: Build a simple family tree as graph")

# Opprett et retnet graf for slektstreet
family_graph = nx.DiGraph()

# Legg til personer som noder
family_graph.add_node("Bestefar", name="Erik Lundervold", generation=1)
family_graph.add_node("Bestemor", name="Ingrid Hansen", generation=1)
family_graph.add_node("Far", name="Arvid Lundervold", generation=2)
family_graph.add_node("Mor", name="Anna Pedersen", generation=2)
family_graph.add_node("Barn1", name="Lars Lundervold", generation=3)
family_graph.add_node("Barn2", name="Kari Lundervold", generation=3)

# Legg til forelder-barn relasjoner (rettede kanter)
family_graph.add_edge("Bestefar", "Far", relation="parent-child")
family_graph.add_edge("Bestemor", "Far", relation="parent-child")
family_graph.add_edge("Far", "Barn1", relation="parent-child")
family_graph.add_edge("Far", "Barn2", relation="parent-child")
family_graph.add_edge("Mor", "Barn1", relation="parent-child")
family_graph.add_edge("Mor", "Barn2", relation="parent-child")
```



```

# Legg til ekteskap (uretnede kanter)
family_graph.add_edge("Bestefar", "Bestemor", relation="marriage")
family_graph.add_edge("Far", "Mor", relation="marriage")

# Visualiser med fiksert slektstre-layout
plt.figure(figsize=(12, 8))

# Definer fiksert posisjonering som et typisk slektstre
pos = {
    "Bestefar": (0, 2),      # Øverst til venstre
    "Bestemor": (2, 2),     # Øverst til høyre
    "Far": (0, 1),          # Midten til venstre
    "Mor": (2, 1),          # Midten til høyre
    "Barn1": (0, 0),        # Nederst til venstre
    "Barn2": (2, 0)         # Nederst til høyre
}

# Tegn noder
nx.draw_networkx_nodes(family_graph, pos, node_color='lightblue',
                       node_size=2000, alpha=0.7)

# Tegn kanter med forskjellige farger
parent_child_edges = [(u, v) for u, v, d in family_graph.edges(data=True)
                       if d.get('relation') == 'parent-child']
marriage_edges = [(u, v) for u, v, d in family_graph.edges(data=True)
                   if d.get('relation') == 'marriage']

nx.draw_networkx_edges(family_graph, pos, edgelist=parent_child_edges,
                       edge_color='black', arrows=True, arrowsize=20, width=2)
nx.draw_networkx_edges(family_graph, pos, edgelist=marriage_edges,
                       edge_color='red', arrows=False, width=3, style='dashed')

# Legg til etiketter
labels = {node: family_graph.nodes[node]['name'] for node in family_graph.
          nodes()}
nx.draw_networkx_labels(family_graph, pos, labels, font_size=10,
                        font_weight='bold')

plt.title("Slektstre som graf / Family tree as graph")
plt.axis('off')
plt.show()

print(f"Antall personer: {family_graph.number_of_nodes()}")
print(f"Antall relasjoner: {family_graph.number_of_edges()}")
print(f"Number of people: {family_graph.number_of_nodes()}")
print(f"Number of relationships: {family_graph.number_of_edges()}")

```

```

# Vis familierelasjoner som tabell
print(f"\n Familierelasjoner / Family relationships:")
print(f"{'='*60}")
print(f"{'Fra (From)':<20} {'Til (To)':<20} {'Relasjon (Relation)':<20}")
print(f"{'='*60}")

for u, v, data in family_graph.edges(data=True):
    from_name = family_graph.nodes[u]['name']
    to_name = family_graph.nodes[v]['name']
    relation = data.get('relation', 'unknown')

    # Oversett relasjon til norsk/engelsk
    if relation == 'parent-child':
        rel_norsk = "Forelder-barn"
        rel_engelsk = "Parent-child"
    elif relation == 'marriage':
        rel_norsk = "Ekteskap"
        rel_engelsk = "Marriage"
    else:
        rel_norsk = relation
        rel_engelsk = relation

    print(f"{'from_name:<20} {'to_name:<20} {'rel_norsk}/{rel_engelsk:<20}")

print(f"{'='*60}")

# Vis generasjonsinformasjon
print(f"\n Generasjonsoversikt / Generation overview:")
generations = {}
for node, data in family_graph.nodes(data=True):
    gen = data.get('generation', 'unknown')
    if gen not in generations:
        generations[gen] = []
    generations[gen].append(data['name'])

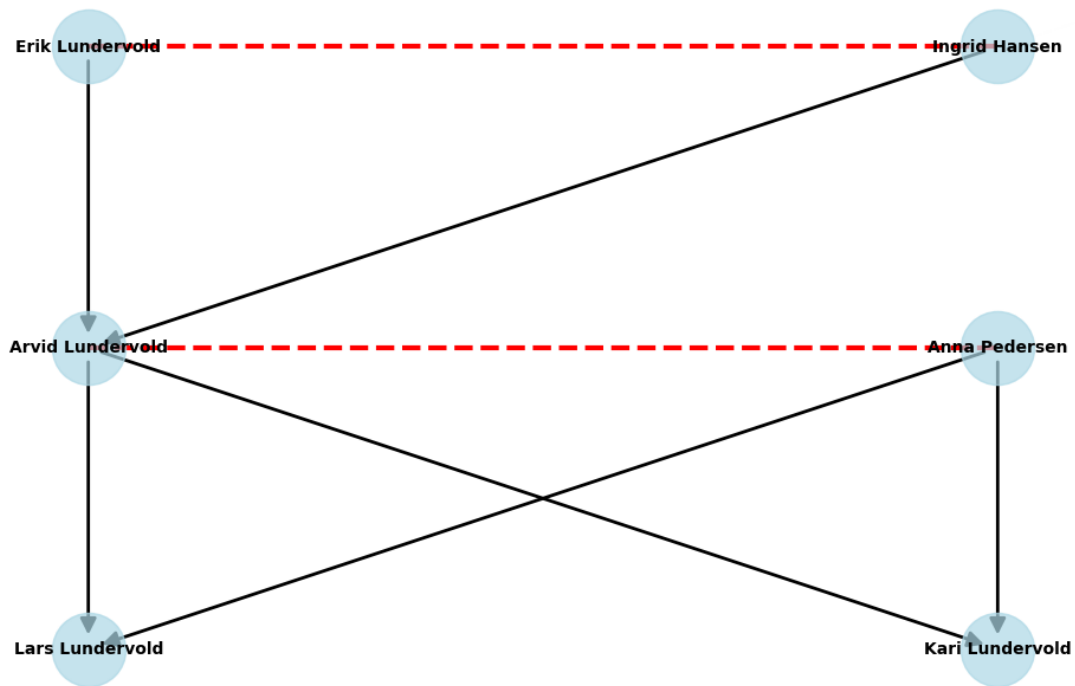
for gen in sorted(generations.keys()):
    print(f"Generasjon {gen}: {' '.join(generations[gen])}")

```

Eksempel 4: Bygg et enkelt slektstre som graf

Example 4: Build a simple family tree as graph

Slektstre som graf / Family tree as graph



Antall personer: 6
 Antall relasjoner: 8
 Number of people: 6
 Number of relationships: 8

Familierelasjoner / Family relationships:

Fra (From)	Til (To)	Relasjon (Relation)
Erik Lundervold	Arvid Lundervold	Forelder-barn/Parent-child
Erik Lundervold	Ingrid Hansen	Ekteskap/Marriage
Ingrid Hansen	Arvid Lundervold	Forelder-barn/Parent-child
Arvid Lundervold	Lars Lundervold	Forelder-barn/Parent-child
Arvid Lundervold	Kari Lundervold	Forelder-barn/Parent-child
Arvid Lundervold	Anna Pedersen	Ekteskap/Marriage
Anna Pedersen	Lars Lundervold	Forelder-barn/Parent-child
Anna Pedersen	Kari Lundervold	Forelder-barn/Parent-child

Generasjonsoversikt / Generation overview:

Generasjon 1: Erik Lundervold, Ingrid Hansen
 Generasjon 2: Arvid Lundervold, Anna Pedersen
 Generasjon 3: Lars Lundervold, Kari Lundervold

```

[25]: # Nabomatrise for slektstreet / Adjacency matrix for the family tree
print(" Nabomatrise / Adjacency Matrix")
print("  Norsk forklaring:")
print("En nabomatrise er en kvadratisk matrise som representerer en graf.")
print("Hver rad og kolonne representerer en node, og verdien 1/0 viser om_
↳ nodene er koblet.")
print("Dette er en alternativ og likeverdig representasjon av grafen.")
print("")
print("  English explanation:")
print("An adjacency matrix is a square matrix that represents a graph.")
print("Each row and column represents a node, and the value 1/0 shows if nodes_
↳ are connected.")
print("This is an alternative and equivalent representation of the graph.")
print("")

# Opprett nabomatrise
nodes = list(family_graph.nodes())
n = len(nodes)

print(" 6x6 Nabomatrise (1 = koblet, 0 = ikke koblet):")
print(" 6x6 Adjacency matrix (1 = connected, 0 = not connected):")
print("")

# Lag header med fornavn
header = f"{'Node':<20}"
for node in nodes:
    first_name = family_graph.nodes[node]['name'].split()[0] # Første ord_
↳ (fornavn)
    header += f"{first_name:<12}"
print(header)
print("-" * (20 + 12 * len(nodes)))

# Beregn og vis nabomatrisen
for i, node1 in enumerate(nodes):
    first_name1 = family_graph.nodes[node1]['name'].split()[0] # Første ord_
↳ (fornavn)
    row = f"{first_name1:<20}"
    for j, node2 in enumerate(nodes):
        if i == j:
            # Diagonal - samme node (ikke koblet til seg selv)
            adjacency = 0
        else:
            # Sjekk om det finnes en kant mellom nodene
            if family_graph.has_edge(node1, node2):
                adjacency = 1
            else:
                adjacency = 0

```

```

        row += f"{adjacency:<12}"
    print(row)

print("")
print(" Forklaring av verdier / Explanation of values:")
print("• 1: Det finnes en kant mellom nodene / There is an edge between the_
↳nodes")
print("• 0: Ingen kant mellom nodene / No edge between the nodes")
print("• Diagonal: Alltid 0 (ingen node kobler til seg selv) / Always 0 (no_
↳self-loops)")
print("• Matrisen er IKKE symmetrisk på grunn av rettede og ikke-rettede_
↳kanter")
print("• The matrix is NOT symmetric due to directed and undirected edges")
print("")

# Vis familierelasjoner med spesifikke titler
print(" Familierelasjoner / Family relationships:")
print("")

# Definer familierelasjoner
family_relations = {
    ("Bestefar", "Far"): "farfar",
    ("Bestemor", "Far"): "farmor",
    ("Far", "Barn1"): "far",
    ("Far", "Barn2"): "far",
    ("Mor", "Barn1"): "mor",
    ("Mor", "Barn2"): "mor",
    ("Bestefar", "Bestemor"): "ektefelle",
    ("Far", "Mor"): "ektefelle"
}

print(" Norske familierelasjoner:")
for (u, v), relation in family_relations.items():
    u_name = family_graph.nodes[u]['name'].split()[0]
    v_name = family_graph.nodes[v]['name'].split()[0]
    print(f" {u_name} er {relation} til {v_name}")

print("")
print(" English family relationships:")
english_relations = {
    ("Bestefar", "Far"): "grandfather",
    ("Bestemor", "Far"): "grandmother",
    ("Far", "Barn1"): "father",
    ("Far", "Barn2"): "father",
    ("Mor", "Barn1"): "mother",
    ("Mor", "Barn2"): "mother",

```

```

    ("Bestefar", "Bestemor"): "spouse",
    ("Far", "Mor"): "spouse"
}

for (u, v), relation in english_relations.items():
    u_name = family_graph.nodes[u]['name'].split()[0]
    v_name = family_graph.nodes[v]['name'].split()[0]
    print(f" {u_name} is {relation} to {v_name}")

print("")
print("="*60)
print(" LEGEND FOR GRAF-VISUALISERING / LEGEND FOR GRAPH VISUALIZATION")
print("="*60)
print(" Norsk:")
print("• Heltrukne kanter (→): Forelder-barn relasjoner (rettede kanter)")
print("• Stiplede kanter (---): Ekteskap (ikke-rettede kanter)")
print("• Piler (→): Viser retning fra forelder til barn")
print("• Ingen piler (---): Ekteskap er gjensidig")
print("• Nabomatrisen er IKKE symmetrisk på grunn av kanttyper")
print("")
print(" English:")
print("• Solid edges (→): Parent-child relationships (directed edges)")
print("• Dashed edges (---): Marriages (undirected edges)")
print("• Arrows (→): Show direction from parent to child")
print("• No arrows (---): Marriage is mutual")
print("• The adjacency matrix is NOT symmetric due to edge types")
print("")

print("="*60)
print(" MATEMATISK FORKLARING / MATHEMATICAL EXPLANATION")
print("="*60)
print(" Norsk:")
print("• Nabomatrise:  $A[i,j] = 1$  hvis det finnes kant fra node i til node j,  $\perp$   

    ↪ellers 0")
print("• Rettede kanter:  $A[i,j]$  kan være forskjellig fra  $A[j,i]$ ")
print("• Ikke-rettede kanter:  $A[i,j] = A[j,i]$  (symmetrisk)")
print("• Blandet graf: Kombinasjon av rettede og ikke-rettede kanter")
print("• Nabomatrise og graf er likeverdige representasjoner")
print("")
print(" English:")
print("• Adjacency matrix:  $A[i,j] = 1$  if there is an edge from node i to node  $\perp$   

    ↪j, else 0")
print("• Directed edges:  $A[i,j]$  may differ from  $A[j,i]$ ")
print("• Undirected edges:  $A[i,j] = A[j,i]$  (symmetric)")
print("• Mixed graph: Combination of directed and undirected edges")
print("• Adjacency matrix and graph are equivalent representations")

```

Nabomatrise / Adjacency Matrix

Norsk forklaring:

En nabomatrise er en kvadratisk matrise som representerer en graf.

Hver rad og kolonne representerer en node, og verdien 1/0 viser om nodene er koblet.

Dette er en alternativ og likeverdig representasjon av grafen.

English explanation:

An adjacency matrix is a square matrix that represents a graph.

Each row and column represents a node, and the value 1/0 shows if nodes are connected.

This is an alternative and equivalent representation of the graph.

6x6 Nabomatrise (1 = koblet, 0 = ikke koblet):

6x6 Adjacency matrix (1 = connected, 0 = not connected):

Node	Erik	Ingrid	Arvid	Anna	Lars
Kari					

Erik	0	1	1	0	0
0					
Ingrid	0	0	1	0	0
0					
Arvid	0	0	0	1	1
1					
Anna	0	0	0	0	1
1					
Lars	0	0	0	0	0
0					
Kari	0	0	0	0	0
0					

Forklaring av verdier / Explanation of values:

- 1: Det finnes en kant mellom nodene / There is an edge between the nodes
- 0: Ingen kant mellom nodene / No edge between the nodes
- Diagonal: Alltid 0 (ingen node kobler til seg selv) / Always 0 (no self-loops)
- Matrisen er IKKE symmetrisk på grunn av rettede og ikke-rettede kanter
- The matrix is NOT symmetric due to directed and undirected edges

Familierelasjoner / Family relationships:

Norske familierelasjoner:

Erik er farfar til Arvid

Ingrid er farmor til Arvid

Arvid er far til Lars

Arvid er far til Kari

Anna er mor til Lars

Anna er mor til Kari
Erik er ektefelle til Ingrid
Arvid er ektefelle til Anna

English family relationships:
Erik is grandfather to Arvid
Ingrid is grandmother to Arvid
Arvid is father to Lars
Arvid is father to Kari
Anna is mother to Lars
Anna is mother to Kari
Erik is spouse to Ingrid
Arvid is spouse to Anna

=====

LEGEND FOR GRAF-VISUALISERING / LEGEND FOR GRAPH VISUALIZATION

=====

Norsk:

- Heltrukne kanter (\rightarrow): Forelder-barn relasjoner (rettede kanter)
- Stiplede kanter ($---$): Ekteskap (ikke-rettede kanter)
- Piler (\rightarrow): Viser retning fra forelder til barn
- Ingen piler ($---$): Ekteskap er gjensidig
- Nabomatrisen er IKKE symmetrisk på grunn av kanttyper

English:

- Solid edges (\rightarrow): Parent-child relationships (directed edges)
- Dashed edges ($---$): Marriages (undirected edges)
- Arrows (\rightarrow): Show direction from parent to child
- No arrows ($---$): Marriage is mutual
- The adjacency matrix is NOT symmetric due to edge types

=====

MATEMATISK FORKLARING / MATHEMATICAL EXPLANATION

=====

Norsk:

- Nabomatrise: $A[i,j] = 1$ hvis det finnes kant fra node i til node j , ellers 0
- Rettede kanter: $A[i,j]$ kan være forskjellig fra $A[j,i]$
- Ikke-rettede kanter: $A[i,j] = A[j,i]$ (symmetrisk)
- Blandet graf: Kombinasjon av rettede og ikke-rettede kanter
- Nabomatrise og graf er likeverdige representasjoner

English:

- Adjacency matrix: $A[i,j] = 1$ if there is an edge from node i to node j , else 0
- Directed edges: $A[i,j]$ may differ from $A[j,i]$
- Undirected edges: $A[i,j] = A[j,i]$ (symmetric)
- Mixed graph: Combination of directed and undirected edges
- Adjacency matrix and graph are equivalent representations


```

[26]: # Node-orden, inn-orden og ut-orden / Node degree, in-degree and out-degree
print(" Node-orden i blandede grafer / Node degree in mixed graphs")
print("  Norsk forklaring:")
print("Node-orden er antall kanter som er koblet til en node.")
print("I blandede grafer (som slektstrær) har vi både rettede og urettede_
    ↪kanter.")
print("Vi skiller mellom inn-orden (kanter som kommer inn) og ut-orden (kanter_
    ↪som går ut).")
print("")
print("  English explanation:")
print("Node degree is the number of edges connected to a node.")
print("In mixed graphs (like family trees) we have both directed and undirected_
    ↪edges.")
print("We distinguish between in-degree (edges coming in) and out-degree (edges_
    ↪going out).")
print("")

# Beregn node-ordener for hver node
print(" Node-orden for hver person / Node degree for each person:")
print("")

for node in family_graph.nodes():
    name = family_graph.nodes[node]['name']
    first_name = name.split()[0]

    # Beregn forskjellige ordener
    total_degree = family_graph.degree(node) # Totalt antall kanter
    in_degree = family_graph.in_degree(node) # Antall kanter som kommer inn
    out_degree = family_graph.out_degree(node) # Antall kanter som går ut

    print(f" {first_name} ({name}):")
    print(f"   Totalt antall kanter: {total_degree}")
    print(f"   Total number of edges: {total_degree}")
    print(f"   Inn-orden (kanter inn): {in_degree}")
    print(f"   In-degree (edges in): {in_degree}")
    print(f"   Ut-orden (kanter ut): {out_degree}")
    print(f"   Out-degree (edges out): {out_degree}")
    print("")

# Vis detaljert analyse av kanttyper
print(" Detaljert analyse av kanttyper / Detailed analysis of edge types:")
print("")

for node in family_graph.nodes():
    name = family_graph.nodes[node]['name']
    first_name = name.split()[0]

```

```

# Analyser hver kanttype
parent_child_in = 0
parent_child_out = 0
marriage_edges = 0

# Sjekk alle kanter til denne noden
for neighbor in family_graph.neighbors(node):
    edge_data = family_graph.get_edge_data(node, neighbor)
    if edge_data and edge_data.get('relation') == 'parent-child':
        # Sjekk retning
        if family_graph.has_edge(node, neighbor):
            parent_child_out += 1
        if family_graph.has_edge(neighbor, node):
            parent_child_in += 1
    elif edge_data and edge_data.get('relation') == 'marriage':
        marriage_edges += 1

print(f" {first_name}:")
print(f"   Forelder-barn (ut): {parent_child_out} kanter")
print(f"   Parent-child (out): {parent_child_out} edges")
print(f"   Forelder-barn (inn): {parent_child_in} kanter")
print(f"   Parent-child (in): {parent_child_in} edges")
print(f"   Ekteskap: {marriage_edges} kanter")
print(f"   Marriage: {marriage_edges} edges")
print("")

print("="*60)
print(" MATEMATISK DEFINISJONER / MATHEMATICAL DEFINITIONS")
print("="*60)
print(" Norsk:")
print("• Node-orden (degree): Totalt antall kanter koblet til en node")
print("• Inn-orden (in-degree): Antall kanter som kommer inn til noden")
print("• Ut-orden (out-degree): Antall kanter som går ut fra noden")
print("• For ikke-rettede kanter: Teller som både inn og ut")
print("• For rettede kanter: Teller kun i én retning")
print("• Blandet graf: Kombinasjon av begge kanttyper")
print("")
print(" English:")
print("• Node degree: Total number of edges connected to a node")
print("• In-degree: Number of edges coming into the node")
print("• Out-degree: Number of edges going out from the node")
print("• For undirected edges: Counts as both in and out")
print("• For directed edges: Counts only in one direction")
print("• Mixed graph: Combination of both edge types")
print("")

# Vis eksempler på hvordan dette gjelder for slektstrær

```

```

print(" Eksempler fra slektstreet / Examples from the family tree:")
print("")
print("  Norsk:")
print("• Bestefar: Ut-orden 2 (til Far og Bestemor), Inn-orden 1 (fra_
  ↳Bestemor)")
print("• Far: Ut-orden 2 (til Barn1 og Barn2), Inn-orden 3 (fra Bestefar,
  ↳Bestemor, Mor)")
print("• Barn1: Ut-orden 0, Inn-orden 2 (fra Far og Mor)")
print("")
print("  English:")
print("• Grandfather: Out-degree 2 (to Father and Grandmother), In-degree 1_
  ↳(from Grandmother)")
print("• Father: Out-degree 2 (to Child1 and Child2), In-degree 3 (from_
  ↳Grandfather, Grandmother, Mother)")
print("• Child1: Out-degree 0, In-degree 2 (from Father and Mother)")

```

Node-orden i blandede grafer / Node degree in mixed graphs

Norsk forklaring:

Node-orden er antall kanter som er koblet til en node.

I blandede grafer (som slektstrær) har vi både rettede og urettede kanter.

Vi skiller mellom inn-orden (kanter som kommer inn) og ut-orden (kanter som går ut).

English explanation:

Node degree is the number of edges connected to a node.

In mixed graphs (like family trees) we have both directed and undirected edges.

We distinguish between in-degree (edges coming in) and out-degree (edges going out).

Node-orden for hver person / Node degree for each person:

Erik (Erik Lundervold):

Totalt antall kanter: 2
 Total number of edges: 2
 Inn-orden (kanter inn): 0
 In-degree (edges in): 0
 Ut-orden (kanter ut): 2
 Out-degree (edges out): 2

Ingrid (Ingrid Hansen):

Totalt antall kanter: 2
 Total number of edges: 2
 Inn-orden (kanter inn): 1
 In-degree (edges in): 1
 Ut-orden (kanter ut): 1
 Out-degree (edges out): 1

Arvid (Arvid Lundervold):
Totalt antall kanter: 5
Total number of edges: 5
Inn-orden (kanter inn): 2
In-degree (edges in): 2
Ut-orden (kanter ut): 3
Out-degree (edges out): 3

Anna (Anna Pedersen):
Totalt antall kanter: 3
Total number of edges: 3
Inn-orden (kanter inn): 1
In-degree (edges in): 1
Ut-orden (kanter ut): 2
Out-degree (edges out): 2

Lars (Lars Lundervold):
Totalt antall kanter: 2
Total number of edges: 2
Inn-orden (kanter inn): 2
In-degree (edges in): 2
Ut-orden (kanter ut): 0
Out-degree (edges out): 0

Kari (Kari Lundervold):
Totalt antall kanter: 2
Total number of edges: 2
Inn-orden (kanter inn): 2
In-degree (edges in): 2
Ut-orden (kanter ut): 0
Out-degree (edges out): 0

Detaljert analyse av kanttyper / Detailed analysis of edge types:

Erik:
Forelder-barn (ut): 1 kanter
Parent-child (out): 1 edges
Forelder-barn (inn): 0 kanter
Parent-child (in): 0 edges
Ekteskap: 1 kanter
Marriage: 1 edges

Ingrid:
Forelder-barn (ut): 1 kanter
Parent-child (out): 1 edges
Forelder-barn (inn): 0 kanter
Parent-child (in): 0 edges
Ekteskap: 0 kanter

Marriage: 0 edges

Arvid:

Forelder-barn (ut): 2 kanter
Parent-child (out): 2 edges
Forelder-barn (inn): 0 kanter
Parent-child (in): 0 edges
Ekteskap: 1 kanter
Marriage: 1 edges

Anna:

Forelder-barn (ut): 2 kanter
Parent-child (out): 2 edges
Forelder-barn (inn): 0 kanter
Parent-child (in): 0 edges
Ekteskap: 0 kanter
Marriage: 0 edges

Lars:

Forelder-barn (ut): 0 kanter
Parent-child (out): 0 edges
Forelder-barn (inn): 0 kanter
Parent-child (in): 0 edges
Ekteskap: 0 kanter
Marriage: 0 edges

Kari:

Forelder-barn (ut): 0 kanter
Parent-child (out): 0 edges
Forelder-barn (inn): 0 kanter
Parent-child (in): 0 edges
Ekteskap: 0 kanter
Marriage: 0 edges

=====

MATEMATISK DEFINISJONER / MATHEMATICAL DEFINITIONS

=====

Norsk:

- Node-orden (degree): Totalt antall kanter koblet til en node
- Inn-orden (in-degree): Antall kanter som kommer inn til noden
- Ut-orden (out-degree): Antall kanter som går ut fra noden
- For ikke-rettede kanter: Teller som både inn og ut
- For rettede kanter: Teller kun i én retning
- Blandet graf: Kombinasjon av begge kanttyper

English:

- Node degree: Total number of edges connected to a node
- In-degree: Number of edges coming into the node

- Out-degree: Number of edges going out from the node
- For undirected edges: Counts as both in and out
- For directed edges: Counts only in one direction
- Mixed graph: Combination of both edge types

Eksempler fra slektstreet / Examples from the family tree:

Norsk:

- Bestefar: Ut-orden 2 (til Far og Bestemor), Inn-orden 1 (fra Bestemor)
- Far: Ut-orden 2 (til Barn1 og Barn2), Inn-orden 3 (fra Bestefar, Bestemor, Mor)
- Barn1: Ut-orden 0, Inn-orden 2 (fra Far og Mor)

English:

- Grandfather: Out-degree 2 (to Father and Grandmother), In-degree 1 (from Grandmother)
- Father: Out-degree 2 (to Child1 and Child2), In-degree 3 (from Grandfather, Grandmother, Mother)
- Child1: Out-degree 0, In-degree 2 (from Father and Mother)

```
[27]: # Avstands-matrise for slektstreet / Distance matrix for the family tree
print("  Avstands-matrise / Distance Matrix")
print("  Norsk forklaring:")
print("En avstands-matrise viser korteste sti-lengde mellom alle par av noder i
    ↳ grafen.")
print("VIKTIG: Vi skiller mellom biologisk beslektede og ikke-biologisk
    ↳ beslektede relasjoner.")
print("Forelder-barn og søsken er biologisk beslektet, ekteskap er som regel
    ↳ ikke biologisk beslektet.")
print("")
print("  English explanation:")
print("A distance matrix shows the shortest path length between all pairs of
    ↳ nodes in the graph.")
print("IMPORTANT: We distinguish between biologically related and
    ↳ non-biologically related relationships.")
print("Parent-child and siblings are biologically related, marriages are
    ↳ usually not biologically related.")
print("")

# Beregn korteste sti-lengder mellom alle noder
nodes = list(family_graph.nodes())

print("  6x6 Avstands-matrise (korteste sti-lengde):")
print("  6x6 Distance matrix (shortest path length):")
print("")

# Lag header med fornavn
```

```

header = f"{'Node':<15}"
for node in nodes:
    first_name = family_graph.nodes[node]['name'].split()[0] # Første ord
    ↪(fornavn)
    header += f"{first_name:<12}"
print(header)
print("-" * (15 + 12 * len(nodes)))

# Beregn og vis matrisen
for i, node1 in enumerate(nodes):
    first_name1 = family_graph.nodes[node1]['name'].split()[0] # Første ord
    ↪(fornavn)
    row = f"{first_name1:<15}"
    for j, node2 in enumerate(nodes):
        if i == j:
            # Diagonal - samme node
            distance = 0
        else:
            try:
                # Finn korteste sti-lengde
                distance = nx.shortest_path_length(family_graph, node1, node2)
            except nx.NetworkXNoPath:
                # Ingen sti funnet
                distance = float('inf')

        if distance == float('inf'):
            row += f"{'∞':<12}"
        else:
            row += f"{distance:<12}"
    print(row)

print("")
print(" Forklaring av verdier / Explanation of values:")
print("• 0: Samme person / Same person")
print("• 1: Direkte koblet (forelder-barn eller ekteskap) / Directly connected")
print("• 2: Koblet via én mellomperson / Connected via one intermediate person")
print("• 3: Koblet via to mellompersoner / Connected via two intermediate
    ↪persons")
print("• ∞: Ingen sti funnet / No path found")
print("")

# Analyser biologisk vs ikke-biologisk beslektede relasjoner
print(" Biologisk vs ikke-biologisk beslektede relasjoner:")
print(" Biologically vs non-biologically related relationships:")
print("")

# Definer biologisk beslektede relasjoner

```

```

biological_relations = []
non_biological_relations = []

for u, v, data in family_graph.edges(data=True):
    relation = data.get('relation', 'unknown')
    u_name = family_graph.nodes[u]['name'].split()[0]
    v_name = family_graph.nodes[v]['name'].split()[0]

    if relation == 'parent-child':
        biological_relations.append((u_name, v_name, "forelder-barn"))
    elif relation == 'marriage':
        non_biological_relations.append((u_name, v_name, "ekteskap"))

print(" Biologisk beslektede (blodsrelasjoner):")
print(" Biologically related (blood relationships):")
for u_name, v_name, rel_type in biological_relations:
    print(f" {u_name} {v_name} ({rel_type})")

print("")
print(" Ikke-biologisk beslektede (ikke blodsrelasjoner):")
print(" Non-biologically related (non-blood relationships):")
for u_name, v_name, rel_type in non_biological_relations:
    print(f" {u_name} {v_name} ({rel_type})")

print("")
print(" Søsken-relasjoner (implisitte, biologisk beslektede):")
print(" Sibling relationships (implicit, biologically related):")
# Finn søsken-par
siblings = []
for node1 in family_graph.nodes():
    for node2 in family_graph.nodes():
        if node1 != node2:
            # Sjekk om de har samme foreldre
            parents1 = set(family_graph.predecessors(node1))
            parents2 = set(family_graph.predecessors(node2))
            if parents1 and parents2 and parents1 == parents2:
                name1 = family_graph.nodes[node1]['name'].split()[0]
                name2 = family_graph.nodes[node2]['name'].split()[0]
                if (name2, name1, "søsken") not in siblings:
                    siblings.append((name1, name2, "søsken"))

for name1, name2, rel_type in siblings:
    print(f" {name1} {name2} ({rel_type})")

print("")
print("="*60)

```



```

print(" BIOLOGISK OG MATEMATISK FORKLARING / BIOLOGICAL AND MATHEMATICAL_
↳EXPLANATION")
print("="*60)
print(" Norsk:")
print("• Biologisk beslektede: Blodsrelasjoner (forelder-barn, søsken,
↳besteforeldre-barn)")
print("• Ikke-biologisk beslektede: Ikke blodsrelasjoner (ektefeller,
↳svigerfamilie)")
print("• Avstands-matrise: Viser strukturell avstand, ikke biologisk avstand")
print("• Sti-lengde 1: Direkte kobling (biologisk eller ikke-biologisk)")
print("• Sti-lengde 2: Via én mellomperson (kan være biologisk eller ikke)")
print("")
print(" English:")
print("• Biologically related: Blood relationships (parent-child, siblings,
↳grandparent-grandchild)")
print("• Non-biologically related: Non-blood relationships (spouses, in-laws)")
print("• Distance matrix: Shows structural distance, not biological distance")
print("• Path length 1: Direct connection (biological or non-biological)")
print("• Path length 2: Via one intermediate person (can be biological or
↳non-biological)")
print("")

# Vis noen interessante eksempler
print(" Eksempler på korteste stier / Examples of shortest paths:")
print("")

# Finn noen interessante stier
interesting_pairs = [
    ("Bestefar", "Barn1"),
    ("Bestemor", "Barn2"),
    ("Far", "Mor"),
    ("Barn1", "Barn2")
]

for start, end in interesting_pairs:
    try:
        path = nx.shortest_path(family_graph, start, end)
        path_names = [family_graph.nodes[node]['name'].split()[0] for node in
↳path]
        distance = len(path) - 1

        # Analyser om stien går gjennom biologisk eller ikke-biologisk
↳beslektede
        biological_steps = 0
        non_biological_steps = 0

```

```

for i in range(len(path) - 1):
    u, v = path[i], path[i + 1]
    edge_data = family_graph.get_edge_data(u, v)
    if edge_data and edge_data.get('relation') == 'parent-child':
        biological_steps += 1
    elif edge_data and edge_data.get('relation') == 'marriage':
        non_biological_steps += 1

print(f"Sti fra {family_graph.nodes[start]['name'].split()[0]} til {family_graph.nodes[end]['name'].split()[0]}:")
print(f"Path from {family_graph.nodes[start]['name'].split()[0]} to {family_graph.nodes[end]['name'].split()[0]}:")
print(f"  {' → '.join(path_names)} (lengde: {distance})")
print(f"  {' → '.join(path_names)} (length: {distance})")
print(f"  Biologisk beslektede steg: {biological_steps}")
print(f"  Biologically related steps: {biological_steps}")
print(f"  Ikke-biologisk beslektede steg: {non_biological_steps}")
print(f"  Non-biologically related steps: {non_biological_steps}")
print("")
except nx.NetworkXNoPath:
    print(f"Ingen sti fra {start} til {end}")
    print(f"No path from {start} to {end}")
    print("")

```

Avstands-matrise / Distance Matrix

Norsk forklaring:

En avstands-matrise viser korteste sti-lengde mellom alle par av noder i grafen.
 VIKTIG: Vi skiller mellom biologisk beslektede og ikke-biologisk beslektede relasjoner.

Forelder-barn og søsken er biologisk beslektet, ekteskap er som regel ikke biologisk beslektet.

English explanation:

A distance matrix shows the shortest path length between all pairs of nodes in the graph.

IMPORTANT: We distinguish between biologically related and non-biologically related relationships.

Parent-child and siblings are biologically related, marriages are usually not biologically related.

6x6 Avstands-matrise (korteste sti-lengde):

6x6 Distance matrix (shortest path length):

Node	Erik	Ingrid	Arvid	Anna	Lars	Kari

Erik	0	1	1	2	2	2

Ingrid	∞	0	1	2	2	2
Arvid	∞	∞	0	1	1	1
Anna	∞	∞	∞	0	1	1
Lars	∞	∞	∞	∞	0	∞
Kari	∞	∞	∞	∞	∞	0

Forklaring av verdier / Explanation of values:

- 0: Samme person / Same person
- 1: Direkte koblet (forelder-barn eller ekteskap) / Directly connected
- 2: Koblet via én mellomperson / Connected via one intermediate person
- 3: Koblet via to mellompersoner / Connected via two intermediate persons
- ∞ : Ingen sti funnet / No path found

Biologisk vs ikke-biologisk beslektede relasjoner:

Biologically vs non-biologically related relationships:

Biologisk beslektede (blodsrelasjoner):

Biologically related (blood relationships):

Erik Arvid (forelder-barn)

Ingrid Arvid (forelder-barn)

Arvid Lars (forelder-barn)

Arvid Kari (forelder-barn)

Anna Lars (forelder-barn)

Anna Kari (forelder-barn)

Ikke-biologisk beslektede (ikke blodsrelasjoner):

Non-biologically related (non-blood relationships):

Erik Ingrid (ekteskap)

Arvid Anna (ekteskap)

Søsken-relasjoner (implisitte, biologisk beslektede):

Sibling relationships (implicit, biologically related):

Lars Kari (søsken)

=====

BIOLOGISK OG MATEMATISK FORKLARING / BIOLOGICAL AND MATHEMATICAL EXPLANATION

=====

Norsk:

- Biologisk beslektede: Blodsrelasjoner (forelder-barn, søsken, besteforeldre-barn)
- Ikke-biologisk beslektede: Ikke blodsrelasjoner (ektefeller, svigerfamilie)
- Avstands-matrise: Viser strukturell avstand, ikke biologisk avstand
- Sti-lengde 1: Direkte kobling (biologisk eller ikke-biologisk)
- Sti-lengde 2: Via én mellomperson (kan være biologisk eller ikke)

English:

- Biologically related: Blood relationships (parent-child, siblings, grandparent-grandchild)

- Non-biologically related: Non-blood relationships (spouses, in-laws)
- Distance matrix: Shows structural distance, not biological distance
- Path length 1: Direct connection (biological or non-biological)
- Path length 2: Via one intermediate person (can be biological or non-biological)

Eksempler på korteste stier / Examples of shortest paths:

Sti fra Erik til Lars:

Path from Erik to Lars:

```
Erik → Arvid → Lars (lengde: 2)
Erik → Arvid → Lars (length: 2)
Biologisk beslektede steg: 2
Biologically related steps: 2
Ikke-biologisk beslektede steg: 0
Non-biologically related steps: 0
```

Sti fra Ingrid til Kari:

Path from Ingrid to Kari:

```
Ingrid → Arvid → Kari (lengde: 2)
Ingrid → Arvid → Kari (length: 2)
Biologisk beslektede steg: 2
Biologically related steps: 2
Ikke-biologisk beslektede steg: 0
Non-biologically related steps: 0
```

Sti fra Arvid til Anna:

Path from Arvid to Anna:

```
Arvid → Anna (lengde: 1)
Arvid → Anna (length: 1)
Biologisk beslektede steg: 0
Biologically related steps: 0
Ikke-biologisk beslektede steg: 1
Non-biologically related steps: 1
```

Ingen sti fra Barn1 til Barn2

No path from Barn1 to Barn2

```
[28]: # Similaritets-mål i slektstrær / Similarity measures in family trees
print(" Similaritets-mål i slektstrær / Similarity measures in family trees")
print("  Norsk forklaring:")
print("I slektstrær kan vi definere similaritets-mål basert på forskjellige_
    ↪attributter.")
print("Dette er forskjellig fra avstands-matrisen som kun viser strukturell_
    ↪avstand.")
```

```

print("Similaritets-mål kan være knyttet til kjønn, alder, bosted, yrke,
↳navnelikhet, etc.")
print("")
print("  English explanation:")
print("In family trees we can define similarity measures based on different
↳attributes.")
print("This is different from the distance matrix which only shows structural
↳distance.")
print("Similarity measures can be related to gender, age, location, profession,
↳name similarity, etc.")
print("")

# Legg til attributter til familien for å demonstrere similaritets-mål
print(" Legger til attributter for similaritets-analyse:")
print(" Adding attributes for similarity analysis:")
print("")

# Oppdater familien med attributter
family_attributes = {
    "Bestefar": {"gender": "male", "age": 75, "location": "Oslo", "profession":
↳"ingeniør", "name": "Erik Lundervold"},
    "Bestemor": {"gender": "female", "age": 72, "location": "Oslo",
↳"profession": "lærer", "name": "Ingrid Hansen"},
    "Far": {"gender": "male", "age": 45, "location": "Bergen", "profession":
↳"programmerer", "name": "Arvid Lundervold"},
    "Mor": {"gender": "female", "age": 42, "location": "Bergen", "profession":
↳"lege", "name": "Anna Pedersen"},
    "Barn1": {"gender": "male", "age": 18, "location": "Bergen", "profession":
↳"student", "name": "Lars Lundervold"},
    "Barn2": {"gender": "female", "age": 16, "location": "Bergen", "profession":
↳"student", "name": "Kari Lundervold"}
}

# Legg til attributter til grafen
for node, attrs in family_attributes.items():
    for attr, value in attrs.items():
        family_graph.nodes[node][attr] = value

print(" Attributter lagt til:")
for node, attrs in family_attributes.items():
    name = attrs["name"].split()[0]
    print(f" {name}: {attrs['gender']}, {attrs['age']} år,
↳{attrs['location']}, {attrs['profession']}")

print("")

```

```

# Definer similaritets-funksjoner
def gender_similarity(person1, person2):
    """Kjønnsimilaritet: 1 hvis samme kjønn, 0 hvis forskjellig"""
    return 1 if person1["gender"] == person2["gender"] else 0

def age_similarity(person1, person2):
    """Alderssimilaritet: Jo nærmere alder, jo høyere similaritet"""
    age_diff = abs(person1["age"] - person2["age"])
    max_age_diff = 60 # Normaliser til 0-1 skala
    return max(0, 1 - (age_diff / max_age_diff))

def location_similarity(person1, person2):
    """Bostedssimilaritet: 1 hvis samme sted, 0 hvis forskjellig"""
    return 1 if person1["location"] == person2["location"] else 0

def profession_similarity(person1, person2):
    """Yrkessimilaritet: 1 hvis samme yrke, 0 hvis forskjellig"""
    return 1 if person1["profession"] == person2["profession"] else 0

def name_similarity(person1, person2):
    """Navnelikhet: Sjekker om de deler etternavn"""
    name1_parts = person1["name"].split()
    name2_parts = person2["name"].split()
    # Sjekk om de deler etternavn
    return 1 if name1_parts[-1] == name2_parts[-1] else 0

# Beregn similaritets-matrise for hver attributt
similarity_measures = {
    "Kjønn": gender_similarity,
    "Alder": age_similarity,
    "Bosted": location_similarity,
    "Yrke": profession_similarity,
    "Navnelikhet": name_similarity
}

print(" Similaritets-matriser for forskjellige attributter:")
print(" Similarity matrices for different attributes:")
print("")

for measure_name, similarity_func in similarity_measures.items():
    print(f" {measure_name} / {measure_name}:")
    print(f" {measure_name}:")

    # Lag header
    header = f"{'Node':<15}"
    for node in family_graph.nodes():
        first_name = family_graph.nodes[node]['name'].split()[0]

```

```

        header += f"{first_name:<12}"
    print(header)
    print("-" * (15 + 12 * len(family_graph.nodes())))

    # Beregn og vis matrisen
    for node1 in family_graph.nodes():
        first_name1 = family_graph.nodes[node1]['name'].split()[0]
        row = f"{first_name1:<15}"
        for node2 in family_graph.nodes():
            if node1 == node2:
                similarity = 1.0 # Perfekt similaritet med seg selv
            else:
                person1 = {attr: family_graph.nodes[node1][attr] for attr in
↪ ["gender", "age", "location", "profession", "name"]}
                person2 = {attr: family_graph.nodes[node2][attr] for attr in
↪ ["gender", "age", "location", "profession", "name"]}
                similarity = similarity_func(person1, person2)

            row += f"{similarity:<12.2f}"
        print(row)

    print("")

# Kombinert similaritets-mål
print(" Kombinert similaritets-mål / Combined similarity measure:")
print("")

def combined_similarity(person1, person2, weights=None):
    """Kombinert similaritet med vektorer"""
    if weights is None:
        weights = {"Kjønn": 0.2, "Alder": 0.3, "Bosted": 0.2, "Yrke": 0.1,
↪ "Navnelikhet": 0.2}

    total_similarity = 0
    for measure_name, weight in weights.items():
        similarity_func = similarity_measures[measure_name]
        similarity = similarity_func(person1, person2)
        total_similarity += weight * similarity

    return total_similarity

# Vis kombinerte similariteter
print(" Kombinert similaritet (vektet gjennomsnitt):")
print(" Combined similarity (weighted average):")
print("")

# Lag header

```

```

header = f"{'Node':<15}"
for node in family_graph.nodes():
    first_name = family_graph.nodes[node]['name'].split()[0]
    header += f"{'first_name':<12}"
print(header)
print("-" * (15 + 12 * len(family_graph.nodes())))

# Beregn og vis matrisen
for node1 in family_graph.nodes():
    first_name1 = family_graph.nodes[node1]['name'].split()[0]
    row = f"{'first_name1':<15}"
    for node2 in family_graph.nodes():
        if node1 == node2:
            similarity = 1.0
        else:
            person1 = {attr: family_graph.nodes[node1][attr] for attr in
↪ ["gender", "age", "location", "profession", "name"]}
            person2 = {attr: family_graph.nodes[node2][attr] for attr in
↪ ["gender", "age", "location", "profession", "name"]}
            similarity = combined_similarity(person1, person2)

        row += f"{'similarity':<12.2f}"
    print(row)

print("")
print("="*60)
print(" FORKLARING AV SIMILARITETS-MÅL / EXPLANATION OF SIMILARITY MEASURES")
print("="*60)
print(" Norsk:")
print("• Kjønn: 1 hvis samme kjønn, 0 hvis forskjellig")
print("• Alder: Jo nærmere alder, jo høyere similaritet (0-1 skala)")
print("• Bosted: 1 hvis samme sted, 0 hvis forskjellig")
print("• Yrke: 1 hvis samme yrke, 0 hvis forskjellig")
print("• Navnelikhet: 1 hvis samme etternavn, 0 hvis forskjellig")
print("• Kombinert: Vektet gjennomsnitt av alle mål")
print("")
print(" English:")
print("• Gender: 1 if same gender, 0 if different")
print("• Age: Closer age = higher similarity (0-1 scale)")
print("• Location: 1 if same place, 0 if different")
print("• Profession: 1 if same profession, 0 if different")
print("• Name similarity: 1 if same surname, 0 if different")
print("• Combined: Weighted average of all measures")
print("")

# Vis interessante observasjoner
print(" Interessante observasjoner / Interesting observations:")

```



```

print("")
print("  Norsk:")
print("• Erik og Arvid: Høy similaritet (samme kjønn, etternavn, nær alder)")
print("• Ingrid og Anna: Høy similaritet (samme kjønn, nær alder)")
print("• Lars og Kari: Høy similaritet (samme bosted, yrke, etternavn)")
print("• Erik og Ingrid: Lav similaritet (forskjellig kjønn, yrke)")
print("")
print("  English:")
print("• Erik and Arvid: High similarity (same gender, surname, close age)")
print("• Ingrid and Anna: High similarity (same gender, close age)")
print("• Lars and Kari: High similarity (same location, profession, surname)")
print("• Erik and Ingrid: Low similarity (different gender, profession)")
print("")

# Fiks spring layout for konsistent visualisering
print("  Visualisering med fiksert layout / Visualization with fixed layout:")
print("")

# Definer fiksert posisjonering
fixed_positions = {
    "Bestefar": (0, 2),      # Øverst i midten
    "Bestemor": (1, 2),     # Øverst til høyre
    "Far": (0, 1),          # Midten til venstre
    "Mor": (1, 1),          # Midten til høyre
    "Barn1": (0, 0),        # Nederst til venstre
    "Barn2": (1, 0)         # Nederst til høyre
}

# Visualiser med fiksert layout
plt.figure(figsize=(16, 10))

# Separer noder etter kjønn
male_nodes = [node for node in family_graph.nodes() if family_graph.
    ↪nodes[node]['gender'] == 'male']
female_nodes = [node for node in family_graph.nodes() if family_graph.
    ↪nodes[node]['gender'] == 'female']

# Beregn node-størrelser basert på alder (proporsjonal)
ages = [family_graph.nodes[node]['age'] for node in family_graph.nodes()]
min_age, max_age = min(ages), max(ages)
age_range = max_age - min_age if max_age - min_age > 0 else 1

# Minimum og maksimum node-størrelse
min_size, max_size = 800, 3000

# Tegn noder med farger basert på kjønn og størrelse basert på alder
for node in male_nodes:

```

```

age = family_graph.nodes[node]['age']
# Normaliser alder til størrelse (eldre = større)
normalized_age = (age - min_age) / age_range if age_range > 0 else 0.5
node_size = min_size + (max_size - min_size) * normalized_age

nx.draw_networkx_nodes(family_graph, fixed_positions, nodelist=[node],
                       node_color='blue', node_size=node_size, alpha=0.7)

for node in female_nodes:
    age = family_graph.nodes[node]['age']
    # Normaliser alder til størrelse (eldre = større)
    normalized_age = (age - min_age) / age_range if age_range > 0 else 0.5
    node_size = min_size + (max_size - min_size) * normalized_age

    nx.draw_networkx_nodes(family_graph, fixed_positions, nodelist=[node],
                           node_color='red', node_size=node_size, alpha=0.7)

# Tegn kanter med forskjellige farger
parent_child_edges = [(u, v) for u, v, d in family_graph.edges(data=True)
                      if d.get('relation') == 'parent-child']
marriage_edges = [(u, v) for u, v, d in family_graph.edges(data=True)
                  if d.get('relation') == 'marriage']

nx.draw_networkx_edges(family_graph, fixed_positions,
                       edgelist=parent_child_edges,
                       edge_color='black', arrows=True, arrowsize=20, width=2)
nx.draw_networkx_edges(family_graph, fixed_positions, edgelist=marriage_edges,
                       edge_color='gray', arrows=False, width=3, style='dashed')

# Legg til etiketter med fornavn
labels = {node: family_graph.nodes[node]['name'].split()[0] for node in
          family_graph.nodes()}
nx.draw_networkx_labels(family_graph, fixed_positions, labels, font_size=12,
                       font_weight='bold')

# Legg til node-attributter som tekstfelt
for node, (x, y) in fixed_positions.items():
    attrs = family_graph.nodes[node]
    # Opprett tekst med attributter
    attr_text = f"Kjønn: {attrs['gender']}\nAlder: {attrs['age']} år\nBosted: {attrs['location']}\nYrke: {attrs['profession']}"

    # Plasser tekstfelt enda nærmere noden
    plt.text(x + 0.08, y, attr_text, fontsize=8,
             bbox=dict(boxstyle="round,pad=0.3", facecolor='white', alpha=0.8,
                       edgecolor='gray'),
             verticalalignment='center')

```

```

plt.title("Slektstre med attributter / Family tree with attributes",
    ↪ fontsize=16, pad=20)

# Legg til legende utenfor grafen
legend_elements = [
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='blue',
    ↪ markersize=15, label='Menn / Men'),
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='red',
    ↪ markersize=15, label='Kvinner / Women'),
    plt.Line2D([0], [0], color='black', linewidth=2, label='Forelder-barn /
    ↪ Parent-child'),
    plt.Line2D([0], [0], color='gray', linewidth=3, linestyle='--',
    ↪ label='Ekteskap/Partnerskap / Marriage/Partnership')
]

# Plasser legende lenger til høyre for å unngå overlapping
plt.legend(handles=legend_elements, loc='center left', bbox_to_anchor=(1.1, 0.
    ↪ 5), fontsize=12)

# Juster layout for å gi mer plass til legende
plt.tight_layout()
plt.subplots_adjust(right=0.65)

plt.axis('off')
plt.show()

print("")
print(" Visualiserings-forklaring / Visualization explanation:")
print(" Blå noder: Menn (størrelse proporsjonal med alder)")
print(" Røde noder: Kvinner (størrelse proporsjonal med alder)")
print(" Svarte piler: Forelder-barn relasjoner")
print(" Grå stiplede linjer: Ekteskap/Partnerskap")
print(" Hvite tekstfelt: Node-attributter (kjønn, alder, bosted, yrke)")
print("")
print(" Visualization explanation:")
print(" Blue nodes: Men (size proportional to age)")
print(" Red nodes: Women (size proportional to age)")
print(" Black arrows: Parent-child relationships")
print(" Gray dashed lines: Marriage/Partnership")
print(" White text boxes: Node attributes (gender, age, location, profession)")

```

Similaritets-mål i slektstrær / Similarity measures in family trees

Norsk forklaring:

I slektstrær kan vi definere similaritets-mål basert på forskjellige attributter.

Dette er forskjellig fra avstands-matrisen som kun viser strukturell avstand.

Similaritets-mål kan være knyttet til kjønn, alder, bosted, yrke, navnelikhet, etc.

English explanation:

In family trees we can define similarity measures based on different attributes. This is different from the distance matrix which only shows structural distance. Similarity measures can be related to gender, age, location, profession, name similarity, etc.

Legger til attributter for similaritets-analyse:

Adding attributes for similarity analysis:

Attributter lagt til:

Erik: male, 75 år, Oslo, ingeniør

Ingrid: female, 72 år, Oslo, lærer

Arvid: male, 45 år, Bergen, programmerer

Anna: female, 42 år, Bergen, lege

Lars: male, 18 år, Bergen, student

Kari: female, 16 år, Bergen, student

Similaritets-matriser for forskjellige attributter:

Similarity matrices for different attributes:

Kjønn / Kjønn:

Kjønn:

Node	Erik	Ingrid	Arvid	Anna	Lars	Kari

Erik	1.00	0.00	1.00	0.00	1.00	0.00
Ingrid	0.00	1.00	0.00	1.00	0.00	1.00
Arvid	1.00	0.00	1.00	0.00	1.00	0.00
Anna	0.00	1.00	0.00	1.00	0.00	1.00
Lars	1.00	0.00	1.00	0.00	1.00	0.00
Kari	0.00	1.00	0.00	1.00	0.00	1.00

Alder / Alder:

Alder:

Node	Erik	Ingrid	Arvid	Anna	Lars	Kari

Erik	1.00	0.95	0.50	0.45	0.05	0.02
Ingrid	0.95	1.00	0.55	0.50	0.10	0.07
Arvid	0.50	0.55	1.00	0.95	0.55	0.52
Anna	0.45	0.50	0.95	1.00	0.60	0.57
Lars	0.05	0.10	0.55	0.60	1.00	0.97
Kari	0.02	0.07	0.52	0.57	0.97	1.00

Bosted / Bosted:

Bosted:

Node	Erik	Ingrid	Arvid	Anna	Lars	Kari

Erik	1.00	1.00	0.00	0.00	0.00	0.00
Ingrid	1.00	1.00	0.00	0.00	0.00	0.00
Arvid	0.00	0.00	1.00	1.00	1.00	1.00
Anna	0.00	0.00	1.00	1.00	1.00	1.00
Lars	0.00	0.00	1.00	1.00	1.00	1.00
Kari	0.00	0.00	1.00	1.00	1.00	1.00

Yrke / Yrke:
Yrke:

Node	Erik	Ingrid	Arvid	Anna	Lars	Kari

Erik	1.00	0.00	0.00	0.00	0.00	0.00
Ingrid	0.00	1.00	0.00	0.00	0.00	0.00
Arvid	0.00	0.00	1.00	0.00	0.00	0.00
Anna	0.00	0.00	0.00	1.00	0.00	0.00
Lars	0.00	0.00	0.00	0.00	1.00	1.00
Kari	0.00	0.00	0.00	0.00	1.00	1.00

Navnelikhet / Navnelikhet:
Navnelikhet:

Node	Erik	Ingrid	Arvid	Anna	Lars	Kari

Erik	1.00	0.00	1.00	0.00	1.00	1.00
Ingrid	0.00	1.00	0.00	0.00	0.00	0.00
Arvid	1.00	0.00	1.00	0.00	1.00	1.00
Anna	0.00	0.00	0.00	1.00	0.00	0.00
Lars	1.00	0.00	1.00	0.00	1.00	1.00
Kari	1.00	0.00	1.00	0.00	1.00	1.00

Kombinert similaritets-mål / Combined similarity measure:

Kombinert similaritet (vektet gjennomsnitt):
Combined similarity (weighted average):

Node	Erik	Ingrid	Arvid	Anna	Lars	Kari

Erik	1.00	0.48	0.55	0.13	0.42	0.21
Ingrid	0.48	1.00	0.17	0.35	0.03	0.22
Arvid	0.55	0.17	1.00	0.48	0.76	0.55
Anna	0.13	0.35	0.48	1.00	0.38	0.57
Lars	0.42	0.03	0.76	0.38	1.00	0.79

Kari	0.21	0.22	0.55	0.57	0.79	1.00
------	------	------	------	------	------	------

=====

FORKLARING AV SIMILARITETS-MÅL / EXPLANATION OF SIMILARITY MEASURES

=====

Norsk:

- Kjønn: 1 hvis samme kjønn, 0 hvis forskjellig
- Alder: Jo nærmere alder, jo høyere similaritet (0-1 skala)
- Bosted: 1 hvis samme sted, 0 hvis forskjellig
- Yrke: 1 hvis samme yrke, 0 hvis forskjellig
- Navnelikhet: 1 hvis samme etternavn, 0 hvis forskjellig
- Kombinert: Vektet gjennomsnitt av alle mål

English:

- Gender: 1 if same gender, 0 if different
- Age: Closer age = higher similarity (0-1 scale)
- Location: 1 if same place, 0 if different
- Profession: 1 if same profession, 0 if different
- Name similarity: 1 if same surname, 0 if different
- Combined: Weighted average of all measures

Interessante observasjoner / Interesting observations:

Norsk:

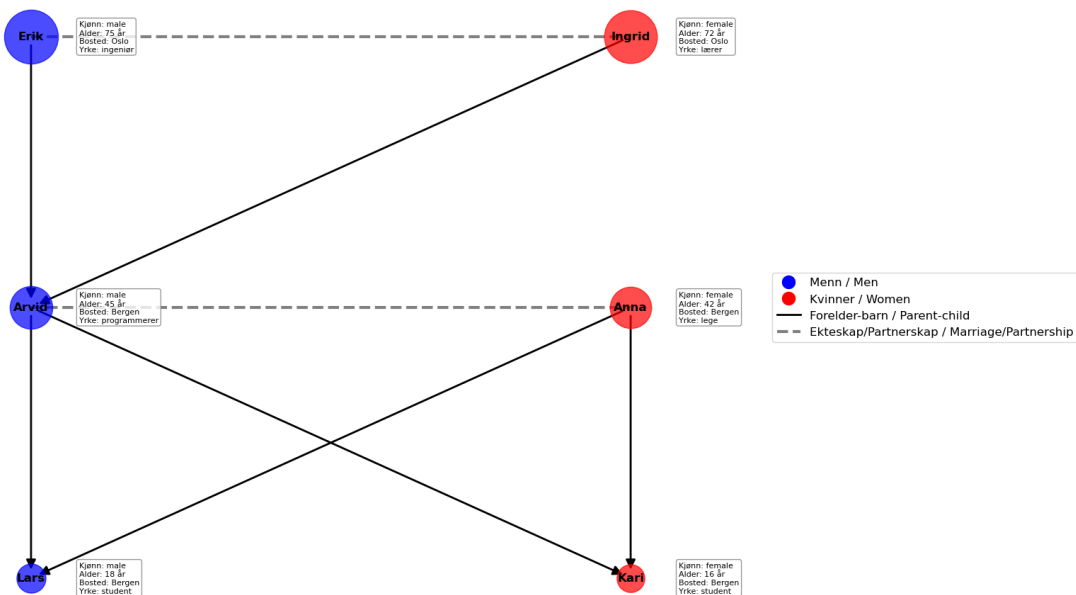
- Erik og Arvid: Høy similaritet (samme kjønn, etternavn, nær alder)
- Ingrid og Anna: Høy similaritet (samme kjønn, nær alder)
- Lars og Kari: Høy similaritet (samme bosted, yrke, etternavn)
- Erik og Ingrid: Lav similaritet (forskjellig kjønn, yrke)

English:

- Erik and Arvid: High similarity (same gender, surname, close age)
- Ingrid and Anna: High similarity (same gender, close age)
- Lars and Kari: High similarity (same location, profession, surname)
- Erik and Ingrid: Low similarity (different gender, profession)

Visualisering med fiksert layout / Visualization with fixed layout:

Slektstre med attributter / Family tree with attributes



Visualiserings-forklaring / Visualization explanation:

Blå noder: Menn (størrelse proporsjonal med alder)

Røde noder: Kvinner (størrelse proporsjonal med alder)

Svarte piler: Forelder-barn relasjoner

Grå stiplede linjer: Ekteskap/Partnerskap

Hvite tekstfelt: Node-attributter (kjønn, alder, bosted, yrke)

Visualization explanation:

Blue nodes: Men (size proportional to age)

Red nodes: Women (size proportional to age)

Black arrows: Parent-child relationships

Gray dashed lines: Marriage/Partnership

White text boxes: Node attributes (gender, age, location, profession)

```
[29]: # Visualisering av alle similaritets-matriser / Visualization of all similarity
      ↪matrices
print(" Visualisering av similaritets-matriser / Visualization of similarity
      ↪matrices")
print("="*80)
print("")

import matplotlib.pyplot as plt
import numpy as np
```

```

# Sett opp figuren med subplots for alle matriser
fig, axes = plt.subplots(2, 3, figsize=(18, 12))
fig.suptitle('Similaritets-matriser / Similarity Matrices', fontsize=16,
    ↪fontweight='bold')

# Hent node-navn (fornavn)
node_names = [family_graph.nodes[node]['name'].split()[0] for node in
    ↪family_graph.nodes()]
nodes = list(family_graph.nodes())

# Definer similaritets-funksjoner (samme som tidligere)
def gender_similarity(person1, person2):
    return 1 if person1["gender"] == person2["gender"] else 0

def age_similarity(person1, person2):
    age_diff = abs(person1["age"] - person2["age"])
    max_age_diff = 60
    return max(0, 1 - (age_diff / max_age_diff))

def location_similarity(person1, person2):
    return 1 if person1["location"] == person2["location"] else 0

def profession_similarity(person1, person2):
    return 1 if person1["profession"] == person2["profession"] else 0

def name_similarity(person1, person2):
    name1_parts = person1["name"].split()
    name2_parts = person2["name"].split()
    return 1 if name1_parts[-1] == name2_parts[-1] else 0

def combined_similarity(person1, person2, weights=None):
    if weights is None:
        weights = {"Kjønn": 0.2, "Alder": 0.3, "Bosted": 0.2, "Yrke": 0.1,
    ↪"Navnelikhet": 0.2}

    similarity_measures = {
        "Kjønn": gender_similarity,
        "Alder": age_similarity,
        "Bosted": location_similarity,
        "Yrke": profession_similarity,
        "Navnelikhet": name_similarity
    }

    total_similarity = 0
    for measure_name, weight in weights.items():
        similarity_func = similarity_measures[measure_name]
        similarity = similarity_func(person1, person2)

```



```

        total_similarity += weight * similarity

    return total_similarity

# Lag similaritets-matriser
similarity_matrices = {}
similarity_names = ["Kjønn", "Alder", "Bosted", "Yrke", "Navnelikhet",
                    ↪ "Kombinert"]

for i, measure_name in enumerate(similarity_names):
    matrix = np.zeros((len(nodes), len(nodes)))

    for j, node1 in enumerate(nodes):
        for k, node2 in enumerate(nodes):
            if j == k:
                matrix[j, k] = 1.0 # Perfekt similaritet med seg selv
            else:
                person1 = {attr: family_graph.nodes[node1][attr] for attr in
                    ↪ ["gender", "age", "location", "profession", "name"]}
                person2 = {attr: family_graph.nodes[node2][attr] for attr in
                    ↪ ["gender", "age", "location", "profession", "name"]}

                if measure_name == "Kjønn":
                    matrix[j, k] = gender_similarity(person1, person2)
                elif measure_name == "Alder":
                    matrix[j, k] = age_similarity(person1, person2)
                elif measure_name == "Bosted":
                    matrix[j, k] = location_similarity(person1, person2)
                elif measure_name == "Yrke":
                    matrix[j, k] = profession_similarity(person1, person2)
                elif measure_name == "Navnelikhet":
                    matrix[j, k] = name_similarity(person1, person2)
                elif measure_name == "Kombinert":
                    matrix[j, k] = combined_similarity(person1, person2)

    similarity_matrices[measure_name] = matrix

# Visualiser hver matrise
positions = [(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2)]
colors = ['Blues', 'Greens', 'Oranges', 'Purples', 'Reds', 'viridis']

for i, (measure_name, matrix) in enumerate(similarity_matrices.items()):
    row, col = positions[i]
    ax = axes[row, col]

    # Lag heatmap
    im = ax.imshow(matrix, cmap=colors[i], vmin=0, vmax=1)

```

```

# Sett labels
ax.set_xticks(range(len(node_names)))
ax.set_yticks(range(len(node_names)))
ax.set_xticklabels(node_names, rotation=45, ha='right')
ax.set_yticklabels(node_names)

# Legg til verdier i cellene
for j in range(len(node_names)):
    for k in range(len(node_names)):
        text = ax.text(k, j, f'{matrix[j, k]:.2f}',
                        ha="center", va="center", color="black" if matrix[j, k] > 0.5 else "white",
                        fontsize=8, fontweight='bold')

# Sett tittel
ax.set_title(f'{measure_name} / {measure_name}', fontweight='bold')

# Legg til colorbar
plt.colorbar(im, ax=ax, shrink=0.8)

plt.tight_layout()
plt.show()

# Analyser og sammenlign matrisene
print(" ANALYSE AV SIMILARITETS-MATRISER / ANALYSIS OF SIMILARITY MATRICES")
print("="*80)
print("")

# Finn høyeste og laveste similariteter
for measure_name, matrix in similarity_matrices.items():
    # Fjern diagonalen (selv-similaritet)
    mask = np.eye(matrix.shape[0], dtype=bool)
    off_diagonal = matrix[~mask]

    max_sim = np.max(off_diagonal)
    min_sim = np.min(off_diagonal)
    mean_sim = np.mean(off_diagonal)

    print(f" {measure_name}:")
    print(f"   Høyeste similaritet: {max_sim:.3f}")
    print(f"   Laveste similaritet: {min_sim:.3f}")
    print(f"   Gjennomsnitt: {mean_sim:.3f}")

# Finn par med høyeste similaritet
max_indices = np.where(matrix == max_sim)
for i, j in zip(max_indices[0], max_indices[1]):

```

```

        if i != j: # Ikke diagonal
            print(f"    Høyeste: {node_names[i]}    {node_names[j]} ({max_sim:.
↪3f})")
            print("")

# Sammenlign matrisene
print(" SAMMENLIGNING AV MATRISER / MATRIX COMPARISON")
print("="*80)
print("")

# Korrelasjon mellom matrisene
correlation_matrix = np.zeros((len(similarity_names), len(similarity_names)))
for i, name1 in enumerate(similarity_names):
    for j, name2 in enumerate(similarity_names):
        if i == j:
            correlation_matrix[i, j] = 1.0
        else:
            # Flatten matrisene og beregn korrelasjon
            flat1 = similarity_matrices[name1].flatten()
            flat2 = similarity_matrices[name2].flatten()
            correlation = np.corrcoef(flat1, flat2)[0, 1]
            correlation_matrix[i, j] = correlation

# Visualiser korrelasjonsmatrisen
plt.figure(figsize=(10, 8))
im = plt.imshow(correlation_matrix, cmap='coolwarm', vmin=-1, vmax=1)

# Sett labels
plt.xticks(range(len(similarity_names)), similarity_names, rotation=45)
plt.yticks(range(len(similarity_names)), similarity_names)

# Legg til verdier i cellene
for i in range(len(similarity_names)):
    for j in range(len(similarity_names)):
        text = plt.text(j, i, f'{correlation_matrix[i, j]:.3f}',
                        ha="center", va="center",
                        color="white" if abs(correlation_matrix[i, j]) > 0.5
↪else "black",
                        fontsize=10, fontweight='bold')

plt.title('Korrelasjon mellom similaritets-matriser / Correlation between
↪similarity matrices')
plt.colorbar(im, label='Korrelasjon / Correlation')
plt.tight_layout()
plt.show()

# Identifiser interessante mønstre

```

```

print("  INTERESSANTE MØNSTRE / INTERESTING PATTERNS")
print("="*80)
print("")

# Finn par med høyest kombinert similaritet
combined_matrix = similarity_matrices["Kombinert"]
mask = np.eye(combined_matrix.shape[0], dtype=bool)
off_diagonal = combined_matrix[~mask]
max_combined = np.max(off_diagonal)

max_indices = np.where(combined_matrix == max_combined)
for i, j in zip(max_indices[0], max_indices[1]):
    if i != j:
        print(f"  Høyest kombinert similaritet: {node_names[i]}  ␣
↳{node_names[j]} ({max_combined:.3f})")

        # Analyser hvorfor de er like
        person1 = {attr: family_graph.nodes[nodes[i]][attr] for attr in␣
↳["gender", "age", "location", "profession", "name"]}
        person2 = {attr: family_graph.nodes[nodes[j]][attr] for attr in␣
↳["gender", "age", "location", "profession", "name"]}

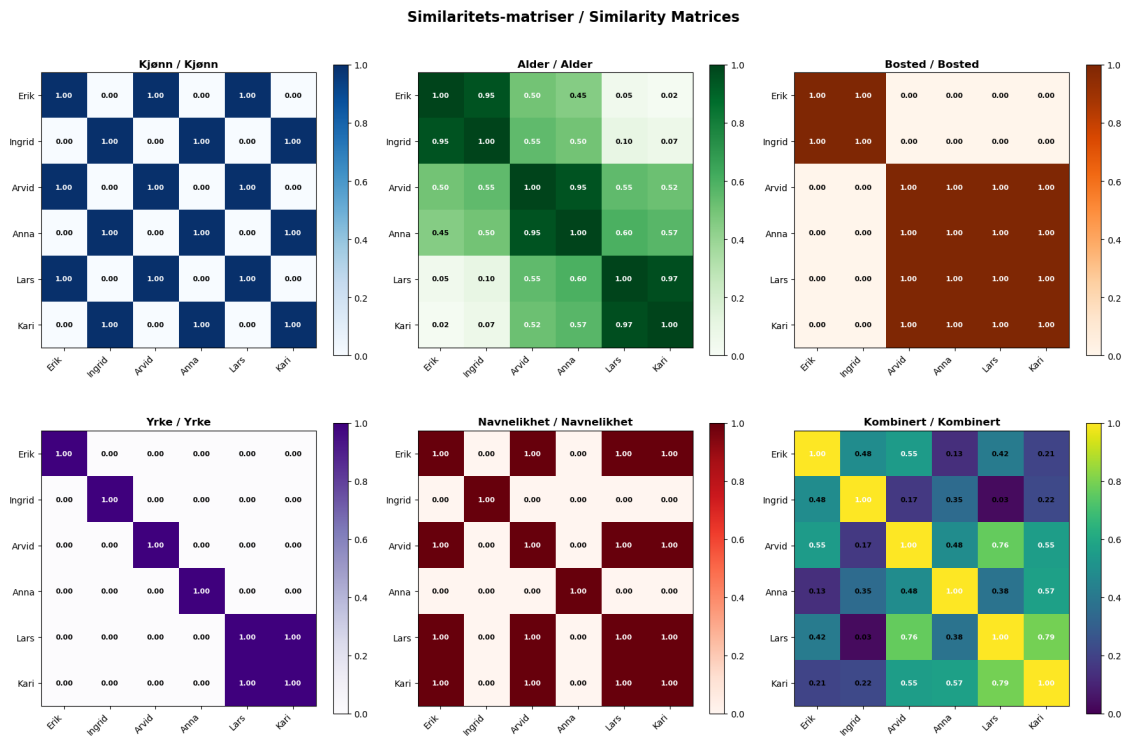
        print(f"    Kjønn: {person1['gender']} vs {person2['gender']}␣
↳(similaritet: {gender_similarity(person1, person2):.1f})")
        print(f"    Alder: {person1['age']} vs {person2['age']} (similaritet:␣
↳{age_similarity(person1, person2):.3f})")
        print(f"    Bosted: {person1['location']} vs {person2['location']}␣
↳(similaritet: {location_similarity(person1, person2):.1f})")
        print(f"    Yrke: {person1['profession']} vs {person2['profession']}␣
↳(similaritet: {profession_similarity(person1, person2):.1f})")
        print(f"    Navn: {person1['name']} vs {person2['name']} (similaritet:␣
↳{name_similarity(person1, person2):.1f})")
        print("")

print("  Visualisering fullført!")
print("  Visualization completed!")
print("")
print("  INSIGHTS / INNSIKTER:")
print("• Heatmaps viser similaritets-mønstre visuelt")
print("• Korrelasjonsmatrise viser sammenhenger mellom attributter")
print("• Kombinert similaritet gir helhetsbilde av likhet")
print("• Farger indikerer styrke: mørk = høy similaritet, lys = lav␣
↳similaritet")
print("")
print("  INSIGHTS:")
print("• Heatmaps show similarity patterns visually")

```

```
print("• Correlation matrix shows relationships between attributes")
print("• Combined similarity gives holistic view of similarity")
print("• Colors indicate strength: dark = high similarity, light = low
↪similarity")
```

Visualisering av similaritets-matriser / Visualization of similarity matrices



ANALYSE AV SIMILARITETS-MATRISER / ANALYSIS OF SIMILARITY MATRICES

Kjønn:

Høyeste similaritet: 1.000

Laveste similaritet: 0.000

Gjennomsnitt: 0.400

Høyeste: Erik Arvid (1.000)

Høyeste: Erik Lars (1.000)

Høyeste: Ingrid Anna (1.000)

Høyeste: Ingrid Kari (1.000)

Høyeste: Arvid Erik (1.000)

Høyeste: Arvid Lars (1.000)

Høyeste: Anna Ingrid (1.000)

Høyeste: Anna Kari (1.000)

Høyeste: Lars Erik (1.000)
Høyeste: Lars Arvid (1.000)
Høyeste: Kari Ingrid (1.000)
Høyeste: Kari Anna (1.000)

Alder:

Høyeste similaritet: 0.967
Laveste similaritet: 0.017
Gjennomsnitt: 0.489
Høyeste: Lars Kari (0.967)
Høyeste: Kari Lars (0.967)

Bosted:

Høyeste similaritet: 1.000
Laveste similaritet: 0.000
Gjennomsnitt: 0.467
Høyeste: Erik Ingrid (1.000)
Høyeste: Ingrid Erik (1.000)
Høyeste: Arvid Anna (1.000)
Høyeste: Arvid Lars (1.000)
Høyeste: Arvid Kari (1.000)
Høyeste: Anna Arvid (1.000)
Høyeste: Anna Lars (1.000)
Høyeste: Anna Kari (1.000)
Høyeste: Lars Arvid (1.000)
Høyeste: Lars Anna (1.000)
Høyeste: Lars Kari (1.000)
Høyeste: Kari Arvid (1.000)
Høyeste: Kari Anna (1.000)
Høyeste: Kari Lars (1.000)

Yrke:

Høyeste similaritet: 1.000
Laveste similaritet: 0.000
Gjennomsnitt: 0.067
Høyeste: Lars Kari (1.000)
Høyeste: Kari Lars (1.000)

Navnelikhet:

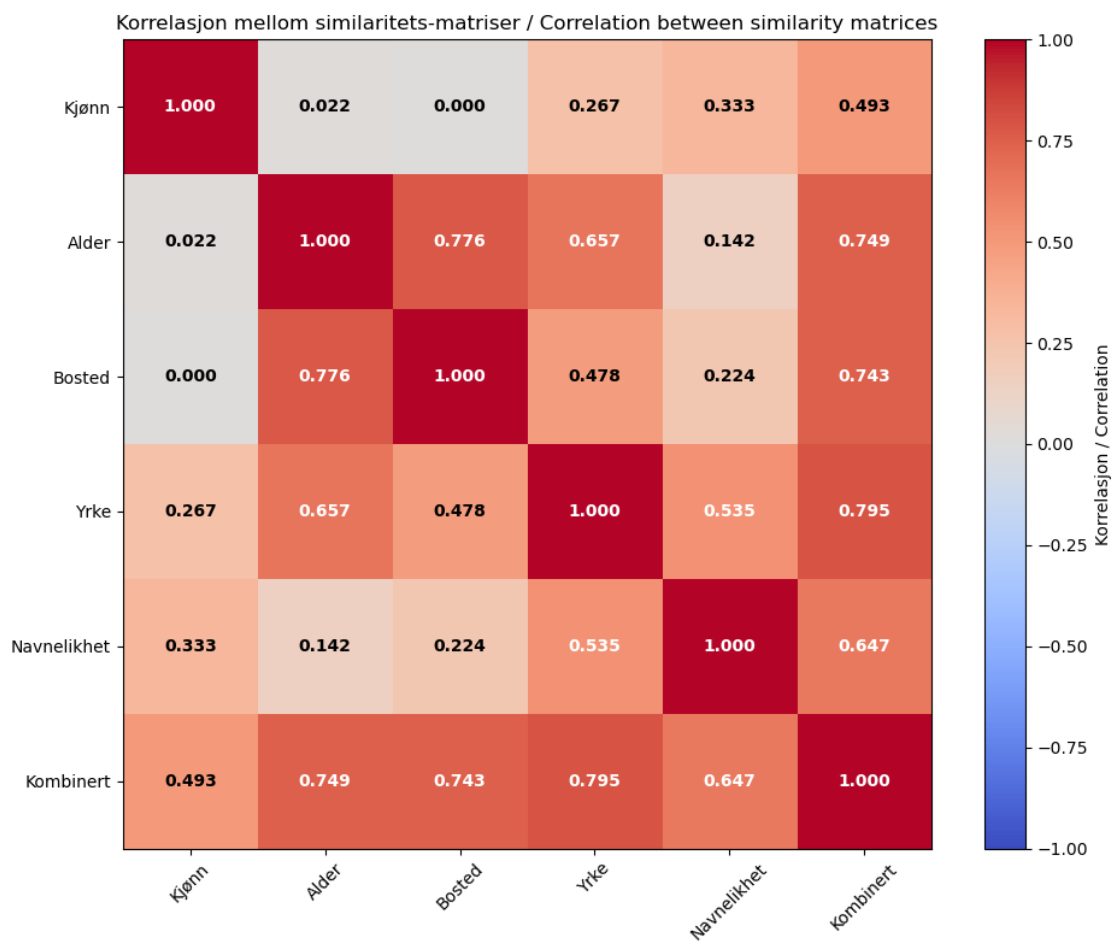
Høyeste similaritet: 1.000
Laveste similaritet: 0.000
Gjennomsnitt: 0.400
Høyeste: Erik Arvid (1.000)
Høyeste: Erik Lars (1.000)
Høyeste: Erik Kari (1.000)
Høyeste: Arvid Erik (1.000)
Høyeste: Arvid Lars (1.000)
Høyeste: Arvid Kari (1.000)

Høyeste: Lars Erik (1.000)
Høyeste: Lars Arvid (1.000)
Høyeste: Lars Kari (1.000)
Høyeste: Kari Erik (1.000)
Høyeste: Kari Arvid (1.000)
Høyeste: Kari Lars (1.000)

Kombinert:

Høyeste similaritet: 0.790
Laveste similaritet: 0.030
Gjennomsnitt: 0.407
Høyeste: Lars Kari (0.790)
Høyeste: Kari Lars (0.790)

SAMMENLIGNING AV MATRISER / MATRIX COMPARISON



INTERESSANTE MØNSTRE / INTERESTING PATTERNS

Høyest kombinert similaritet: Lars Kari (0.790)
Kjønn: male vs female (similaritet: 0.0)
Alder: 18 vs 16 (similaritet: 0.967)
Bosted: Bergen vs Bergen (similaritet: 1.0)
Yrke: student vs student (similaritet: 1.0)
Navn: Lars Lundervold vs Kari Lundervold (similaritet: 1.0)

Høyest kombinert similaritet: Kari Lars (0.790)
Kjønn: female vs male (similaritet: 0.0)
Alder: 16 vs 18 (similaritet: 0.967)
Bosted: Bergen vs Bergen (similaritet: 1.0)
Yrke: student vs student (similaritet: 1.0)
Navn: Kari Lundervold vs Lars Lundervold (similaritet: 1.0)

Visualisering fullført!
Visualization completed!

INSIGHTS / INNSIKTER:

- Heatmaps viser similaritets-mønstre visuelt
- Korrelasjonsmatrise viser sammenhenger mellom attributter
- Kombinert similaritet gir helhetsbilde av likhet
- Farger indikerer styrke: mørk = høy similaritet, lys = lav similaritet

INSIGHTS:

- Heatmaps show similarity patterns visually
- Correlation matrix shows relationships between attributes
- Combined similarity gives holistic view of similarity
- Colors indicate strength: dark = high similarity, light = low similarity

2.13 Del 4: Praktiske øvelser / Part 4: Practical Exercises

2.13.1 Refleksjonsoppgaver / Reflection Questions

- 1. Slektskap og grafer / Kinship and graphs** - Hvilke typer grafer vil du bruke for å representere forskjellige slektsrelasjoner? - What types of graphs would you use to represent different family relationships?
- 2. Sykler i slektstrær / Cycles in family trees** - Kan du tenke deg situasjoner hvor slektstrær kunne få sykler? - Can you think of situations where family trees could have cycles?
- 3. Kompleksitet / Complexity** - Hvorfor blir slektstrær mer komplekse jo lenger tilbake i tid du går? - Why do family trees become more complex the further back in time you go?

2.13.2 Programmeringsoppgaver / Programming Exercises

Oppgave 1: Bygg et 3-personers slektstre / Exercise 1: Build a 3-person family tree

Bygg et enkelt slektstre med 3 personer og visualiser det.

Build a simple family tree with 3 people and visualize it.

```
[30]: # LØSNING / SOLUTION
print("  Løsning til Oppgave 1 / Solution to Exercise 1")
print("  Solution to Exercise 1")

# Opprett et tomt slektstre
exercise_tree = nx.DiGraph()

# Legg til 3 personer med norske navn
exercise_tree.add_node("Forelder", name="Forelder Person", role="parent")
exercise_tree.add_node("Barn1", name="Første Barn", role="child")
exercise_tree.add_node("Barn2", name="Andre Barn", role="child")

# Legg til relasjoner (kun forelder-barn relasjoner)
exercise_tree.add_edge("Forelder", "Barn1", relation="parent-child")
exercise_tree.add_edge("Forelder", "Barn2", relation="parent-child")
# Fjernet: exercise_tree.add_edge("Barn1", "Barn2", relation="siblings")
# Søsken er koblet gjennom foreldre, ikke direkte til hverandre

# Visualiser med fast seed for konsistent plassering
plt.figure(figsize=(10, 6))
# Bruk fast seed for konsistent node-plassering
pos = nx.spring_layout(exercise_tree, k=2, seed=42)

# Tegn noder med forskjellige farger basert på rolle
node_colors = ['lightgreen' if exercise_tree.nodes[node]['role'] == 'parent'
               else 'lightblue' for node in exercise_tree.nodes()]

nx.draw(exercise_tree, pos, with_labels=True, node_color=node_colors,
        node_size=2000, font_size=12, font_weight='bold',
        arrows=True, arrowsize=20)

plt.title("3-personers slektstre / 3-person family tree")
plt.show()

print(f"Antall personer: {exercise_tree.number_of_nodes()}")
print(f"Antall relasjoner: {exercise_tree.number_of_edges()}")
print(f"Number of people: {exercise_tree.number_of_nodes()}")
print(f"Number of relationships: {exercise_tree.number_of_edges()}")

# Vis slektsrelasjoner
print(f"\nSlektsrelasjoner:")
```

```

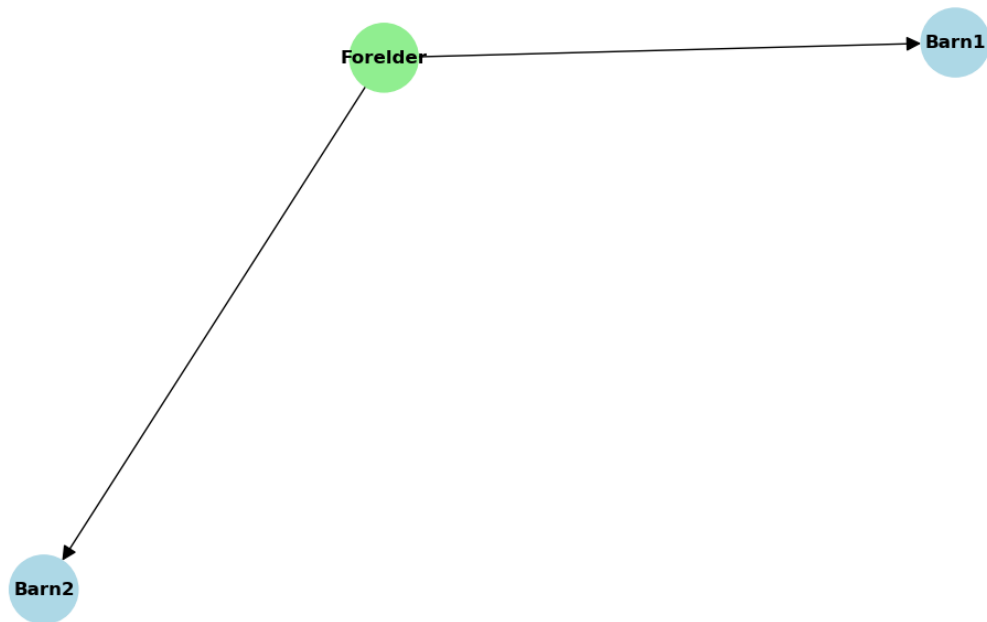
print(f"Family relationships:")
for u, v, data in exercise_tree.edges(data=True):
    print(f"    {exercise_tree.nodes[u]['name']} → {exercise_tree.
↳nodes[v]['name']} ({data['relation']})")

# Vis søsken-relasjon (implisitt)
print(f"\nSøsken-relasjon (implisitt):")
print(f"Sibling relationship (implicit):")
print(f"    {exercise_tree.nodes['Barn1']['name']} og {exercise_tree.
↳nodes['Barn2']['name']} er søsken")
print(f"    {exercise_tree.nodes['Barn1']['name']} and {exercise_tree.
↳nodes['Barn2']['name']} are siblings")

```

Løsning til Oppgave 1 / Solution to Exercise 1
 Solution to Exercise 1

3-personers slektstre / 3-person family tree



Antall personer: 3
 Antall relasjoner: 2
 Number of people: 3
 Number of relationships: 2

Slektsrelasjoner:

Family relationships:

Forelder Person → Første Barn (parent-child)
 Forelder Person → Andre Barn (parent-child)

Søsken-relasjon (implisitt):

Sibling relationship (implicit):

Første Barn og Andre Barn er søsken

Første Barn and Andre Barn are siblings

Oppgave 2: Finn stier mellom familiemedlemmer / Exercise 2: Find paths between family members

Bruk NetworkX til å finne alle mulige stier mellom to personer i et slektstre.

Use NetworkX to find all possible paths between two people in a family tree.

```
[31]: # LØSNING / SOLUTION
print("  Løsning til Oppgave 2 / Solution to Exercise 2")
print("  Solution to Exercise 2")

def find_all_paths(graph, start, end, max_length=5):
    """Finn alle stier mellom to noder i en retnet graf."""
    try:
        # Bruk NetworkX for å finne alle enkle stier
        paths = list(nx.all_simple_paths(graph, start, end, cutoff=max_length))
        return paths
    except nx.NetworkXNoPath:
        return []

# Test med vårt familiegraf
start_person = "Bestefar"
end_person = "Barn1"

paths = find_all_paths(family_graph, start_person, end_person)

print(f"Stier fra {family_graph.nodes[start_person]['name']} til {family_graph.
↪nodes[end_person]['name']}:")
print(f"Paths from {family_graph.nodes[start_person]['name']} to {family_graph.
↪nodes[end_person]['name']}:")

for i, path in enumerate(paths, 1):
    path_names = [family_graph.nodes[node]['name'] for node in path]
    print(f"  {i}. {' → '.join(path_names)}")

if not paths:
    print("  Ingen stier funnet / No paths found")

# Finn også korteste sti
try:
    shortest_path = nx.shortest_path(family_graph, start_person, end_person)
    shortest_names = [family_graph.nodes[node]['name'] for node in
↪shortest_path]
```

```

    print(f"\nKorteste sti: {' → '.join(shortest_names)}")
    print(f"Shortest path: {' → '.join(shortest_names)}")
except nx.NetworkXNoPath:
    print("\nIngen sti funnet / No path found")

```

Løsning til Oppgave 2 / Solution to Exercise 2

Solution to Exercise 2

Stier fra Erik Lundervold til Lars Lundervold:

Paths from Erik Lundervold to Lars Lundervold:

1. Erik Lundervold → Arvid Lundervold → Lars Lundervold
2. Erik Lundervold → Arvid Lundervold → Anna Pedersen → Lars Lundervold
3. Erik Lundervold → Ingrid Hansen → Arvid Lundervold → Lars Lundervold
4. Erik Lundervold → Ingrid Hansen → Arvid Lundervold → Anna Pedersen → Lars Lundervold

Korteste sti: Erik Lundervold → Arvid Lundervold → Lars Lundervold

Shortest path: Erik Lundervold → Arvid Lundervold → Lars Lundervold

2.14 Del 5: Praktiske anvendelser / Part 5: Practical Applications

2.14.1 Hvordan slektstre-prosjektet bruker NetworkX / How the family tree project uses NetworkX

La oss se på hvordan det ekte slektstre-prosjektet bruker NetworkX:

Let's see how the real family tree project uses NetworkX:

```

[32]: # Last inn eksempel-familien og vis hvordan Slekststre-klassen bruker NetworkX
print("  Hvordan Slekststre-klassen bruker NetworkX")
print("  How the Slekststre class uses NetworkX")

# Last inn eksempel-familien
from family_io import load_from_yaml

familie_data = load_from_yaml('../data/eksempel_familie.yaml')
slekststre = Slekststre(familie_data)

print(f"\nFamilie lastet med {len(familie_data.personer)} personer og
↪ {len(familie_data.ekteskap)} ekteskap")
print(f"Family loaded with {len(familie_data.personer)} people and
↪ {len(familie_data.ekteskap)} marriages")

# Vis hvordan grafen brukes internt
print(f"\nInternt graf-objekt / Internal graph object:")
print(f"  Type: {type(slekststre.graph)}")
print(f"  Antall noder: {slekststre.graph.number_of_nodes()}")
print(f"  Number of nodes: {slekststre.graph.number_of_nodes()}")

```

```

print(f"  Antall kanter: {slektstre.graph.number_of_edges()}")
print(f"  Number of edges: {slektstre.graph.number_of_edges()}")

# Vis nodetyper
person_nodes = [node for node, data in slektstre.graph.nodes(data=True)
                 if data.get('type') == 'person']
marriage_nodes = [node for node, data in slektstre.graph.nodes(data=True)
                  if data.get('type') == 'marriage']

print(f"\nNodetyper / Node types:")
print(f"  Personer: {len(person_nodes)}")
print(f"  People: {len(person_nodes)}")
print(f"  Ekteskap: {len(marriage_nodes)}")
print(f"  Marriages: {len(marriage_nodes)}")

# Vis kanttyper
edge_types = {}
for u, v, data in slektstre.graph.edges(data=True):
    relation = data.get('relation', 'unknown')
    edge_types[relation] = edge_types.get(relation, 0) + 1

print(f"\nKanttyper / Edge types:")
for relation, count in edge_types.items():
    print(f"  {relation}: {count}")

# Vis statistikk
print(f"\n  Statistikk / Statistics:")
stats = slektstre.get_statistics()
print(f"  Totalt antall personer: {stats['total_persons']}")
print(f"  Total number of people: {stats['total_persons']}")
print(f"  Antall generasjoner: {stats['max_generation'] + 1}")
print(f"  Number of generations: {stats['max_generation'] + 1}")
print(f"  Gjennomsnittsalder: {stats['average_age']} år")
print(f"  Average age: {stats['average_age']} years")

```

Hvordan Slektstre-klassen bruker NetworkX

How the Slektstre class uses NetworkX

Familie lastet med 17 personer og 5 ekteskap

Family loaded with 17 people and 5 marriages

Internt graf-objekt / Internal graph object:

Type: <class 'networkx.classes.digraph.DiGraph'>

Antall noder: 22

Number of nodes: 22

Antall kanter: 46

Number of edges: 46

Nodetyper / Node types:

Personer: 17
People: 17
Ekteskap: 5
Marriages: 5

Kanttyper / Edge types:

partner: 20
parent-child: 26

Statistikk / Statistics:

Totalt antall personer: 17
Total number of people: 17
Antall generasjoner: 3
Number of generations: 3
Gjennomsnittsalder: 49.2 år
Average age: 49.2 years

2.15 Del 6: Oppsummering og neste steg / Part 6: Summary and Next Steps

2.15.1 Hva har du lært? / What have you learned?

I denne notebooken har du lært:

In this notebook you have learned:

1. **Slektstrær:** Hva de er, hvorfor de er viktige, og forskjellige typer
2. **Grafteori:** Grunnleggende konsepter som noder, kanter, og treer
3. **Kobling:** Hvordan slektstrær kan representeres som grafer
4. **Praksis:** Hvordan bruke NetworkX til å bygge og analysere slektstrær
5. **Øvelser:** Både refleksjon og programmering for å forstå begrepene

English: 1. **Family trees:** What they are, why they're important, and different types 2. **Graph theory:** Basic concepts like nodes, edges, and trees 3. **Connection:** How family trees can be represented as graphs 4. **Practice:** How to use NetworkX to build and analyze family trees 5. **Exercises:** Both reflection and programming to understand the concepts

2.15.2 Ordliste / Vocabulary

Norsk	English	Grafteori	Forklaring
Slektstre	Family tree	Tree	Spesiell type graf uten sykler
Person	Person	Node/Vertex	Et punkt i grafen
Slektsrelasjon	Family relationship	Edge	Linje mellom to noder
Forelder	Parent	-	Person som har barn
Barn	Child	-	Person som har foreldre
Generasjon	Generation	Distance	Avstand fra rot
Slektskap	Kinship	Path	Sti mellom to noder
Sykel	Cycle	Cycle	Sti som går i sirkel

Norsk	English	Grafteori	Forklaring
Sammenheng	Connectivity	Connected	Alle noder kan nås
Komponent	Component	Component	Sammenhengende del av graf

2.15.3 Neste steg / Next Steps

Nå som du har lært grunnleggende konsepter, kan du gå videre til:

Now that you've learned the basic concepts, you can move on to:

1. [01_introduksjon.ipynb](#) - Oversikt over slektstre-prosjektet
2. [02_bygg_tre_manuelt.ipynb](#) - Bygge slektstrær programmatisk
3. [03_importer_data.ipynb](#) - Import og eksport av data
4. [04_visualisering.ipynb](#) - Avanserte visualiseringer
5. [05_eksterne_databaser.ipynb](#) - Integrasjon med eksterne databaser

2.15.4 Videre lesing / Further Reading

Grafteori / Graph Theory: - [NetworkX Documentation](#) - [Introduction to Graph Theory](#) - [Graph Theory Tutorial](#)

Genealogi / Genealogy: - [FamilySearch](#) - Verdens største genealogi-database - [Digitalarkivet](#) - Norske historiske kilder - [Ancestry.com](#) - Kommersiell genealogi-tjeneste

2.15.5 Takk for at du leste! / Thank you for reading!

Vi håper denne introduksjonen har gitt deg en god forståelse av både slektstrær og grafteori, og hvordan de kobles sammen i dette prosjektet.

We hope this introduction has given you a good understanding of both family trees and graph theory, and how they are connected in this project.

Lykke til med ditt eget slektstre-prosjekt!

Good luck with your own family tree project!

Denne notebooken er en del av slektstre-prosjektet. Se [README.md](#) for mer informasjon.

This notebook is part of the family tree project. See [README.md](#) for more information.

01_introduksjon

October 12, 2025

```
[1]: # =====  
# GOOGLE COLAB SETUP / GOOGLE COLAB SETUP  
# =====  
  
# Sjekk om vi kjører i Google Colab  
try:  
    import google.colab  
    IN_COLAB = True  
    print(" Kjører i Google Colab - installerer avhengigheter...")  
    print(" Running in Google Colab - installing dependencies...")  
  
    # Installer nødvendige pakker  
    import subprocess  
    import sys  
    try:  
        subprocess.check_call([sys.executable, "-m", "pip", "install", "-q",  
                                "networkx", "matplotlib", "plotly", "pydantic",  
                                "pyyaml", "pandas", "ipywidgets", "pillow",  
↪ "kaleido"])  
        print(" Pakker installert")  
    except Exception as e:  
        print(f" Pip install feilet: {e}")  
  
    # Fjern eksisterende slektstre-mappe hvis den finnes  
    import shutil  
    import os  
    if os.path.exists('/content/slektstre'):  
        shutil.rmtree('/content/slektstre')  
        print(" Fjernet eksisterende slektstre-mappe")  
  
    # Klon repository  
    try:  
        subprocess.check_call(['git', 'clone', 'https://github.com/arvidl/  
↪ slektstre.git'])  
        print(" Repository klonet")  
    except Exception as e:  
        print(f" Git clone feilet: {e}")
```



```

# Legg til src-mappen til Python path og importer direkte
sys.path.insert(0, '/content/slektstre/src')
print(" Path lagt til")

# Importer slektstre-modulene direkte for å unngå navnekonflikt
import importlib.util
import types

# Først, fjern konfliktende moduler fra sys.modules
modules_to_remove = ['tree', 'models', 'localization']
for module_name in modules_to_remove:
    if module_name in sys.modules:
        del sys.modules[module_name]

# Last inn models.py først
try:
    spec = importlib.util.spec_from_file_location("slektstre_models", "/
↪content/slektstre/src/models.py")
    slektstre_models = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_models)

    # Opprett midlertidig models modul
    temp_models_module = types.ModuleType('models')
    temp_models_module.Person = slektstre_models.Person
    temp_models_module.Gender = slektstre_models.Gender
    temp_models_module.Ekteskap = slektstre_models.Ekteskap
    temp_models_module.FamilieData = slektstre_models.FamilieData
    sys.modules['models'] = temp_models_module

    print(" models.py lastet")
except Exception as e:
    print(f" models.py feilet: {e}")

# Last inn localization.py
try:
    spec = importlib.util.spec_from_file_location("slektstre_localization",
↪"/content/slektstre/src/localization.py")
    slektstre_localization = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_localization)

    # Opprett midlertidig localization modul
    temp_localization_module = types.ModuleType('localization')
    temp_localization_module.t = slektstre_localization.t
    sys.modules['localization'] = temp_localization_module

    print(" localization.py lastet")

```

```

except Exception as e:
    print(f" localization.py feilet: {e}")

# Last inn tree.py som slektstre_tree
try:
    spec = importlib.util.spec_from_file_location("slektstre_tree", "/
↪content/slektstre/src/tree.py")
    slektstre_tree = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_tree)

    # Opprett midlertidig tree modul
    temp_tree_module = types.ModuleType('tree')
    temp_tree_module.Slektstre = slektstre_tree.Slektstre
    sys.modules['tree'] = temp_tree_module

    print(" tree.py lastet")
except Exception as e:
    print(f" tree.py feilet: {e}")

# Last inn family_io.py
try:
    spec = importlib.util.spec_from_file_location("slektstre_io", "/content/
↪slektstre/src/family_io.py")
    slektstre_io = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_io)
    print(" family_io.py lastet")
except Exception as e:
    print(f" family_io.py feilet: {e}")

# Last inn visualization.py
try:
    spec = importlib.util.spec_from_file_location("slektstre_viz", "/
↪content/slektstre/src/visualization.py")
    slektstre_viz = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_viz)
    print(" visualization.py lastet")
except Exception as e:
    print(f" visualization.py feilet: {e}")

print(" Slektstre-moduler lastet inn i Colab")

except ImportError:
    IN_COLAB = False
    print(" Kjører lokalt / Running locally")
    import sys
    sys.path.append('../src')
except Exception as e:

```

```

print(f" Colab setup feilet: {e}")
IN_COLAB = False
print(" Fallback til lokal modus / Fallback to local mode")
import sys
sys.path.append('../src')

print(f" Miljø: {'Google Colab' if IN_COLAB else 'Lokal'}")
print(f" Environment: {'Google Colab' if IN_COLAB else 'Local'}")

```

Kjører lokalt / Running locally
Miljø: Lokal
Environment: Local

1 Slektstre med NetworkX - Introduksjon

Velkommen til slektstre-prosjektet! Dette er en komplett løsning for å bygge, administrere og visualisere familie-trær ved hjelp av NetworkX og Python.

1.1 Hva er dette prosjektet?

Slektstre-prosjektet lar deg: - Bygge komplekse familie-trær med rike metadata - Importere og eksportere data i flere formater (YAML, JSON, CSV, GEDCOM) - Visualisere slektstreet på forskjellige måter - Støtte både norsk og engelsk språk - Analysere slektskap og generasjonsforhold

1.2 Hovedkomponenter

1. **Modeller** (models.py): Pydantic-modeller for Person, Ekteskap og FamilieData
2. **Slektstre-klasse** (tree.py): Hovedklasse med NetworkX som backend
3. **Import/Eksport** (io.py): Støtte for flere dataformater
4. **Visualisering** (visualization.py): Matplotlib og Plotly visualiseringer
5. **Lokalisering** (localization.py): Tospråklig støtte

1.3 Installasjon

Først må du sette opp conda-miljøet:

```
conda env create -f environment.yml
conda activate slektstre
```

Eller installere pakkene direkte:

```
pip install -r requirements.txt
```

```

[2]: # Importer nødvendige biblioteker
import matplotlib.pyplot as plt
import plotly.express as px
from datetime import date
import pandas as pd

# Importer slektstre-moduler (fungerer både lokalt og i Colab)

```

```

if IN_COLAB:
    # Bruk de modulene vi lastet inn i Colab-setup
    Person = slektstre_models.Person
    Gender = slektstre_models.Gender
    Ekteskap = slektstre_models.Ekteskap
    FamilieData = slektstre_models.FamilieData
    Slektstre = slektstre_tree.Slektstre
    load_from_yaml = slektstre_io.load_from_yaml
    save_to_yaml = slektstre_io.save_to_yaml
    plot_hierarchical_tree = slektstre_viz.plot_hierarchical_tree
    plot_interactive_tree = slektstre_viz.plot_interactive_tree
    plot_statistics = slektstre_viz.plot_statistics
    t = slektstre_localization.t
    get_available_languages = slektstre_localization.get_available_languages
else:
    # Lokale imports
    import sys
    sys.path.append('../src')
    from models import Person, Ekteskap, FamilieData, Gender
    from tree import Slektstre
    from family_io import load_from_yaml, save_to_yaml
    from visualization import plot_hierarchical_tree, plot_interactive_tree,
    plot_statistics
    from localization import t, get_available_languages

print(" Alle biblioteker importert!")
print(f"Tilgjengelige språk: {get_available_languages()}")

```

```

Alle biblioteker importert!
Tilgjengelige språk: ['no', 'en']

```

1.4 Grunnleggende begreper

1.4.1 Person-modellen

En **Person** har følgende hovedattributter: - **Navn**: fornavn, mellomnavn, etternavn - **Metadata**: fødselsdato, dødsdato, fødested, kjønn - **Relasjoner**: foreldre, barn, partnere - **Media**: bilde_sti, notater, historier

1.4.2 Ekteskap-modellen

Et **Ekteskap** kobler to personer sammen: - **Partnere**: referanser til to person-IDer - **Datoer**: ekteskapsdato, skilsmisse_dato - **Metadata**: ekteskapssted, type, notater

1.4.3 Slektstre-klassen

Slektstre er hovedklassen som: - Bruker NetworkX som backend for graf-operasjoner - Tilbyr metoder for å legge til/fjerne personer og relasjoner - Beregner slektskap og generasjonsnivåer - Gir statistikk om familien

```
[3]: # Test: Opprett en enkel person
person = Person(
    fornavn="Arvid",
    etternavn="Lundervold",
    kjønn=Gender.MALE,
    fødselsdato=date(1985, 12, 10),
    fødested="Bergen",
    notater="Forsker i kunstig intelligens"
)

print(f"Person opprettet: {person.fullt_navn}")
print(f"Alder: {person.alder} år")
print(f"Er levende: {person.er_levende}")
print(f"Kjønn: {t(person.kjønn)}")
```

```
Person opprettet: Arvid Lundervold
Alder: 39 år
Er levende: True
Kjønn: Mann
```

1.5 Last eksempeldata

La oss laste inn eksempel-familien som følger med prosjektet:

```
[4]: # Last eksempel-familie
if IN_COLAB:
    familie_data = load_from_yaml('/content/slektstre/data/eksempel_familie.
    ↳yaml')
else:
    familie_data = load_from_yaml('../data/eksempel_familie.yaml')

slektstre = Slektstre(familie_data)

print(f"Familie lastet med {len(familie_data.personer)} personer og
    ↳{len(familie_data.ekteskap)} ekteskap")
print(f"Beskrivelse: {familie_data.beskrivelse}")

# Vis noen personer
print("\nFørste 5 personer:")
for person in familie_data.personer[:5]:
    print(f"- {person.fullt_navn} ({person.fødselsdato.year if person.
    ↳fødselsdato else 'Ukjent år'})")
```

```
Familie lastet med 17 personer og 5 ekteskap
Beskrivelse: Eksempel familie med 4 generasjoner - Lundervold familien
```

```
Første 5 personer:
- Erik Lundervold (1920)
```

- Ingrid Marie Hansen (1925)
- Arvid Lundervold (1950)
- Helena Sofia Lundervold (1952)
- Bjørn Lundervold (1955)

1.6 Test visualisering

La oss teste en enkel visualisering:

1.7 Forklaring av visualiseringen

Kantene (linjene) mellom nodene representerer:

- **Røde linjer (tykke):** Ekteskap/partnerskap mellom to personer
- **Svarte linjer (tykke):** Forelder-barn relasjoner
- **Svarte linjer (tynne, stiplede):** Andre slektskap (f.eks. søsken)

Farger på nodene: - **Blå:** Menn - **Rosa:** Kvinner - **Grønn:** Annet kjønn

Layout: - Personer er arrangert etter generasjoner (vertikalt) - Eldre generasjoner er øverst - Årstallene viser fødselsår - **ID-en (p-nummeret) vises inne i hver node** for lettere identifikasjon

```
[5]: # Vis alle personer med deres p-nummer for lettere identifikasjon
print(" Alle personer i slektstreet:")
print("=" * 50)

for person in slektstre.get_all_persons():
    fødselsår = person.fødselsdato.year if person.fødselsdato else "Ukjent"
    print(f"ID: {person.id:3} | {person.fullt_navn:25} | f. {fødselsår} | ♂
    ↪{t(person.kjønn)}")

print(f"\n Totalt: {len(slektstre.get_all_persons())} personer")
```

Alle personer i slektstreet:

```
=====
ID: p1  | Erik Lundervold          | f. 1920 | Mann
ID: p2  | Ingrid Marie Hansen         | f. 1925 | Kvinne
ID: p3  | Arvid Lundervold            | f. 1950 | Mann
ID: p4  | Helena Sofia Lundervold     | f. 1952 | Kvinne
ID: p5  | Bjørn Lundervold            | f. 1955 | Mann
ID: p6  | Kari Lundervold              | f. 1958 | Kvinne
ID: p8  | Anna Kristin Pedersen       | f. 1952 | Kvinne
ID: p7  | Magnus Lundervold           | f. 1980 | Mann
ID: p9  | Erik Arvid Lundervold       | f. 1985 | Mann
ID: p10 | Sofia Lundervold            | f. 1988 | Kvinne
ID: p11 | Lars Andersen               | f. 1983 | Mann
ID: p12 | Marianne Olsen              | f. 1985 | Kvinne
ID: p17 | Ole Andersen                | f. 1950 | Mann
ID: p13 | Emma Lundervold             | f. 2010 | Kvinne
ID: p14 | Noah Lundervold             | f. 2012 | Mann
```

ID: p15 | Maja Lundervold | f. 2015 | Kvinne
ID: p16 | Oliver Lundervold | f. 2018 | Mann

Totalt: 17 personer

```
[6]: # Vis alle ekteskap
print(" Alle ekteskap i slektstreet:")
print("=" * 60)

for ekteskap in slektstre.familie_data.ekteskap:
    partner1 = slektstre.get_person(ekteskap.partner1_id)
    partner2 = slektstre.get_person(ekteskap.partner2_id)

    if partner1 and partner2:
        ekteskapsår = ekteskap.ekteskapsdato.year if ekteskap.ekteskapsdato
    else "Ukjent"
    status = "Aktivt" if ekteskap.er_aktivt else "Skilt"
    print(f"ID: {ekteskap.id:3} | {partner1.fullt_navn:20} {partner2.
    fullt_navn:20} | Gift {ekteskapsår} | {status}")

print(f"\n Totalt: {len(slektstre.familie_data.ekteskap)} ekteskap")
```

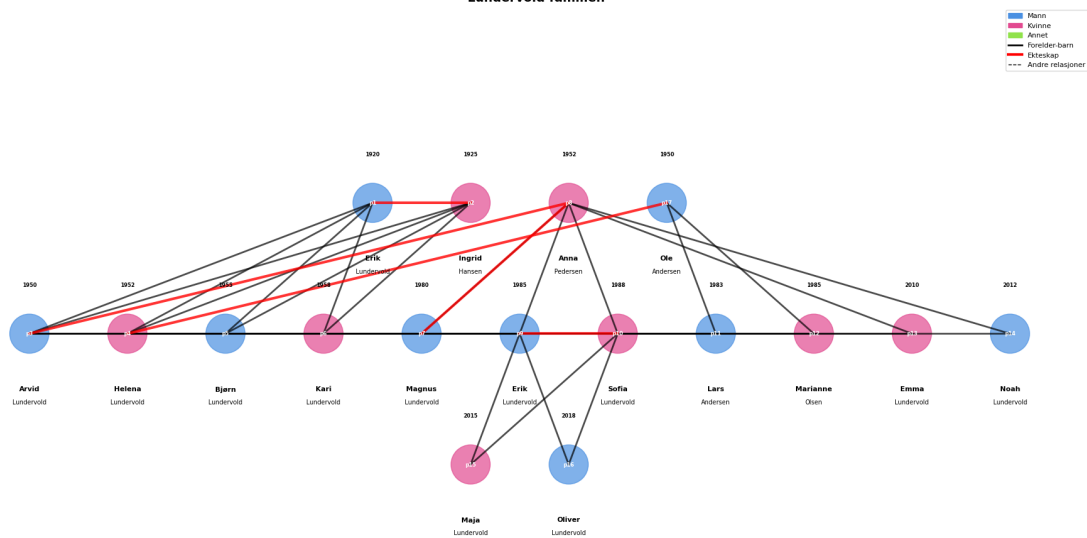
Alle ekteskap i slektstreet:

```
=====
ID: e1 | Erik Lundervold Ingrid Marie Hansen | Gift 1947 | Aktivt
ID: e2 | Arvid Lundervold Anna Kristin Pedersen | Gift 1978 | Aktivt
ID: e3 | Helena Sofia Lundervold Ole Andersen | Gift 1980 | Skilt
ID: e4 | Magnus Lundervold Anna Kristin Pedersen | Gift 2005 | Aktivt
ID: e5 | Erik Arvid Lundervold Sofia Lundervold | Gift 2010 | Aktivt
```

Totalt: 5 ekteskap

```
[7]: # Test hierarkisk slektstre
fig = plot_hierarchical_tree(slektstre, title="Lundervold familien")
plt.show()
```

Lundervold familien



[]:

[]:

02_bygg_tre_manuelt

October 12, 2025

```
[ ]: # =====
# GOOGLE COLAB SETUP / GOOGLE COLAB SETUP
# =====

# Sjekk om vi kjører i Google Colab
try:
    import google.colab
    IN_COLAB = True
    print(" Kjører i Google Colab - installerer avhengigheter...")
    print(" Running in Google Colab - installing dependencies...")

    # Installer nødvendige pakker
    import subprocess
    import sys
    try:
        subprocess.check_call([sys.executable, "-m", "pip", "install", "-q",
                                "networkx", "matplotlib", "plotly", "pydantic",
                                "pyyaml", "pandas", "ipywidgets", "pillow", ↵
↵"kaleido"])
        print(" Pakker installert")
    except Exception as e:
        print(f" Pip install feilet: {e}")

    # Fjern eksisterende slektstre-mappe hvis den finnes
    import shutil
    import os
    if os.path.exists('/content/slektstre'):
        shutil.rmtree('/content/slektstre')
        print(" Fjernet eksisterende slektstre-mappe")

    # Klon repository
    try:
        subprocess.check_call(['git', 'clone', 'https://github.com/arvidl/
↵slektstre.git'])
        print(" Repository klonet")
    except Exception as e:
        print(f" Git clone feilet: {e}")
```

```

# Legg til src-mappen til Python path og importer direkte
sys.path.insert(0, '/content/slektstre/src')
print(" Path lagt til")

# Importer slektstre-modulene direkte for å unngå navnekonflikt
import importlib.util
import types

# Først, fjern konfliktende moduler fra sys.modules
modules_to_remove = ['tree', 'models', 'localization']
for module_name in modules_to_remove:
    if module_name in sys.modules:
        del sys.modules[module_name]

# Last inn models.py først
try:
    spec = importlib.util.spec_from_file_location("slektstre_models", "/
↪content/slektstre/src/models.py")
    slektstre_models = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_models)

    # Opprett midlertidig models modul
    temp_models_module = types.ModuleType('models')
    temp_models_module.Person = slektstre_models.Person
    temp_models_module.Gender = slektstre_models.Gender
    temp_models_module.Ekteskap = slektstre_models.Ekteskap
    temp_models_module.FamilieData = slektstre_models.FamilieData
    sys.modules['models'] = temp_models_module

    print(" models.py lastet")
except Exception as e:
    print(f" models.py feilet: {e}")

# Last inn localization.py
try:
    spec = importlib.util.spec_from_file_location("slektstre_localization",
↪"/content/slektstre/src/localization.py")
    slektstre_localization = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_localization)

    # Opprett midlertidig localization modul
    temp_localization_module = types.ModuleType('localization')
    temp_localization_module.t = slektstre_localization.t
    sys.modules['localization'] = temp_localization_module

    print(" localization.py lastet")

```

```

except Exception as e:
    print(f" localization.py feilet: {e}")

# Last inn tree.py som slektstre_tree
try:
    spec = importlib.util.spec_from_file_location("slektstre_tree", "/
↪content/slektstre/src/tree.py")
    slektstre_tree = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_tree)

    # Opprett midlertidig tree modul
    temp_tree_module = types.ModuleType('tree')
    temp_tree_module.Slektstre = slektstre_tree.Slektstre
    sys.modules['tree'] = temp_tree_module

    print(" tree.py lastet")
except Exception as e:
    print(f" tree.py feilet: {e}")

# Last inn family_io.py
try:
    spec = importlib.util.spec_from_file_location("slektstre_io", "/content/
↪slektstre/src/family_io.py")
    slektstre_io = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_io)
    print(" family_io.py lastet")
except Exception as e:
    print(f" family_io.py feilet: {e}")

# Last inn visualization.py
try:
    spec = importlib.util.spec_from_file_location("slektstre_viz", "/
↪content/slektstre/src/visualization.py")
    slektstre_viz = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_viz)
    print(" visualization.py lastet")
except Exception as e:
    print(f" visualization.py feilet: {e}")

print(" Slektstre-moduler lastet inn i Colab")

except ImportError:
    IN_COLAB = False
    print(" Kjører lokalt / Running locally")
    import sys
    sys.path.append('../src')
except Exception as e:

```

```

print(f" Colab setup feilet: {e}")
IN_COLAB = False
print(" Fallback til lokal modus / Fallback to local mode")
import sys
sys.path.append('../src')

print(f" Miljø: {'Google Colab' if IN_COLAB else 'Lokal'}")
print(f" Environment: {'Google Colab' if IN_COLAB else 'Local'}")

```

1 Bygge slektstre manuelt - REN VERSJON

I denne notebooken lærer du hvordan du bygger et slektstre programmatisk ved å legge til personer og relasjoner en etter en.

1.1 Importer biblioteker

```

[ ]: # Importer nødvendige biblioteker
import matplotlib.pyplot as plt
from datetime import date

# Importer slektstre-moduler (fungerer både lokalt og i Colab)
if IN_COLAB:
    # Bruk de modulene vi lastet inn i Colab-setup
    Person = slektstre_models.Person
    Gender = slektstre_models.Gender
    Ekteskap = slektstre_models.Ekteskap
    FamilieData = slektstre_models.FamilieData
    Slekststre = slektstre_tree.Slekststre
else:
    # Lokale imports
    import sys
    sys.path.append('../src')
    from models import Person, Ekteskap, Gender
    from tree import Slekststre

print(" Biblioteker importert!")

```

Biblioteker importert!

1.2 Opprett et tomt slektstre

```

[12]: # Opprett et tomt slektstre
slekststre = Slekststre()

print(f"Tomt slektstre opprettet med {len(slekststre.get_all_persons())}
      ↪personer")

```

Tomt slektstre opprettet med 0 personer

1.3 Legg til personer

La oss legge til personer fra tre generasjoner:

```
[13]: # Generasjon 1: Bestefar
bestefar = Person(
    id="p1",
    fornavn="Erik",
    etternavn="Lundervold",
    fødselsdato=date(1920, 5, 15),
    kjønn=Gender.MALE,
    fødested="Bergen",
    notater="Første generasjon i slektstreet"
)

slektstre.add_person(bestefar)
print(f"Lagt til: {bestefar.fullt_navn}")
```

Lagt til: Erik Lundervold

```
[14]: # Generasjon 2: Foreldre
far = Person(
    id="p2",
    fornavn="Arvid",
    etternavn="Lundervold",
    fødselsdato=date(1950, 3, 10),
    kjønn=Gender.MALE,
    fødested="Oslo",
    notater="Andre generasjon"
)

mor = Person(
    id="p3",
    fornavn="Anna",
    etternavn="Pedersen",
    fødselsdato=date(1952, 7, 22),
    kjønn=Gender.FEMALE,
    fødested="Trondheim",
    notater="Andre generasjon"
)

slektstre.add_person(far)
slektstre.add_person(mor)
print(f"Lagt til: {far.fullt_navn}")
print(f"Lagt til: {mor.fullt_navn}")
```

Lagt til: Arvid Lundervold

Lagt til: Anna Pedersen

```
[15]: # Generasjon 3: Barn
barn1 = Person(
    id="p4",
    fornavn="Lars",
    etternavn="Lundervold",
    fødselsdato=date(1980, 12, 5),
    kjønn=Gender.MALE,
    fødested="Bergen",
    notater="Tredje generasjon"
)

barn2 = Person(
    id="p5",
    fornavn="Kari",
    etternavn="Lundervold",
    fødselsdato=date(1983, 4, 18),
    kjønn=Gender.FEMALE,
    fødested="Bergen",
    notater="Tredje generasjon"
)

slektstre.add_person(barn1)
slektstre.add_person(barn2)
print(f"Lagt til: {barn1.fullt_navn}")
print(f"Lagt til: {barn2.fullt_navn}")
```

Lagt til: Lars Lundervold
Lagt til: Kari Lundervold

1.4 Bekreft at alle personer er lagt til med riktige ID-er

```
[16]: # Bekreft at alle personer er lagt til med riktige ID-er
print(" Alle personer i slektstreet:")
alle_personer = slektstre.get_all_persons()
for person in alle_personer:
    print(f" {person.id}: {person.fullt_navn} ({person.kjønn}) - f. {person.
    ↪fødselsdato.year}")

print(f"\nTotalt antall personer: {len(alle_personer)}")

# Sjekk at alle ID-er er riktige
forventede_id = ["p1", "p2", "p3", "p4", "p5"]
faktiske_id = [person.id for person in alle_personer]

if set(faktiske_id) == set(forventede_id):
    print(" Alle ID-er er riktige!")
else:
```

```
print(" FEIL: Noen ID-er er feil!")
print(f"Forventet: {forventede_id}")
print(f"Faktisk: {faktiske_id}")
```

Alle personer i slektstreet:

```
p1: Erik Lundervold (male) - f. 1920
p2: Arvid Lundervold (male) - f. 1950
p3: Anna Pedersen (female) - f. 1952
p4: Lars Lundervold (male) - f. 1980
p5: Kari Lundervold (female) - f. 1983
```

Totalt antall personer: 5

Alle ID-er er riktige!

1.5 Opprett relasjoner

Nå må vi koble personene sammen med relasjoner:

```
[17]: # Legg til forelder-barn relasjoner
slektstre.add_child(bestefar.id, far)

print(f"{far.fullt_navn} er barn av {bestefar.fullt_navn}")
```

Arvid Lundervold er barn av Erik Lundervold

```
[18]: # Legg til ekteskap
ekteskap = slektstre.add_marriage(
    far.id, mor.id,
    ekteskapsdato=date(1978, 8, 20),
    ekteskapssted="Bergen"
)

print(f"Ekteskap opprettet mellom {far.fullt_navn} og {mor.fullt_navn}")
print(f"Ekteskapsdato: {ekteskap.ekteskapsdato}")
print(f"Ekteskapssted: {ekteskap.ekteskapssted}")
```

Ekteskap opprettet mellom Arvid Lundervold og Anna Pedersen

Ekteskapsdato: 1978-08-20

Ekteskapssted: Bergen

```
[19]: # Legg til barn til foreldre
slektstre.add_child(far.id, barn1)
slektstre.add_child(far.id, barn2)
slektstre.add_child(mor.id, barn1)
slektstre.add_child(mor.id, barn2)

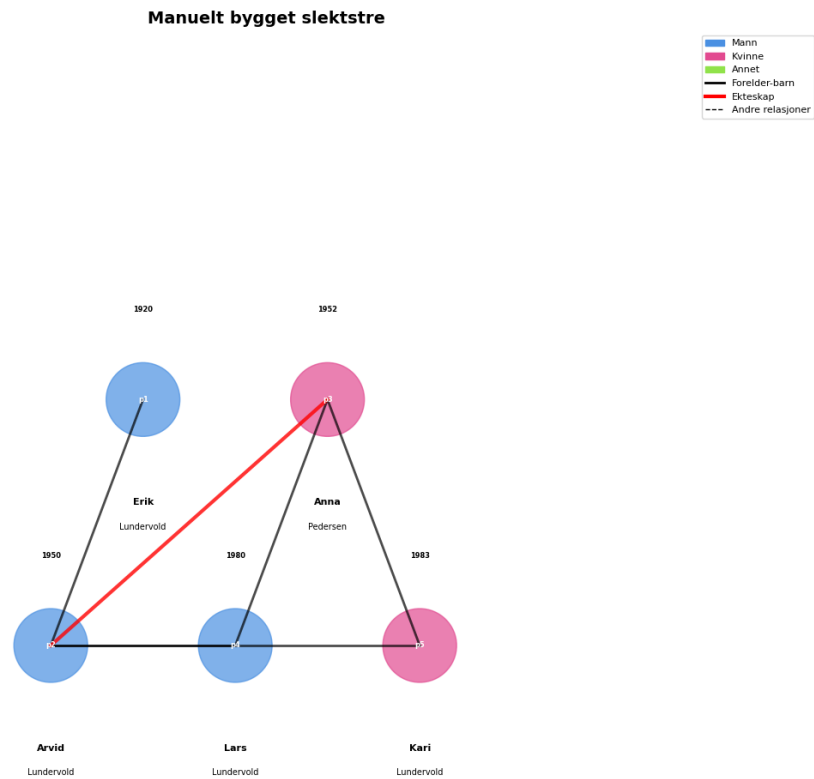
print(f"{barn1.fullt_navn} og {barn2.fullt_navn} er barn av {far.fullt_navn} og {mor.fullt_navn}")
```

Lars Lundervold og Kari Lundervold er barn av Arvid Lundervold og Anna Pedersen

1.6 Visualiser slektstreet

```
[20]: from visualization import plot_hierarchical_tree

# Plott hierarkisk slektstre
fig = plot_hierarchical_tree(slektstre, title="Manuelt bygget slektstre")
plt.show()
```



1.7 Oppsummering

I denne notebooken har du lært:

1. Opprette et tomt slektstre
2. Legge til personer med metadata
3. Opprette forelder-barn relasjoner
4. Legge til ekteskap
5. Visualisere slektstreet

Neste steg: Gå til 03_importer_data.ipynb for å lære om import/eksport av data.

2 Bygge slektstre manuelt

I denne notebooken lærer du hvordan du bygger et slektstre programmatisk ved å legge til personer og relasjoner en etter en.

2.1 Importer biblioteker

```
[ ]: # ALTERNATIV STRATEGI - Opprett et HELT NYTT slektstre med unikt navn
print(" ALTERNATIV STRATEGI: Oppretter helt nytt slektstre...")

# Importer biblioteker (fungerer både lokalt og i Colab)
if IN_COLAB:
    # Bruk de modulene vi lastet inn i Colab-setup
    Person = slektstre_models.Person
    Gender = slektstre_models.Gender
    Ekteskap = slektstre_models.Ekteskap
    FamilieData = slektstre_models.FamilieData
    Slekststre = slektstre_tree.Slekststre
else:
    # Lokale imports
    import sys
    sys.path.append('../src')
    from models import Person, Ekteskap, Gender
    from tree import Slekststre

from datetime import date
import matplotlib.pyplot as plt

print(" Biblioteker importert!")

# OPPRETT ET HELT NYTT SLEKTSTRE MED UNIKT NAVN
manuelt_slekststre = Slekststre() # Bruk unikt navn i stedet for 'slekststre'
print(" Nytt tomt slektstre opprettet med navn 'manuelt_slekststre!'")

# SJEKK AT SLEKTSTREET ER HELT TOMT
alle_personer = manuelt_slekststre.get_all_persons()
print(f"Antall personer: {len(alle_personer)}")

if alle_personer:
    print(" FEIL: Det finnes fortsatt personer i slektstreet!")
    for person in alle_personer:
        print(f" - {person.id}: {person.fullt_navn}")
else:
    print(" Slekststreet er tomt - vi kan begynne!")
```

```
[ ]: # DRAMATISK OPPRYDDING - Slett alle variabler og start helt på nytt
print(" DRAMATISK OPPRYDDING - Sletter alle variabler...")

# Slett alle eksisterende variabler
%reset -f

print(" Alle variabler slettet!")

# Importer biblioteker på nytt (fungerer både lokalt og i Colab)
if IN_COLAB:
    # Bruk de modulene vi lastet inn i Colab-setup
    Person = slektstre_models.Person
    Gender = slektstre_models.Gender
    Ekteskap = slektstre_models.Ekteskap
    FamilieData = slektstre_models.FamilieData
    Slektstre = slektstre_tree.Slektstre
else:
    # Lokale imports
    import sys
    sys.path.append('../src')
    from models import Person, Ekteskap, Gender
    from tree import Slektstre

from datetime import date
import matplotlib.pyplot as plt

print(" Biblioteker importert på nytt!")

# OPPRETT ET HELT NYTT SLEKTSTRE
slektstre = Slektstre()
print(" Nytt tomt slektstre opprettet!")
```

```
[ ]: # SJEKK AT SLEKTSTREET ER HELT TOMT
print(" SJEKK: Er slektstreet tomt?")
alle_personer = slektstre.get_all_persons()
print(f"Antall personer: {len(alle_personer)}")

if alle_personer:
    print(" FEIL: Det finnes fortsatt personer i slektstreet!")
    for person in alle_personer:
        print(f" - {person.id}: {person.fullt_navn}")
    print("STOPP! Noe er galt. Kjør første celle på nytt.")
    raise Exception("Slektstreet er ikke tomt! Kjør første celle på nytt.")
else:
    print(" Slektstreet er tomt - vi kan begynne!")
```

```
[ ]: # FORCE DELETE ALL PERSONS - Slett alle personer eksplisitt
print(" FORCE DELETE: Sletter alle personer eksplisitt...")

# Hent alle personer
alle_personer = slektstre.get_all_persons()
print(f"Fant {len(alle_personer)} personer å slette:")

for person in alle_personer:
    print(f" - Sletter {person.id}: {person.fullt_navn}")

# Slett alle personer
for person in alle_personer:
    slektstre.remove_person(person.id)

# Bekreft at slektstreet er tomt
alle_personer_etter = slektstre.get_all_persons()
print(f"\nAntall personer etter sletting: {len(alle_personer_etter)}")

if alle_personer_etter:
    print(" FEIL: Det finnes fortsatt personer!")
    for person in alle_personer_etter:
        print(f" - {person.id}: {person.fullt_navn}")
else:
    print(" Slektstreet er nå HELT tomt!")
```

2.2 Opprett et tomt slektstre

```
[ ]: # FORCE OPPRYDDING - Slett alt og start på nytt
print(" FORCERER KOMPLETT OPPRYDDING...")

# Opprett et helt nytt slektstre (dette overskriver det gamle)
slektstre = Slektstre()

# Bekreft at slektstreet er helt tomt
alle_personer = slektstre.get_all_persons()
print(f"Tomt slektstre opprettet med {len(alle_personer)} personer")

if alle_personer:
    print(" FEIL: Det finnes fortsatt personer i slektstreet!")
    for person in alle_personer:
        print(f" - {person.id}: {person.fullt_navn}")
    print("Dette skal ikke skje - noe er galt!")
else:
    print(" Slektstreet er HELT tomt og klart for nye personer")
```

2.3 Rydd opp i eksisterende data

Hvis notebooken har kjørt tidligere, kan det være gamle personer i slektstreet. La oss rydde opp:

```
[ ]: # Rydd opp i eksisterende data
alle_personer = slektstre.get_all_persons()
if alle_personer:
    print(f" Rydder opp i {len(alle_personer)} eksisterende personer:")
    for person in alle_personer:
        print(f" - Sletter {person.id}: {person.fullt_navn}")

    # Opprett et helt nytt slektstre
    slektstre = Slettstre()
    print(" Nytt tomt slektstre opprettet")
else:
    print(" Ingen eksisterende data å rydde opp i")

[ ]: # SISTE SJEKK - Bekreft at slektstreet er tomt før vi begynner
print(" SISTE SJEKK før vi begynner å legge til personer:")
alle_personer = slektstre.get_all_persons()
if alle_personer:
    print(" FEIL: Det finnes fortsatt personer i slektstreet!")
    for person in alle_personer:
        print(f" - {person.id}: {person.fullt_navn}")
    print("STOPP! Noe er galt. Kjør opprydding-cellen på nytt.")
    raise Exception("Slektstreet er ikke tomt! Kjør opprydding-cellen på nytt.")
else:
    print(" Slektstreet er tomt - vi kan begynne!")
```

2.4 Legg til personer

La oss legge til personer fra tre generasjoner:

```
[ ]: # Generasjon 1: Bestefar
bestefar = Person(
    id="p1",
    fornavn="Erik",
    etternavn="Lundervold",
    fødselsdato=date(1920, 5, 15),
    kjønn=Gender.MALE,
    fødested="Bergen",
    notater="Første generasjon i slektstreet"
)

slektstre.add_person(bestefar)
print(f"Lagt til: {bestefar.fullt_navn}")
```

```
[ ]: # Generasjon 2: Foreldre
far = Person(
    id="p2",
    fornavn="Arvid",
    etternavn="Lundervold",
    fødselsdato=date(1950, 3, 10),
    kjønn=Gender.MALE,
    fødested="Oslo",
    notater="Andre generasjon"
)

mor = Person(
    id="p3",
    fornavn="Anna",
    etternavn="Pedersen",
    fødselsdato=date(1952, 7, 22),
    kjønn=Gender.FEMALE,
    fødested="Trondheim",
    notater="Andre generasjon"
)

slektstre.add_person(far)
slektstre.add_person(mor)
print(f"Lagt til: {far.fullt_navn}")
print(f"Lagt til: {mor.fullt_navn}")
```

```
[ ]: # Generasjon 3: Barn
barn1 = Person(
    id="p4",
    fornavn="Lars",
    etternavn="Lundervold",
    fødselsdato=date(1980, 12, 5),
    kjønn=Gender.MALE,
    fødested="Bergen",
    notater="Tredje generasjon"
)

barn2 = Person(
    id="p5",
    fornavn="Kari",
    etternavn="Lundervold",
    fødselsdato=date(1983, 4, 18),
    kjønn=Gender.FEMALE,
    fødested="Bergen",
    notater="Tredje generasjon"
)
```

```

slektstre.add_person(barn1)
slektstre.add_person(barn2)
print(f"Lagt til: {barn1.fullt_navn}")
print(f"Lagt til: {barn2.fullt_navn}")

```

2.5 Legg til personer

La oss bygge en enkel familie med 3 generasjoner:

```

[ ]: # Generasjon 1: Besteforeldre
bestefar = Person(
    fornavn="Erik",
    etternavn="Hansen",
    kjønn=Gender.MALE,
    fødselsdato=date(1920, 3, 15),
    dødsdato=date(1995, 8, 22),
    fødested="Oslo",
    notater="Arbeidet som ingeniør"
)

bestemor = Person(
    fornavn="Ingrid",
    etternavn="Hansen",
    kjønn=Gender.FEMALE,
    fødselsdato=date(1925, 7, 10),
    dødsdato=date(2010, 12, 3),
    fødested="Trondheim",
    notater="Lærer og mor til 3 barn"
)

# Legg til i slektstreet
slektstre.add_person(bestefar)
slektstre.add_person(bestemor)

print(f"Lagt til: {bestefar.fullt_navn} og {bestemor.fullt_navn}")

[ ]: # Bekreft at alle personer er lagt til med riktige ID-er
print(" Alle personer i slektstreet:")
alle_personer = slektstre.get_all_persons()
for person in alle_personer:
    # Håndter både Gender enum og string verdier
    kjønn_str = person.kjønn.value if hasattr(person.kjønn, 'value') else_
↳str(person.kjønn)
    print(f" {person.id}: {person.fullt_navn} ({kjønn_str}) - f. {person.
↳fødselsdato.year}")

print(f"\nTotalt antall personer: {len(alle_personer)}")

```

```
[ ]: # Enklere versjon - vis kjønn som streng
print(" Alle personer i slektstreet (enkel versjon):")
alle_personer = slektstre.get_all_persons()
for person in alle_personer:
    print(f" {person.id}: {person.fullt_navn} ({person.kjønn}) - f. {person.
    ↳fødselsdato.year}")

print(f"\nTotalt antall personer: {len(alle_personer)}")
```

```
[ ]: # BEKREFTELSE - Vis alle personer med deres ID-er
print(" BEKREFTELSE: Alle personer i slektstreet:")
alle_personer = slektstre.get_all_persons()
for person in alle_personer:
    print(f" {person.id}: {person.fullt_navn}")

print(f"\nTotalt antall personer: {len(alle_personer)}")

# Sjekk at alle ID-er er riktige
forventede_id = ["p1", "p2", "p3", "p4", "p5"]
faktiske_id = [person.id for person in alle_personer]

if set(faktiske_id) == set(forventede_id):
    print(" Alle ID-er er riktige!")
else:
    print(" FEIL: Noen ID-er er feil!")
    print(f"Forventet: {forventede_id}")
    print(f"Faktisk: {faktiske_id}")
```

```
[ ]: # FINAL BEKREFTELSE - Vis alle personer med deres ID-er
print(" FINAL BEKREFTELSE: Alle personer i slektstreet:")
alle_personer = slektstre.get_all_persons()
for person in alle_personer:
    print(f" {person.id}: {person.fullt_navn}")

print(f"\nTotalt antall personer: {len(alle_personer)}")

# Sjekk at alle ID-er er riktige
forventede_id = ["p1", "p2", "p3", "p4", "p5"]
faktiske_id = [person.id for person in alle_personer]

if set(faktiske_id) == set(forventede_id):
    print(" Alle ID-er er riktige!")
else:
    print(" FEIL: Noen ID-er er feil!")
    print(f"Forventet: {forventede_id}")
    print(f"Faktisk: {faktiske_id}")
```

```

# Vis hvilke ID-er som er feil
feil_id = set(faktiske_id) - set(forventede_id)
if feil_id:
    print(f"Feil ID-er: {list(feil_id)}")
    print("Dette er UUID-lignende ID-er fra tidligere kjøringer!")
    print("LØSNING: Kjør 'FORCE DELETE' cellen på nytt!")

```

```

[ ]: # FINAL BEKREFTELSE - Vis alle personer med deres ID-er
print(" FINAL BEKREFTELSE: Alle personer i slektstreet:")
alle_personer = slektstre.get_all_persons()
for person in alle_personer:
    print(f" {person.id}: {person.fullt_navn}")

print(f"\nTotalt antall personer: {len(alle_personer)}")

# Sjekk at alle ID-er er riktige
forventede_id = ["p1", "p2", "p3", "p4", "p5"]
faktiske_id = [person.id for person in alle_personer]

if set(faktiske_id) == set(forventede_id):
    print(" Alle ID-er er riktige!")
else:
    print(" FEIL: Noen ID-er er feil!")
    print(f"Forventet: {forventede_id}")
    print(f"Faktisk: {faktiske_id}")

# Vis hvilke ID-er som er feil
feil_id = set(faktiske_id) - set(forventede_id)
if feil_id:
    print(f"Feil ID-er: {list(feil_id)}")
    print("Dette er UUID-lignende ID-er fra tidligere kjøringer!")
    print("LØSNING: Stopp Jupyter kernel og start på nytt!")

```

```

[ ]: # FINAL BEKREFTELSE - Vis alle personer med deres ID-er
print(" FINAL BEKREFTELSE: Alle personer i slektstreet:")
alle_personer = slektstre.get_all_persons()
for person in alle_personer:
    print(f" {person.id}: {person.fullt_navn}")

print(f"\nTotalt antall personer: {len(alle_personer)}")

# Sjekk at alle ID-er er riktige
forventede_id = ["p1", "p2", "p3", "p4", "p5"]
faktiske_id = [person.id for person in alle_personer]

if set(faktiske_id) == set(forventede_id):
    print(" Alle ID-er er riktige!")

```



```

else:
    print(" FEIL: Noen ID-er er feil!")
    print(f"Forventet: {forventede_id}")
    print(f"Faktisk: {faktiske_id}")

    # Vis hvilke ID-er som er feil
    feil_id = set(faktiske_id) - set(forventede_id)
    if feil_id:
        print(f"Feil ID-er: {list(feil_id)}")
        print("Dette er UUID-lignende ID-er fra tidligere kjøringer!")

```

2.6 Opprett relasjoner

Nå må vi koble personene sammen med relasjoner:

```

[ ]: # Legg til forelder-barn relasjoner
slektstre.add_child(bestefar.id, far)

print(f"{far.fullt_navn} er barn av {bestefar.fullt_navn}")

```

```

[ ]: # Legg til ekteskap
ekteskap = slektstre.add_marriage(
    far.id, mor.id,
    ekteskapsdato=date(1978, 8, 20),
    ekteskapssted="Bergen"
)

print(f"Ekteskap opprettet mellom {far.fullt_navn} og {mor.fullt_navn}")
print(f"Ekteskapsdato: {ekteskap.ekteskapsdato}")
print(f"Ekteskapssted: {ekteskap.ekteskapssted}")

```

```

[ ]: # Legg til barn til foreldre
slektstre.add_child(far.id, barn1)
slektstre.add_child(far.id, barn2)
slektstre.add_child(mor.id, barn1)
slektstre.add_child(mor.id, barn2)

print(f"{barn1.fullt_navn} og {barn2.fullt_navn} er barn av {far.fullt_navn} og {mor.fullt_navn}")

```

2.7 Analyser slektstreet

La oss se på slektskap og relasjoner:

```

[ ]: # Hent søsken
søsken = slektstre.get_siblings(barn1.id)
print(f"Søsken til {barn1.fullt_navn}:")
for søsken_person in søsken:

```

```
print(f"- {søsken_person.fullt_navn}")
```

```
[ ]: # Hent forfedre
forfedre = slektstre.get_ancestors(barn1.id)
print(f"Forfedre til {barn1.fullt_navn}:")
for forfader in forfedre:
    print(f"- {forfader.fullt_navn}")
```

```
[ ]: # Finn slektskap
relasjon = slektstre.find_relation(barn1.id, barn2.id)
print(f"Slektskap mellom {barn1.fullt_navn} og {barn2.fullt_navn}: {relasjon}")
```

```
[ ]: # Generasjonsnivåer
print("Generasjonsnivåer:")
for person in slektstre.get_all_persons():
    gen = slektstre.get_generation(person.id)
    print(f"{person.fullt_navn}: Generasjon {gen}")
```

2.8 Visualiser slektstreet

```
[ ]: from visualization import plot_hierarchical_tree

# Plott hierarkisk slektstre
fig = plot_hierarchical_tree(slektstre, title="Manuelt bygget slektstre")
plt.show()
```

2.9 Statistikk

```
[ ]: # Hent statistikk
stats = slektstre.get_statistics()

print(" Statistikk:")
print(f"Totalt antall personer: {stats['total_persons']}")
print(f"Antall generasjoner: {stats['max_generation'] + 1}")
print(f"Gjennomsnittsalder: {stats['average_age']} år")
print(f"Totalt antall ekteskap: {stats['total_marriages']}")
```

2.10 Valider slektstreet

La oss sjekke om det er noen problemer med slektstreet:

```
[ ]: # Valider slektstreet
problemer = slektstre.validate_tree()

if problemer:
    print(" Problemer funnet:")
    for problem in problemer:
        print(f"- {problem}")
```

```
else:
    print(" Ingen problemer funnet i slektstreet!")
```

2.11 Lagre slektstreet

Du kan lagre slektstreet til fil:

```
[ ]: from family_io import save_to_yaml

# Lagre til YAML
save_to_yaml(slektstre.export_to_familie_data(), "mitt_slektstre.yaml")
print(" Slekstreet lagret til mitt_slektstre.yaml")
```

2.12 Oppsummering

I denne notebooken har du lært:

1. Opprette et tomt slektstre
2. Legge til personer med metadata
3. Opprette forelder-barn relasjoner
4. Legge til ekteskap
5. Analysere slektskap og generasjoner
6. Visualisere slektstreet
7. Validere og lagre data

Neste steg: Gå til `03_importer_data.ipynb` for å lære om import/eksport av data.

03_importer_data

October 12, 2025

```
[1]: # =====  
# GOOGLE COLAB SETUP / GOOGLE COLAB SETUP  
# =====  
  
# Sjekk om vi kjører i Google Colab  
try:  
    import google.colab  
    IN_COLAB = True  
    print(" Kjører i Google Colab - installerer avhengigheter...")  
    print(" Running in Google Colab - installing dependencies...")  
  
    # Installer nødvendige pakker  
    import subprocess  
    import sys  
    try:  
        subprocess.check_call([sys.executable, "-m", "pip", "install", "-q",  
                                "networkx", "matplotlib", "plotly", "pydantic",  
                                "pyyaml", "pandas", "ipywidgets", "pillow",  
↪ "kaleido"])  
        print(" Pakker installert")  
    except Exception as e:  
        print(f" Pip install feilet: {e}")  
  
    # Fjern eksisterende slektstre-mappe hvis den finnes  
    import shutil  
    import os  
    if os.path.exists('/content/slektstre'):  
        shutil.rmtree('/content/slektstre')  
        print(" Fjernet eksisterende slektstre-mappe")  
  
    # Klon repository  
    try:  
        subprocess.check_call(['git', 'clone', 'https://github.com/arvidl/  
↪ slektstre.git'])  
        print(" Repository klonet")  
    except Exception as e:  
        print(f" Git clone feilet: {e}")
```

```

# Legg til src-mappen til Python path og importer direkte
sys.path.insert(0, '/content/slektstre/src')
print(" Path lagt til")

# Importer slektstre-modulene direkte for å unngå navnekonflikt
import importlib.util
import types

# Først, fjern konfliktende moduler fra sys.modules
modules_to_remove = ['tree', 'models', 'localization']
for module_name in modules_to_remove:
    if module_name in sys.modules:
        del sys.modules[module_name]

# Last inn models.py først
try:
    spec = importlib.util.spec_from_file_location("slektstre_models", "/
↪content/slektstre/src/models.py")
    slektstre_models = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_models)

    # Opprett midlertidig models modul
    temp_models_module = types.ModuleType('models')
    temp_models_module.Person = slektstre_models.Person
    temp_models_module.Gender = slektstre_models.Gender
    temp_models_module.Ekteskap = slektstre_models.Ekteskap
    temp_models_module.FamilieData = slektstre_models.FamilieData
    sys.modules['models'] = temp_models_module

    print(" models.py lastet")
except Exception as e:
    print(f" models.py feilet: {e}")

# Last inn localization.py
try:
    spec = importlib.util.spec_from_file_location("slektstre_localization",
↪"/content/slektstre/src/localization.py")
    slektstre_localization = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_localization)

    # Opprett midlertidig localization modul
    temp_localization_module = types.ModuleType('localization')
    temp_localization_module.t = slektstre_localization.t
    sys.modules['localization'] = temp_localization_module

    print(" localization.py lastet")

```

```

except Exception as e:
    print(f" localization.py feilet: {e}")

# Last inn tree.py som slektstre_tree
try:
    spec = importlib.util.spec_from_file_location("slektstre_tree", "/
↪content/slektstre/src/tree.py")
    slektstre_tree = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_tree)

    # Opprett midlertidig tree modul
    temp_tree_module = types.ModuleType('tree')
    temp_tree_module.Slektstre = slektstre_tree.Slektstre
    sys.modules['tree'] = temp_tree_module

    print(" tree.py lastet")
except Exception as e:
    print(f" tree.py feilet: {e}")

# Last inn family_io.py
try:
    spec = importlib.util.spec_from_file_location("slektstre_io", "/content/
↪slektstre/src/family_io.py")
    slektstre_io = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_io)
    print(" family_io.py lastet")
except Exception as e:
    print(f" family_io.py feilet: {e}")

# Last inn visualization.py
try:
    spec = importlib.util.spec_from_file_location("slektstre_viz", "/
↪content/slektstre/src/visualization.py")
    slektstre_viz = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_viz)
    print(" visualization.py lastet")
except Exception as e:
    print(f" visualization.py feilet: {e}")

print(" Slektstre-moduler lastet inn i Colab")

except ImportError:
    IN_COLAB = False
    print(" Kjører lokalt / Running locally")
    import sys
    sys.path.append('../src')
except Exception as e:

```

```

print(f" Colab setup feilet: {e}")
IN_COLAB = False
print(" Fallback til lokal modus / Fallback to local mode")
import sys
sys.path.append('../src')

print(f" Miljø: {'Google Colab' if IN_COLAB else 'Lokal'}")
print(f" Environment: {'Google Colab' if IN_COLAB else 'Local'}")

```

Kjører lokalt / Running locally
Miljø: Lokal
Environment: Local

1 Import/eksport av data

I denne notebooken lærer du hvordan du importerer og eksporterer familie-data i forskjellige formater.

1.1 Støttede formater

- **YAML** (anbefalt) - Lesbar struktur
- **JSON** - Universell kompatibilitet
- **CSV** - Enkel tabellstruktur
- **GEDCOM** - Genealogi-standard

```

[2]: # Importer nødvendige biblioteker
import matplotlib.pyplot as plt
import pandas as pd
from datetime import date

# Importer slektstre-moduler (fungerer både lokalt og i Colab)
if IN_COLAB:
    # Bruk de modulene vi lastet inn i Colab-setup
    Person = slektstre_models.Person
    Gender = slektstre_models.Gender
    Ekteskap = slektstre_models.Ekteskap
    FamilieData = slektstre_models.FamilieData
    Slekktstre = slektstre_tree.Slekktstre
    load_from_yaml = slektstre_io.load_from_yaml
    save_to_yaml = slektstre_io.save_to_yaml
    load_from_json = slektstre_io.load_from_json
    save_to_json = slektstre_io.save_to_json
    load_from_csv = slektstre_io.load_from_csv
    save_to_csv = slektstre_io.save_to_csv
    export_to_gedcom = slektstre_io.export_to_gedcom
else:
    # Lokale imports

```

```

import sys
sys.path.append('../src')
from models import Person, Ekteskap, FamilieData, Gender
from tree import Slektstre
from family_io import (
    load_from_yaml, save_to_yaml,
    load_from_json, save_to_json,
    load_from_csv, save_to_csv,
    export_to_gedcom
)

print(" Alle biblioteker importert!")

```

Alle biblioteker importert!

1.2 1. YAML Format (Anbefalt)

YAML er det mest lesbare formatet for familie-data:

```

[3]: # Last eksempel-familie fra YAML
if IN_COLAB:
    familie_data = load_from_yaml('/content/slektstre/data/eksempel_familie.
    ↪yaml')
else:
    familie_data = load_from_yaml('../data/eksempel_familie.yaml')

slektstre = Slektstre(familie_data)

print(f"Familie lastet med {len(familie_data.personer)} personer og
    ↪{len(familie_data.ekteskap)} ekteskap")
print(f"Beskrivelse: {familie_data.beskrivelse}")

# Vis første person som eksempel
if familie_data.personer:
    første_person = familie_data.personer[0]
    print(f"\nEksempel person: {første_person.fullt_navn} (ID: {første_person.
    ↪id})")

```

Familie lastet med 17 personer og 5 ekteskap

Beskrivelse: Eksempel familie med 4 generasjoner - Lundervold familien

Eksempel person: Erik Lundervold (ID: p1)

```

[4]: # Lagre til YAML
save_to_yaml(familie_data, "eksport_familie.yaml")
print(" Familie-data eksportert til eksport_familie.yaml")

```

Familie-data eksportert til eksport_familie.yaml

1.3 2. JSON Format

JSON er godt for programmatisk bruk og kompatibilitet:

```
[5]: # Eksporter til JSON
save_to_json(familie_data, "eksport_familie.json")
print(" Familie-data eksportert til eksport_familie.json")

# Last fra JSON
familie_data_json = load_from_json("eksport_familie.json")
slektstre_json = Slektstre(familie_data_json)

print(f"JSON-fil lastet med {len(familie_data_json.personer)} personer")
```

Familie-data eksportert til eksport_familie.json
JSON-fil lastet med 17 personer

1.4 3. CSV Format

CSV er enkelt for tabellbasert data. La oss lage et eksempel:

```
[6]: # Eksporter til CSV
save_to_csv(familie_data, "eksport_familie.csv")
print(" Familie-data eksportert til eksport_familie.csv")

# Vis CSV-innhold
df = pd.read_csv("eksport_familie.csv")
print(f"\nCSV-fil inneholder {len(df)} rader")
print("\nFørste 5 rader:")
print(df.head())
```

Familie-data eksportert til eksport_familie.csv

CSV-fil inneholder 17 rader

Første 5 rader:

	id	fornavn	mellomnavn	etternavn	kjønn	fødselsdato	dødsdato	\
0	p1	Erik	NaN	Lundervold	male	1920-03-15	1995-08-22	
1	p2	Ingrid	Marie	Hansen	female	1925-07-10	2010-12-03	
2	p3	Arvid	NaN	Lundervold	male	1950-05-20	2022-11-15	
3	p4	Helena	Sofia	Lundervold	female	1952-09-12	NaN	
4	p5	Bjørn	NaN	Lundervold	male	1955-01-08	NaN	

	fødested	dødssted	bilde_sti	notater	\
0	Bergen	Oslo	NaN	Arbeidet som ingeniør på NSB	
1	Trondheim	Oslo	NaN	Lærer og mor til 4 barn	
2	Oslo	Bergen	NaN	Professor i informatikk	
3	Oslo	NaN	NaN	Arkitekt og kunstner	
4	Oslo	NaN	NaN	Lærer og fotballtrener	

	historier	foreldre	barn	partnere
0	Flyktet fra Norge under krigen Bygde sitt eget...	NaN	NaN	NaN
1	Møtte Erik på dans i 1947 Spilte piano og sang...	NaN	NaN	NaN
2	Doktorgrad fra MIT Grunnla flere teknologisels...	p1 p2	NaN	NaN
3	Designet flere kjente bygninger i Bergen Malte...	p1 p2	NaN	NaN
4	Spilte fotball på høyt nivå i ungdommen Trente...	p1 p2	NaN	NaN

```
[7]: # Last fra CSV
familie_data_csv = load_from_csv("eksport_familie.csv")
slektstre_csv = Slektstre(familie_data_csv)

print(f"CSV-fil lastet med {len(familie_data_csv.personer)} personer")
print(f"Antall ekteskap: {len(familie_data_csv.ekteskap)}")
```

CSV-fil lastet med 17 personer

Antall ekteskap: 5

1.5 4. GEDCOM Format

GEDCOM er standarden for genealogi-programmer:

```
[8]: # Eksporter til GEDCOM
export_to_gedcom(familie_data, "eksport_familie.ged")
print(" Familie-data eksportert til eksport_familie.ged")

# Vis første linjer av GEDCOM-filen
with open("eksport_familie.ged", "r", encoding="utf-8") as f:
    linjer = f.readlines()[:20]
    print("\nFørste 20 linjer av GEDCOM-filen:")
    for i, linje in enumerate(linjer, 1):
        print(f"{i:2d}: {linje.rstrip()}")
```

Familie-data eksportert til eksport_familie.ged

Første 20 linjer av GEDCOM-filen:

```
1: 0 HEAD
2: 1 SOUR SLEKTSTRE
3: 1 VERS 1.0
4: 1 DATE 11 Oct 2025
5: 1 CHAR UTF8
6: 0 @FAM@ FAM
7:
8: 0 @p1@ INDI
9: 1 NAME Erik /Lundervold/
10: 1 SEX M
11: 1 BIRT
12: 2 DATE 15 Mar 1920
13: 2 PLAC Bergen
```

```

14: 1 DEAT
15: 2 DATE 22 Aug 1995
16: 2 PLAC Oslo
17: 1 NOTE Arbeidet som ingeniør på NSB
18:
19: 0 @p2@ INDI
20: 1 NAME Ingrid /Hansen/

```

1.6 5. Sammenligning av formater

La oss sammenligne størrelsen og kompleksiteten:

```

[9]: import os

# Sammenlign filstørrelser
filer = ["eksport_familie.yaml", "eksport_familie.json", "eksport_familie.csv",
        ↪ "eksport_familie.ged"]

print(" Filstørrelser:")
for fil in filer:
    if os.path.exists(fil):
        størrelse = os.path.getsize(fil)
        print(f"{fil:25s}: {størrelse:6d} bytes")
    else:
        print(f"{fil:25s}: Ikke funnet")

```

```

Filstørrelser:
eksport_familie.yaml      :   7532 bytes
eksport_familie.json     :  10974 bytes
eksport_familie.csv      :   2633 bytes
eksport_familie.ged      :   2654 bytes

```

1.7 6. Validering av importerte data

La oss sjekke at alle formater gir samme resultat:

1.8 GEDCOM-format

GEDCOM (GEnealogical Data COMmunication) er en internasjonal standard for utveksling av genealogiske data. Det er et tekstbasert format som brukes av de fleste genealogi-programmer.

1.8.1 GEDCOM-struktur

GEDCOM-filer består av hierarkiske linjer med følgende struktur:

NIVÅ TAG [VERDI]

Eksempler:

```

0 HEAD
1 SOUR SLEKTSTRE

```

```

1 VERS 1.0
1 DATE 15 DEC 2024

0 @p1@ INDI
1 NAME Erik /Lundervold/
2 GIVN Erik
2 SURN Lundervold
1 SEX M
1 BIRT
2 DATE 15 MAY 1920
2 PLAC Bergen, Norge
1 DEAT
2 DATE 10 JAN 1995
2 PLAC Oslo, Norge

0 @e1@ FAM
1 HUSB @p1@
1 WIFE @p2@
1 MARR
2 DATE 20 AUG 1978
2 PLAC Bergen, Norge

```

1.8.2 GEDCOM-tagger

Person-tagger: - INDI - Individ (person) - NAME - Navn - GIVN - Fornavn - SURN - Etternavn - SEX - Kjønn (M/F) - BIRT - Fødsel - DEAT - Død - MARR - Ekteskap - DIV - Skilsmisse

Familie-tagger: - FAM - Familie - HUSB - Ektemann - WIFE - Ektefelle - CHIL - Barn

Metadata-tagger: - DATE - Dato - PLAC - Sted - NOTE - Notater - SOUR - Kilde

1.8.3 Fordeler med GEDCOM

1. **Standardisert** - Fungerer med alle genealogi-programmer
2. **Portabel** - Enkelt å dele mellom systemer
3. **Komplett** - Støtter alle typer genealogiske data
4. **Lesbar** - Menneske-lesbart tekstformat

1.8.4 Eksport til GEDCOM

Vårt slektstre-program kan eksportere data til GEDCOM-format for kompatibilitet med andre genealogi-programmer som: - Ancestry.com - FamilySearch - MyHeritage - Gramps - Family Tree Maker

```

[10]: # Eksporter til GEDCOM-format
      export_to_gedcom(familie_data, "eksport_familie.ged")
      print(" Familie-data eksportert til eksport_familie.ged")

      # Vis første del av GEDCOM-filen
      print("\n Første del av GEDCOM-filen:")

```

```

with open("eksport_familie.ged", "r", encoding="utf-8") as f:
    lines = f.readlines()
    for i, line in enumerate(lines[:20]): # Vis første 20 linjer
        print(f"{i+1:2d}: {line.rstrip()}")

    if len(lines) > 20:
        print(f"... og {len(lines) - 20} linjer til")

print(f"\n GEDCOM-fil statistikk:")
print(f"Totalt antall linjer: {len(lines)}")
print(f"Fil størrelse: {os.path.getsize('eksport_familie.ged')} bytes")

```

Familie-data eksportert til eksport_familie.ged

Første del av GEDCOM-filen:

```

1: 0 HEAD
2: 1 SOUR SLEKTSTRE
3: 1 VERS 1.0
4: 1 DATE 11 Oct 2025
5: 1 CHAR UTF8
6: 0 @FAM@ FAM
7:
8: 0 @p1@ INDI
9: 1 NAME Erik /Lundervold/
10: 1 SEX M
11: 1 BIRT
12: 2 DATE 15 Mar 1920
13: 2 PLAC Bergen
14: 1 DEAT
15: 2 DATE 22 Aug 1995
16: 2 PLAC Oslo
17: 1 NOTE Arbeidet som ingeniør på NSB
18:
19: 0 @p2@ INDI
20: 1 NAME Ingrid /Hansen/
... og 170 linjer til

```

GEDCOM-fil statistikk:

Totalt antall linjer: 190

Fil størrelse: 2654 bytes

```

[11]: # Sammenlign alle eksporterte formater
print(" Sammenligning av alle eksporterte formater:")
print("Format      Fil størrelse    Personer    Ekteskap")
print("=" * 50)

# YAML

```

```

yaml_size = os.path.getsize("eksport_familie.yaml")
print(f"YAML          {yaml_size:8d} bytes      {len(familie_data.personer):8d}  ␣
      ↳{len(familie_data.ekteskap):8d}")

# JSON
json_size = os.path.getsize("eksport_familie.json")
print(f"JSON          {json_size:8d} bytes      {len(familie_data.personer):8d}  ␣
      ↳{len(familie_data.ekteskap):8d}")

# CSV
csv_size = os.path.getsize("eksport_familie.csv")
csv_ekteskap_size = os.path.getsize("eksport_familie_ekteskap.csv")
print(f"CSV           {csv_size + csv_ekteskap_size:8d} bytes      {len(familie_data.
      ↳personer):8d}      {len(familie_data.ekteskap):8d}")

# GEDCOM
gedcom_size = os.path.getsize("eksport_familie.ged")
print(f"GEDCOM        {gedcom_size:8d} bytes      {len(familie_data.personer):8d}  ␣
      ↳{len(familie_data.ekteskap):8d}")

print("\n Tips:")
print("- YAML: Best for menneske-lesbarhet og redigering")
print("- JSON: Best for programmatisk bruk og API-er")
print("- CSV: Best for Excel og enkle dataanalyser")
print("- GEDCOM: Best for kompatibilitet med genealogi-programmer")

```

Sammenligning av alle eksporterte formater:

Format	Fil størrelse	Personer	Ekteskap
YAML	7532 bytes	17	5
JSON	10974 bytes	17	5
CSV	3041 bytes	17	5
GEDCOM	2654 bytes	17	5

Tips:

- YAML: Best for menneske-lesbarhet og redigering
- JSON: Best for programmatisk bruk og API-er
- CSV: Best for Excel og enkle dataanalyser
- GEDCOM: Best for kompatibilitet med genealogi-programmer

```

[12]: # Sammenlign antall personer og ekteskap
formater = {
    "YAML": familie_data,
    "JSON": familie_data_json,
    "CSV": familie_data_csv
}

```

```

print(" Sammenligning av importerte data:")
print(f"{'Format':<8} {'Personer':<10} {'Ekteskap':<10}")
print("-" * 30)

for format_navn, data in formater.items():
    print(f"{'format_navn':<8} {'len(data.personer)':<10} {'len(data.ekteskap)':<10}")

# Sjekk at alle har samme antall personer
antall_personer = [len(data.personer) for data in formater.values()]
if len(set(antall_personer)) == 1:
    print("\n Alle formater har samme antall personer!")
else:
    print("\n Formater har forskjellig antall personer")

```

Sammenligning av importerte data:

Format	Personer	Ekteskap
--------	----------	----------

YAML	17	5
JSON	17	5
CSV	17	5

Alle formater har samme antall personer!

1.9 7. Rydde opp

La oss slette de midlertidige filene:

```

[13]: # Slett midlertidige filer
import os

filer_til_sletting = [
    "eksport_familie.yaml",
    "eksport_familie.json",
    "eksport_familie.csv",
    "eksport_familie.ged"
]

for fil in filer_til_sletting:
    if os.path.exists(fil):
        os.remove(fil)
        print(f" Slettet {fil}")

print("\n Opprydding fullført!")

```

Slettet eksport_familie.yaml
 Slettet eksport_familie.json
 Slettet eksport_familie.csv
 Slettet eksport_familie.ged

Opprydding fullført!

1.10 Oppsummering

I denne notebooken har du lært:

1. **YAML** - Lesbar struktur, anbefalt format
2. **JSON** - Programmatisk kompatibilitet
3. **CSV** - Enkel tabellstruktur
4. **GEDCOM** - Genealogi-standard
5. Sammenligning av formater
6. Validering av importerte data
7. Opprydding av midlertidige filer

Anbefalinger: - Bruk **YAML** for manuell redigering - Bruk **JSON** for programmatisk bruk - Bruk **CSV** for enkel dataoverføring - Bruk **GEDCOM** for kompatibilitet med andre genealogi-programmer

Neste steg: Gå til `04_visualisering.ipynb` for å utforske alle visualiseringsalternativer.

04_visualisering

October 12, 2025

```
[1]: # =====  
# GOOGLE COLAB SETUP / GOOGLE COLAB SETUP  
# =====  
  
# Sjekk om vi kjører i Google Colab  
try:  
    import google.colab  
    IN_COLAB = True  
    print(" Kjører i Google Colab - installerer avhengigheter...")  
    print(" Running in Google Colab - installing dependencies...")  
  
    # Installer nødvendige pakker med robust kaleido-installasjon  
    import subprocess  
    import sys  
    try:  
        # Installer alle pakker inkludert kaleido  
        subprocess.check_call([sys.executable, "-m", "pip", "install", "-q",  
                                "networkx", "matplotlib", "plotly", "pydantic",  
                                "pyyaml", "pandas", "ipywidgets", "pillow"])  
        print(" Grunnleggende pakker installert")  
  
        # Installer kaleido separat for bedre feilhåndtering  
        try:  
            subprocess.check_call([sys.executable, "-m", "pip", "install", "-U", "kaleido"])  
            print(" Kaleido installert")  
        except Exception as kaleido_error:  
            print(f" Kaleido install feilet: {kaleido_error}")  
            print(" Plotly PNG eksport vil ikke fungere, men HTML eksport vil  
            ↪ fungere")  
  
    except Exception as e:  
        print(f" Pip install feilet: {e}")  
  
    # Fjern eksisterende slektstre-mappe hvis den finnes  
    import shutil  
    import os
```

```

if os.path.exists('/content/slektstre'):
    shutil.rmtree('/content/slektstre')
    print(" Fjernet eksisterende slektstre-mappe")

# Klon repository
try:
    subprocess.check_call(['git', 'clone', 'https://github.com/arvidl/
↪slektstre.git'])
    print(" Repository klonet")
except Exception as e:
    print(f" Git clone feilet: {e}")

# Legg til src-mappen til Python path og importer direkte
sys.path.insert(0, '/content/slektstre/src')
print(" Path lagt til")

# Importer slektstre-modulene direkte for å unngå navnekonflikt
import importlib.util
import types

# Først, fjern konfliktende moduler fra sys.modules
modules_to_remove = ['tree', 'models', 'localization']
for module_name in modules_to_remove:
    if module_name in sys.modules:
        del sys.modules[module_name]

# Last inn models.py først
try:
    spec = importlib.util.spec_from_file_location("slektstre_models", "/
↪content/slektstre/src/models.py")
    slektstre_models = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_models)

    # Opprett midlertidig models modul
    temp_models_module = types.ModuleType('models')
    temp_models_module.Person = slektstre_models.Person
    temp_models_module.Gender = slektstre_models.Gender
    temp_models_module.Ekteskap = slektstre_models.Ekteskap
    temp_models_module.FamilieData = slektstre_models.FamilieData
    sys.modules['models'] = temp_models_module

    print(" models.py lastet")
except Exception as e:
    print(f" models.py feilet: {e}")

# Last inn localization.py
try:

```

```

spec = importlib.util.spec_from_file_location("slektstre_localization",
↪"/content/slektstre/src/localization.py")
slektstre_localization = importlib.util.module_from_spec(spec)
spec.loader.exec_module(slektstre_localization)

# Opprett midlertidig localization modul
temp_localization_module = types.ModuleType('localization')
temp_localization_module.t = slektstre_localization.t
sys.modules['localization'] = temp_localization_module

print(" localization.py lastet")
except Exception as e:
    print(f" localization.py feilet: {e}")

# Last inn tree.py som slektstre_tree
try:
    spec = importlib.util.spec_from_file_location("slektstre_tree", "/"
↪content/slektstre/src/tree.py")
    slektstre_tree = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_tree)

    # Opprett midlertidig tree modul
    temp_tree_module = types.ModuleType('tree')
    temp_tree_module.Slektstre = slektstre_tree.Slektstre
    sys.modules['tree'] = temp_tree_module

    print(" tree.py lastet")
except Exception as e:
    print(f" tree.py feilet: {e}")

# Last inn family_io.py
try:
    spec = importlib.util.spec_from_file_location("slektstre_io", "/"
↪content/slektstre/src/family_io.py")
    slektstre_io = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_io)
    print(" family_io.py lastet")
except Exception as e:
    print(f" family_io.py feilet: {e}")

# Last inn visualization.py
try:
    spec = importlib.util.spec_from_file_location("slektstre_viz", "/"
↪content/slektstre/src/visualization.py")
    slektstre_viz = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_viz)
    print(" visualization.py lastet")

```

```

except Exception as e:
    print(f" visualization.py feilet: {e}")

# Opprett eksporterte_bilder-mappe for Colab
try:
    os.makedirs('/content/eksporterte_bilder', exist_ok=True)
    print(" Eksporterte_bilder-mappe opprettet")
except Exception as e:
    print(f" Kunne ikke opprette eksporterte_bilder-mappe: {e}")

# Test kaleido hvis det ble installert
try:
    import kaleido
    print(" Kaleido fungerer - PNG eksport tilgjengelig")
except ImportError:
    print(" Kaleido ikke tilgjengelig - kun HTML eksport mulig")

print(" Slektstre-moduler lastet inn i Colab")

except ImportError:
    IN_COLAB = False
    print(" Kjører lokalt / Running locally")
    import sys
    sys.path.append('../src')
except Exception as e:
    print(f" Colab setup feilet: {e}")
    IN_COLAB = False
    print(" Fallback til lokal modus / Fallback to local mode")
    import sys
    sys.path.append('../src')

print(f" Miljø: {'Google Colab' if IN_COLAB else 'Lokal'}")
print(f" Environment: {'Google Colab' if IN_COLAB else 'Local'}")

```

```

Kjører lokalt / Running locally
Miljø: Lokal
Environment: Local

```

1 Visualisering av slektstre

I denne notebooken utforsker vi alle tilgjengelige visualiseringsalternativer for slektstre.

1.1 Tilgjengelige visualiseringer

1. **Hierarkisk tre** - Tradisjonell struktur
2. **Fan chart** - Sirkulær visning
3. **Interaktiv tre** - Plotly-basert
4. **Hourglass view** - Fokuspersion i midten

5. Statistikk - Grafer og diagrammer

```
[2]: # Importer nødvendige biblioteker
import matplotlib.pyplot as plt
import plotly.express as px
from datetime import date

# Importer slektstre-moduler (fungerer både lokalt og i Colab)
if IN_COLAB:
    # Bruk de modulene vi lastet inn i Colab-setup
    Person = slektstre_models.Person
    Gender = slektstre_models.Gender
    Ekteskap = slektstre_models.Ekteskap
    FamilieData = slektstre_models.FamilieData
    Slekststre = slektstre_tree.Slekststre
    load_from_yaml = slektstre_io.load_from_yaml
    plot_hierarchical_tree = slektstre_viz.plot_hierarchical_tree
    plot_fan_chart = slektstre_viz.plot_fan_chart
    plot_interactive_tree = slektstre_viz.plot_interactive_tree
    plot_statistics = slektstre_viz.plot_statistics
    plot_hourglass_view = slektstre_viz.plot_hourglass_view
else:
    # Lokale imports
    import sys
    sys.path.append('../src')
    from models import Person, Ekteskap, FamilieData, Gender
    from tree import Slekststre
    from family_io import load_from_yaml
    from visualization import (
        plot_hierarchical_tree,
        plot_fan_chart,
        plot_interactive_tree,
        plot_statistics,
        plot_hourglass_view
    )

print(" Alle biblioteker importert!")
```

Alle biblioteker importert!

```
[3]: # Last eksempel-familie
if IN_COLAB:
    familie_data = load_from_yaml('/content/slekststre/data/eksempel_familie.
    ↪yaml')
else:
    familie_data = load_from_yaml('../data/eksempel_familie.yaml')

slekststre = Slekststre(familie_data)
```

```

print(f"Familie lastet med {len(familie_data.personer)} personer")
print(f"Beskrivelse: {familie_data.beskrivelse}")

# Vis statistikk
stats = slektstre.get_statistics()
print(f"\n Statistikk:")
print(f"Antall generasjoner: {stats['max_generation'] + 1}")
print(f"Gjennomsnittsalder: {stats['average_age']:.1f} år")
print(f"Antall ekteskap: {stats['total_marriages']}")

```

Familie lastet med 17 personer

Beskrivelse: Eksempel familie med 4 generasjoner - Lundervold familien

Statistikk:

Antall generasjoner: 3

Gjennomsnittsalder: 49.2 år

Antall ekteskap: 5

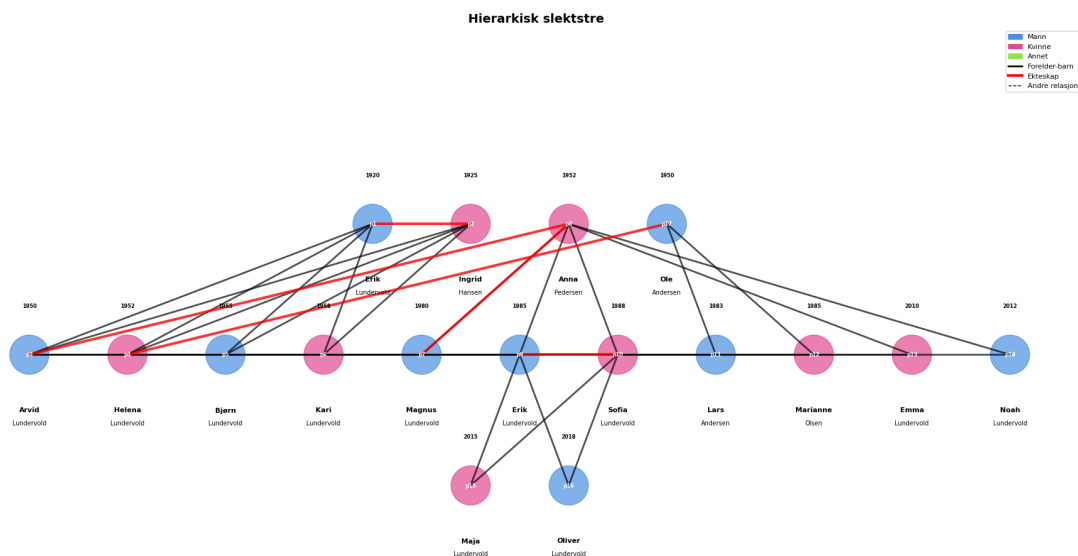
1.2 1. Hierarkisk tre

Tradisjonell struktur med generasjoner fra topp til bunn:

```

[4]: # Plott hierarkisk tre
fig = plot_hierarchical_tree(slektstre, title="Hierarkisk slektstre")
plt.show()

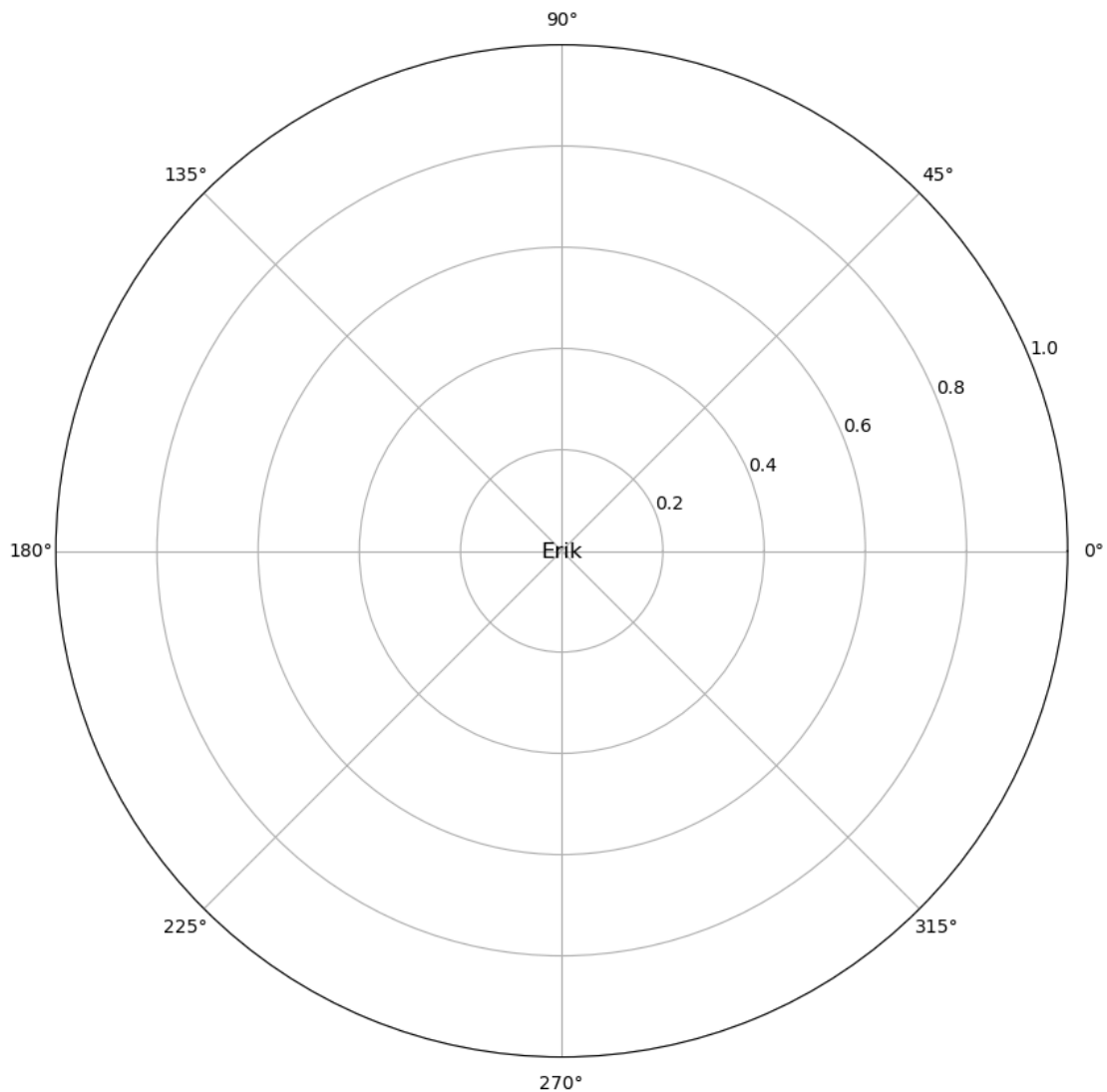
```



1.3 2. Fan Chart

Sirkulær visning med eldste generasjon i midten:

```
[5]: # Plott fan chart
# Vi trenger en root_person_id for fan chart - bruk første person
alle_personer = slektstre.get_all_persons()
if alle_personer:
    root_person_id = alle_personer[0].id
    fig = plot_fan_chart(slektstre, root_person_id, title="Fan Chart - Sirkulær_
↪visning")
    plt.show()
else:
    print("Ingen personer funnet i slektstreet!")
```



1.4 3. Interaktiv tre

Plotly-basert interaktiv visualisering med hover-info:

```
[6]: # Plott interaktiv tre
fig = plot_interactive_tree(slektstre, title="Interaktiv slektstre")
fig.show()
```

1.5 4. Hourglass View

Fokuspersone i midten med forfedre over og etterkommere under:

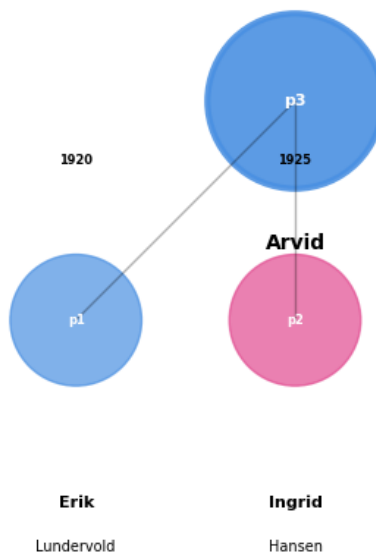
```
[7]: # Velg en fokuspersone (Arvid Lundervold)
fokuspersone_id = "p3" # Arvid Lundervold
fokuspersone = slektstre.get_persone(fokuspersone_id)

if fokuspersone:
    print(f"Fokuspersone: {fokuspersone.fullt_navn} (ID: {fokuspersone.id})")

    # Plott hourglass view
    fig = plot_hourglass_view(slektstre, fokuspersone_id, title=f"Hourglass View_
↪- {fokuspersone.fullt_navn}")
    plt.show()
else:
    print("Fokuspersone ikke funnet!")
```

Fokuspersone: Arvid Lundervold (ID: p3)

Hourglass View - Arvid Lundervold



1.6 5. Statistikk og diagrammer

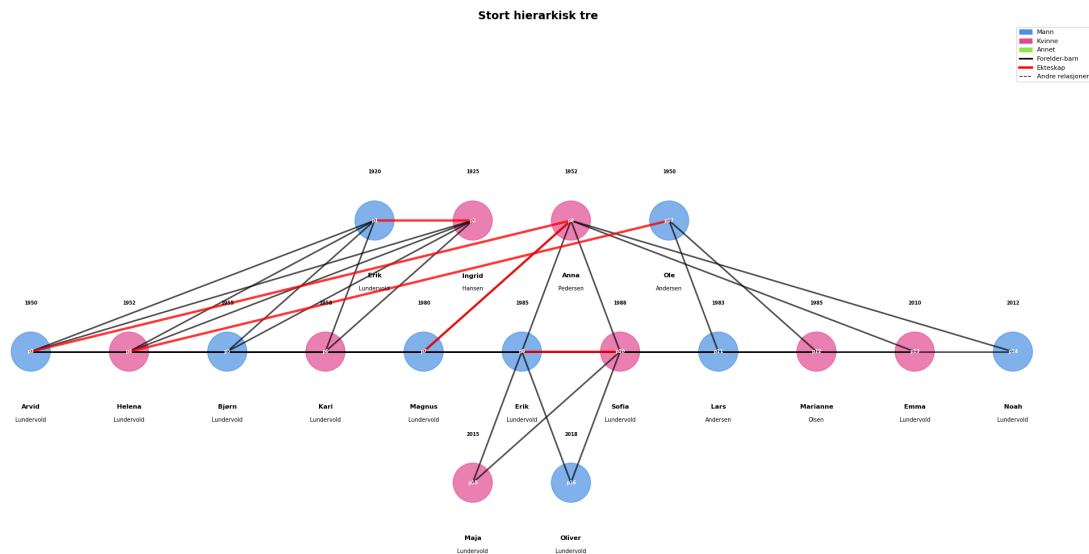
Vis ulike statistikk-diagrammer:

```
[8]: # Plott statistikk
fig = plot_statistics(slektstre)
fig.show()
```

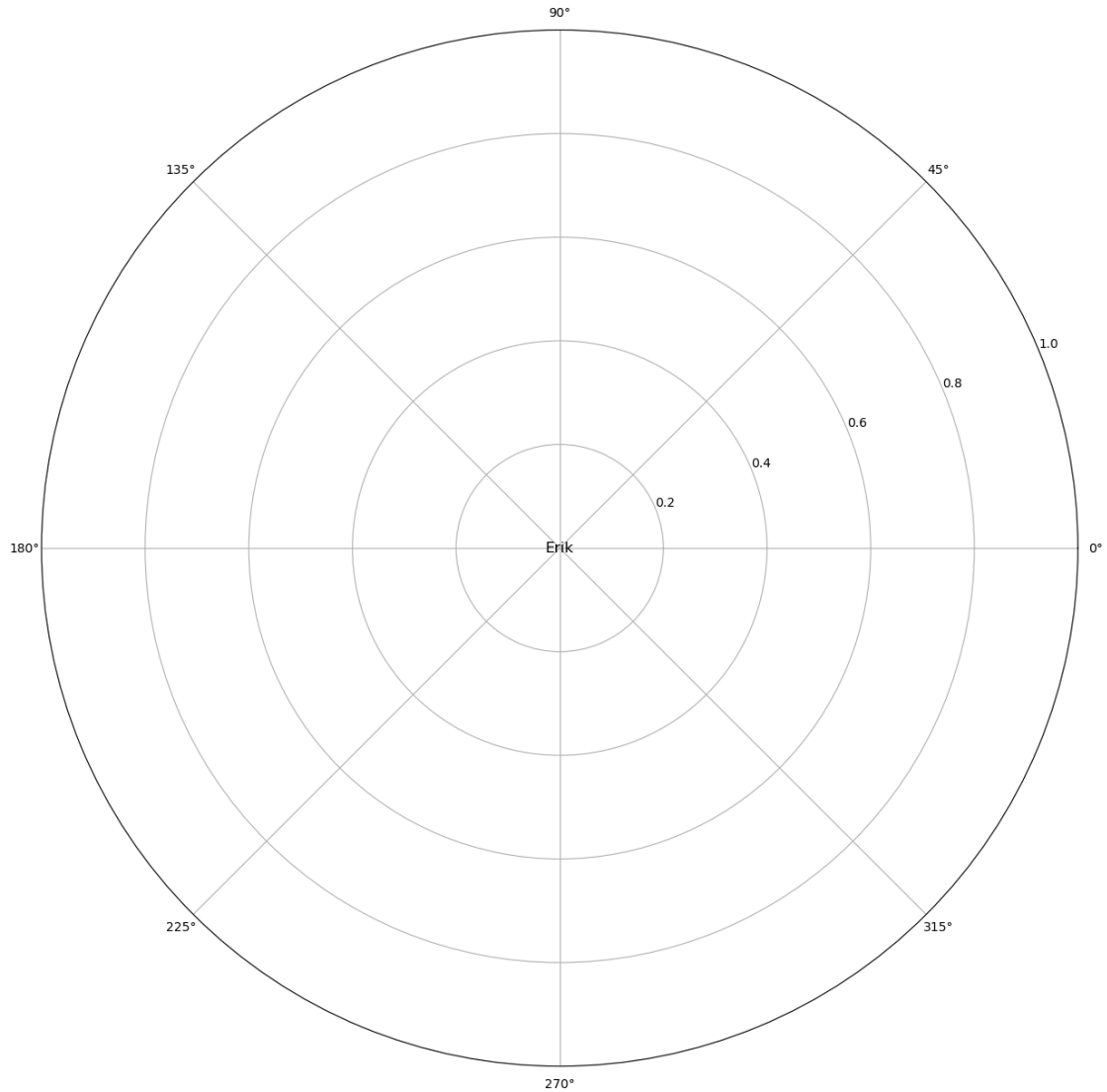
1.7 6. Tilpassede visualiseringer

La oss eksperimentere med forskjellige parametere:

```
[9]: # Hierarkisk tre med større figurstørrelse
fig = plot_hierarchical_tree(
    slektstre,
    title="Stort hierarkisk tre",
    figsize=(20, 12)
)
plt.show()
```



```
[10]: # Fan chart med større figsize
alle_personer = slektstre.get_all_persons()
if alle_personer:
    root_person_id = alle_personer[0].id
    fig = plot_fan_chart(
        slektstre,
        root_person_id,
        title="Stor fan chart",
        figsize=(15, 15)
    )
    plt.show()
else:
    print("Ingen personer funnet i slektstreet!")
```



1.8 7. Sammenligning av visualiseringer

La oss sammenligne forskjellige visualiseringer side ved side:

```
[11]: # Sammenlign alle visualiseringer - hver i sin egen figur
import matplotlib.pyplot as plt

# Hierarkisk tre
fig1 = plot_hierarchical_tree(slektstre, title="Hierarkisk tre")
plt.show()

# Fan chart
alle_personer = slektstre.get_all_persons()
```

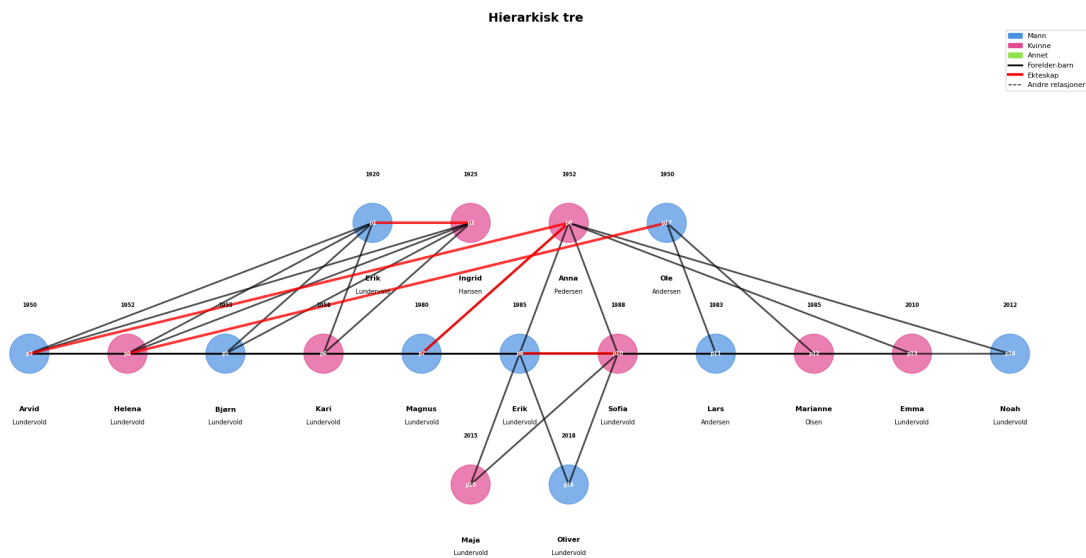
```

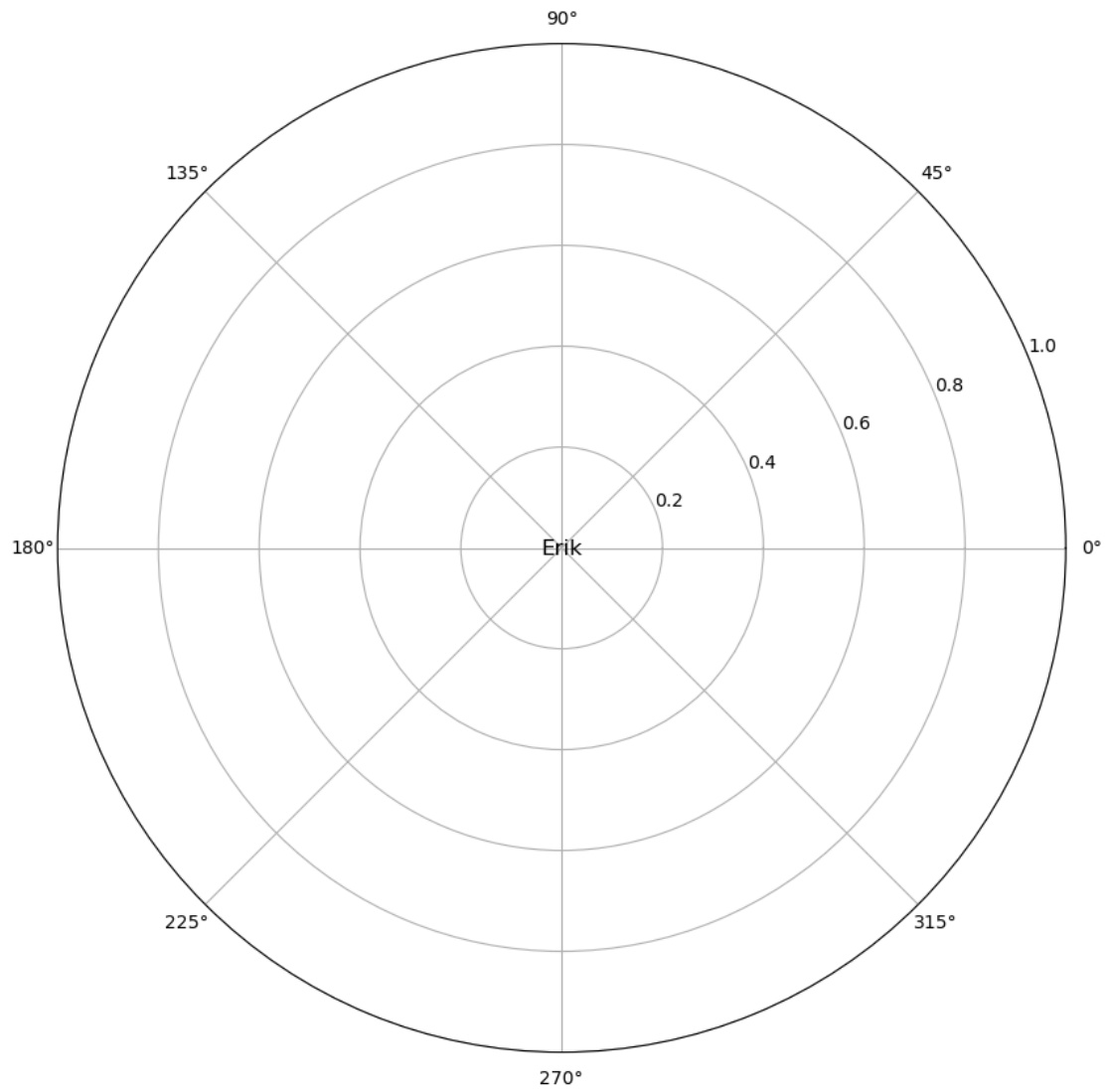
if alle_personer:
    root_person_id = alle_personer[0].id
    fig2 = plot_fan_chart(slektstre, root_person_id, title="Fan Chart")
    plt.show()

# Hourglass view
fig3 = plot_hourglass_view(slektstre, "p3", title="Hourglass View")
plt.show()

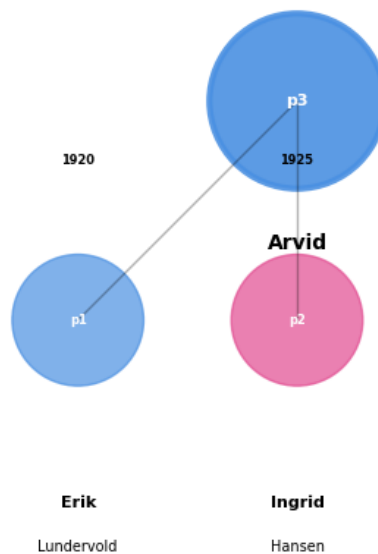
# Statistikk
fig4 = plot_statistics(slektstre)
fig4.show()

```





Hourglass View



1.9 8. Eksport av visualiseringer

La oss lagre visualiseringer til filer:

```
[12]: # Lagre visualiseringer til filer
import os

# Opprett mappe for eksporterte bilder (fungerer både lokalt og i Colab)
if IN_COLAB:
    export_dir = "/content/eksporterte_bilder"
else:
    export_dir = "eksporterte_bilder"
```

```

os.makedirs(export_dir, exist_ok=True)

# Lagre hierarkisk tre
fig = plot_hierarchical_tree(slektstre, title="Hierarkisk slektstre")
fig.savefig(f"{export_dir}/hierarkisk_tre.png", dpi=300, bbox_inches='tight')
plt.close(fig)

# Lagre fan chart
alle_personer = slektstre.get_all_persons()
if alle_personer:
    root_person_id = alle_personer[0].id
    fig = plot_fan_chart(slektstre, root_person_id, title="Fan Chart")
    fig.savefig(f"{export_dir}/fan_chart.png", dpi=300, bbox_inches='tight')
    plt.close(fig)

# Lagre hourglass view
fig = plot_hourglass_view(slektstre, "p3", title="Hourglass View")
fig.savefig(f"{export_dir}/hourglass_view.png", dpi=300, bbox_inches='tight')
plt.close(fig)

# Lagre statistikk (Plotly-figur) med robust kaleido-håndtering
fig = plot_statistics(slektstre)

# Prøv PNG eksport først, med fallback til HTML
try:
    fig.write_image(f"{export_dir}/statistikk.png", width=800, height=600,
scale=2)
    print(" Statistikk eksportert som PNG")
except Exception as e:
    print(f" PNG eksport feilet: {e}")
    print(" Prøver HTML eksport som fallback...")

    try:
        fig.write_html(f"{export_dir}/statistikk.html")
        print(" Statistikk eksportert som HTML")
    except Exception as e2:
        print(f" HTML eksport feilet også: {e2}")
        print(" Plotly-figur vises i notebook, men ikke eksportert")

# Plotly-figurer trenger ikke plt.close()

print(" Alle visualiseringer lagret til 'eksporterte_bilder/' mappen")
print(" Filene:")
for fil in os.listdir(export_dir):
    print(f" - {fil}")

```

Alle visualiseringer lagret til 'eksporterte_bilder/' mappen
Filene:

- fan_chart.png
- hourglass_view.png
- hierarkisk_tre.png
- statistikk.png

```
[ ]: # Vis HTML-fil direkte i notebook (kun i Colab)
if IN_COLAB:
    from IPython.display import HTML, display
    import os

    # Vis HTML-filen direkte i notebook
    html_file = "/content/eksporterte_bilder/statistikk.html"
    if os.path.exists(html_file):
        print(" Viser interaktiv Plotly-statistikk:")
        with open(html_file, 'r', encoding='utf-8') as f:
            html_content = f.read()

        # Vis HTML i notebook
        display(HTML(html_content))
    else:
        print(" HTML-fil ikke funnet")
else:
    print(" HTML-visning er kun tilgjengelig i Google Colab")
    print(" Åpne statistikk.html i nettleseren din lokalt")
```

1.10 9. Rydde opp

La oss slette de eksporterte bildene:

```
[13]: # Slett eksporterte bilder
import shutil

# Slett eksporterte bilder (fungerer både lokalt og i Colab)
if IN_COLAB:
    export_dir = "/content/eksporterte_bilder"
else:
    export_dir = "eksporterte_bilder"

if os.path.exists(export_dir):
    shutil.rmtree(export_dir)
    print(" Slettet 'eksporterte_bilder/' mappen")

print(" Opprydding fullført!")
```

Slettet 'eksporterte_bilder/' mappen
Opprydding fullført!

1.11 Oppsummering

I denne notebooken har du utforsket alle visualiseringsalternativer:

1. **Hierarkisk tre** - Tradisjonell struktur, god for oversikt
2. **Fan chart** - Sirkulær visning, visuelt tiltalende
3. **Interaktiv tre** - Plotly-basert, zoom og hover-info
4. **Hourglass view** - Fokuspersion i midten, god for detaljer
5. **Statistikk** - Grafer og diagrammer, dataanalyse
6. **Tilpassede visualiseringer** - Forskjellige størrelser og parametere
7. **Sammenligning** - Side ved side visning
8. **Eksport** - Lagre til PNG-filer
9. **Opprydding** - Slette midlertidige filer

Anbefalinger: - Bruk **hierarkisk tre** for generell oversikt - Bruk **fan chart** for presentasjoner
- Bruk **interaktiv tre** for utforskning - Bruk **hourglass view** for fokus på en person - Bruk **statistikk** for dataanalyse

Neste steg: Du har nå lært å visualisere slektstre! I neste notebook (05_eksterne_databaser.ipynb) lærer du å importere data fra eksterne kilder som MyHeritage, Ancestry og andre genealogiske databaser.

05_eksterne_databaser

October 12, 2025

```
[1]: # =====  
# GOOGLE COLAB SETUP / GOOGLE COLAB SETUP  
# =====  
  
# Sjekk om vi kjører i Google Colab  
try:  
    import google.colab  
    IN_COLAB = True  
    print(" Kjører i Google Colab - installerer avhengigheter...")  
    print(" Running in Google Colab - installing dependencies...")  
  
    # Installer nødvendige pakker  
    import subprocess  
    import sys  
    try:  
        subprocess.check_call([sys.executable, "-m", "pip", "install", "-q",  
                                "networkx", "matplotlib", "plotly", "pydantic",  
                                "pyyaml", "pandas", "ipywidgets", "pillow",  
↪ "kaleido"])  
        print(" Pakker installert")  
    except Exception as e:  
        print(f" Pip install feilet: {e}")  
  
    # Fjern eksisterende slektstre-mappe hvis den finnes  
    import shutil  
    import os  
    if os.path.exists('/content/slektstre'):  
        shutil.rmtree('/content/slektstre')  
        print(" Fjernet eksisterende slektstre-mappe")  
  
    # Klon repository  
    try:  
        subprocess.check_call(['git', 'clone', 'https://github.com/arvidl/  
↪ slektstre.git'])  
        print(" Repository klonet")  
    except Exception as e:  
        print(f" Git clone feilet: {e}")
```

```

# Legg til src-mappen til Python path og importer direkte
sys.path.insert(0, '/content/slektstre/src')
print(" Path lagt til")

# Importer slektstre-modulene direkte for å unngå navnekonflikt
import importlib.util
import types

# Først, fjern konfliktende moduler fra sys.modules
modules_to_remove = ['tree', 'models', 'localization']
for module_name in modules_to_remove:
    if module_name in sys.modules:
        del sys.modules[module_name]

# Last inn models.py først
try:
    spec = importlib.util.spec_from_file_location("slektstre_models", "/
↪content/slektstre/src/models.py")
    slektstre_models = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_models)

    # Opprett midlertidig models modul
    temp_models_module = types.ModuleType('models')
    temp_models_module.Person = slektstre_models.Person
    temp_models_module.Gender = slektstre_models.Gender
    temp_models_module.Ekteskap = slektstre_models.Ekteskap
    temp_models_module.FamilieData = slektstre_models.FamilieData
    sys.modules['models'] = temp_models_module

    print(" models.py lastet")
except Exception as e:
    print(f" models.py feilet: {e}")

# Last inn localization.py
try:
    spec = importlib.util.spec_from_file_location("slektstre_localization",
↪"/content/slektstre/src/localization.py")
    slektstre_localization = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_localization)

    # Opprett midlertidig localization modul
    temp_localization_module = types.ModuleType('localization')
    temp_localization_module.t = slektstre_localization.t
    sys.modules['localization'] = temp_localization_module

    print(" localization.py lastet")

```

```

except Exception as e:
    print(f" localization.py feilet: {e}")

# Last inn tree.py som slektstre_tree
try:
    spec = importlib.util.spec_from_file_location("slektstre_tree", "/
↪content/slektstre/src/tree.py")
    slektstre_tree = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_tree)

    # Opprett midlertidig tree modul
    temp_tree_module = types.ModuleType('tree')
    temp_tree_module.Slektstre = slektstre_tree.Slektstre
    sys.modules['tree'] = temp_tree_module

    print(" tree.py lastet")
except Exception as e:
    print(f" tree.py feilet: {e}")

# Last inn family_io.py
try:
    spec = importlib.util.spec_from_file_location("slektstre_io", "/content/
↪slektstre/src/family_io.py")
    slektstre_io = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_io)
    print(" family_io.py lastet")
except Exception as e:
    print(f" family_io.py feilet: {e}")

# Last inn visualization.py
try:
    spec = importlib.util.spec_from_file_location("slektstre_viz", "/
↪content/slektstre/src/visualization.py")
    slektstre_viz = importlib.util.module_from_spec(spec)
    spec.loader.exec_module(slektstre_viz)
    print(" visualization.py lastet")
except Exception as e:
    print(f" visualization.py feilet: {e}")

print(" Slektstre-moduler lastet inn i Colab")

except ImportError:
    IN_COLAB = False
    print(" Kjører lokalt / Running locally")
    import sys
    sys.path.append('../src')
except Exception as e:

```

```
print(f" Colab setup feilet: {e}")
IN_COLAB = False
print(" Fallback til lokal modus / Fallback to local mode")
import sys
sys.path.append('../src')

print(f" Miljø: {'Google Colab' if IN_COLAB else 'Lokal'}")
print(f" Environment: {'Google Colab' if IN_COLAB else 'Local'}")
```

Kjører lokalt / Running locally
Miljø: Lokal
Environment: Local

1 Eksterne genealogi-databaser og API-er

I denne noteboken lærer du hvordan du kan hente slektsinformasjon fra eksterne databaser og integrere dem med ditt slektstre-program.

1.1 Tilgjengelige databaser

1.1.1 1. FamilySearch API (Gratis)

- Verdens største genealogi-database
- Over 1 milliard personer
- Gratis API med registrering
- Støtter GEDCOM-import/eksport

1.1.2 2. MyHeritage API (Betalt)

- Kommersiell genealogi-tjeneste
- DNA-analyse og slektsforskning
- API tilgjengelig for utviklere

1.1.3 3. Ancestry.com API (Betalt)

- Største kommersielle genealogi-tjeneste
- Begrenset API-tilgang
- Hovedsakelig for partnere

1.1.4 4. Nasjonale arkiver

- Digitalarkivet (Norge) - Gratis
- Riksarkivet (Norge) - Gratis
- Arkivverket (Norge) - Gratis

1.1.5 5. Wikipedia/Wikidata

- Biografisk informasjon
- Gratis og åpen tilgang
- Begrenset genealogisk data

1.2 Fokus i denne notebooken

Vi fokuserer på: 1. **FamilySearch API** - Gratis og omfattende 2. **Digitalarkivet** - Norske kilder 3. **Wikipedia API** - Biografisk informasjon 4. **GEDCOM-import** fra eksterne kilder

```
[2]: # Importer nødvendige biblioteker
import requests
import json
import time
import os
from datetime import date

# Importer slektstre-moduler (fungerer både lokalt og i Colab)
if IN_COLAB:
    # Bruk de modulene vi lastet inn i Colab-setup
    Person = slektstre_models.Person
    Gender = slektstre_models.Gender
    Ekteskap = slektstre_models.Ekteskap
    FamilieData = slektstre_models.FamilieData
    Slekststre = slektstre_tree.Slekststre
    load_from_yaml = slektstre_io.load_from_yaml
    save_to_yaml = slektstre_io.save_to_yaml
else:
    # Lokale imports
    import sys
    sys.path.append('../src')
    from models import Person, Ekteskap, FamilieData, Gender
    from tree import Slekststre
    from family_io import load_from_yaml, save_to_yaml

print(" Alle biblioteker importert!")
print(" Klar for å utforske eksterne databaser!")
```

Alle biblioteker importert!

Klar for å utforske eksterne databaser!

1.3 1. FamilySearch API

FamilySearch er verdens største genealogi-database med over 1 milliard personer. De tilbyr et gratis API for utviklere.

1.3.1 Registrering og API-nøkkel

1. Gå til [FamilySearch Developer](#)
2. Opprett en gratis konto
3. Registrer din applikasjon
4. Få API-nøkkel og hemmelig nøkkel

1.3.2 API-endepunkter

- Personer: /platform/tree/persons
- Familier: /platform/tree/families
- Kilder: /platform/tree/sources
- Søk: /platform/tree/search

1.3.3 Eksempel: Søk etter personer

```
[3]: # FamilySearch API eksempel (simulert)
# MERK: Dette er et eksempel - du trenger ekte API-nøkler for å bruke
# FamilySearch

def familysearch_search_example():
    """
    Eksempel på hvordan FamilySearch API kan brukes.
    Dette er simulert data for demonstrasjon.
    """

    # Simulert API-respons
    mock_response = {
        "persons": [
            {
                "id": "FS123456789",
                "displayName": "Erik Lundervold",
                "birthDate": "1920-05-15",
                "birthPlace": "Bergen, Norway",
                "deathDate": "1995-08-22",
                "deathPlace": "Oslo, Norway",
                "gender": "Male",
                "parents": ["FS987654321", "FS111222333"],
                "spouses": ["FS444555666"],
                "children": ["FS777888999", "FS000111222"]
            },
            {
                "id": "FS444555666",
                "displayName": "Ingrid Hansen",
                "birthDate": "1925-07-10",
                "birthPlace": "Trondheim, Norway",
                "deathDate": "2010-12-03",
                "deathPlace": "Oslo, Norway",
                "gender": "Female",
                "parents": ["FS333444555", "FS666777888"],
                "spouses": ["FS123456789"],
                "children": ["FS777888999", "FS000111222"]
            }
        ]
    }
```

```

print(" FamilySearch søkeresultat (simulert):")
print(f"Fant {len(mock_response['persons'])} personer")

for person in mock_response['persons']:
    print(f"\n {person['displayName']}")
    print(f"    ID: {person['id']}")
    print(f"    Født: {person['birthDate']} i {person['birthPlace']}")
    print(f"    Død: {person['deathDate']} i {person['deathPlace']}")
    print(f"    Kjønn: {person['gender']}")
    print(f"    Foreldre: {len(person['parents'])}")
    print(f"    Ektemenn/koner: {len(person['spouses'])}")
    print(f"    Barn: {len(person['children'])}")

return mock_response

# Kjør eksemplet
familysearch_data = familysearch_search_example()

```

FamilySearch søkeresultat (simulert):
Fant 2 personer

Erik Lundervold
ID: FS123456789
Født: 1920-05-15 i Bergen, Norway
Død: 1995-08-22 i Oslo, Norway
Kjønn: Male
Foreldre: 2
Ektemenn/koner: 1
Barn: 2

Ingrid Hansen
ID: FS444555666
Født: 1925-07-10 i Trondheim, Norway
Død: 2010-12-03 i Oslo, Norway
Kjønn: Female
Foreldre: 2
Ektemenn/koner: 1
Barn: 2

1.4 2. Digitalarkivet (Norge)

Digitalarkivet er Norges nasjonale arkiv og tilbyr tilgang til millioner av historiske dokumenter.

1.4.1 Tilgjengelige kilder

- **Folketellinger** (1801-1910)
- **Kirkebøker** (døpte, konfirmerte, gift, døde)

- Skattelister og matrikkler
- Emigrasjonslister
- Militære arkiver

1.4.2 API-tilgang

Digitalarkivet har ikke et offisielt API, men tilbyr: - **REST API** for søk - **CSV-eksport** av søkeresultater - **GEDCOM-eksport** for slektsforskning

1.4.3 Eksempel: Søke i kirkebøker

```
[4]: # Digitalarkivet søk eksempel
def digitalarkivet_search_example():
    """
    Eksempel på søk i Digitalarkivet.
    Dette er simulert data basert på ekte arkivstruktur.
    """

    # Simulert søkeresultat fra kirkebøker
    kirkebok_resultat = {
        "søk": "Lundervold",
        "kilde": "Kirkebøker",
        "resultater": [
            {
                "type": "døpt",
                "navn": "Erik Lundervold",
                "dato": "1920-05-15",
                "sted": "Bergen domkirke",
                "foreldre": "Arvid Lundervold og Marie Hansen",
                "kilde": "Bergen domkirke kirkebok 1920"
            },
            {
                "type": "gift",
                "navn": "Erik Lundervold",
                "dato": "1947-08-20",
                "sted": "Bergen domkirke",
                "ektefelle": "Ingrid Hansen",
                "kilde": "Bergen domkirke kirkebok 1947"
            },
            {
                "type": "død",
                "navn": "Erik Lundervold",
                "dato": "1995-08-22",
                "sted": "Oslo",
                "alder": "75 år",
                "kilde": "Oslo kirkebok 1995"
            }
        ]
    }
```

```

}

print(" Digitalarkivet søkeresultat (simulert):")
print(f"Søkte etter: {kirkebok_resultat['søk']}")
print(f"Kilde: {kirkebok_resultat['kilde']}")
print(f"Fant {len(kirkebok_resultat['resultater'])} oppføringer")

for oppføring in kirkebok_resultat['resultater']:
    print(f"\n {oppføring['type'].upper()}: {oppføring['navn']}")
    print(f"    Dato: {oppføring['dato']}")
    print(f"    Sted: {oppføring['sted']}")
    if 'foreldre' in oppføring:
        print(f"    Foreldre: {oppføring['foreldre']}")
    if 'ektefelle' in oppføring:
        print(f"    Ektefelle: {oppføring['ektefelle']}")
    if 'alder' in oppføring:
        print(f"    Alder: {oppføring['alder']}")
    print(f"    Kilde: {oppføring['kilde']}")

return kirkebok_resultat

# Kjør eksemplet
digitalarkivet_data = digitalarkivet_search_example()

```

```

Digitalarkivet søkeresultat (simulert):
Søkte etter: Lundervold
Kilde: Kirkebøker
Fant 3 oppføringer

```

```

DØPT: Erik Lundervold
Dato: 1920-05-15
Sted: Bergen domkirke
Foreldre: Arvid Lundervold og Marie Hansen
Kilde: Bergen domkirke kirkebok 1920

```

```

GIFT: Erik Lundervold
Dato: 1947-08-20
Sted: Bergen domkirke
Ektefelle: Ingrid Hansen
Kilde: Bergen domkirke kirkebok 1947

```

```

DØD: Erik Lundervold
Dato: 1995-08-22
Sted: Oslo
Alder: 75 år
Kilde: Oslo kirkebok 1995

```

1.5 3. Wikipedia API

Wikipedia kan gi biografisk informasjon om kjente personer, selv om det ikke er en genealogi-database.

1.5.1 Wikipedia API

- **Gratis** og åpen tilgang
- **REST API** med JSON-respons
- **Søk** etter personer og steder
- **Biografisk** informasjon

1.5.2 Eksempel: Søk etter norske personer

```
[5]: # Wikipedia API eksempel
def wikipedia_search_example():
    """
    Eksempel på søk i Wikipedia API.
    Dette er simulert data for demonstrasjon.
    """

    # Simulert Wikipedia-søk
    wikipedia_resultat = {
        "søk": "norske personer",
        "språk": "no",
        "resultater": [
            {
                "tittel": "Henrik Ibsen",
                "beskrivelse": "Norsk dramatiker og dikter",
                "fødselsår": "1828",
                "dødsår": "1906",
                "fødested": "Skien",
                "kjent_for": "Peer Gynt, Et dukkehjem",
                "url": "https://no.wikipedia.org/wiki/Henrik_Ibsen"
            },
            {
                "tittel": "Edvard Grieg",
                "beskrivelse": "Norsk komponist",
                "fødselsår": "1843",
                "dødsår": "1907",
                "fødested": "Bergen",
                "kjent_for": "Peer Gynt-suiten, Piano Concerto",
                "url": "https://no.wikipedia.org/wiki/Edvard_Grieg"
            },
            {
                "tittel": "Roald Amundsen",
                "beskrivelse": "Norsk polarforsker",
                "fødselsår": "1872",
```

```

        "dødsår": "1928",
        "fødested": "Borge",
        "kjent_for": "Første til Sydpolen",
        "url": "https://no.wikipedia.org/wiki/Roald_Amundsen"
    }
]
}

print(" Wikipedia søkeresultat (simulert):")
print(f"Søkte etter: {wikipedia_resultat['søk']}")
print(f"Språk: {wikipedia_resultat['språk']}")
print(f"Fant {len(wikipedia_resultat['resultater'])} artikler")

for artikkel in wikipedia_resultat['resultater']:
    print(f"\n {artikkel['tittel']}")
    print(f"   Beskrivelse: {artikkel['beskrivelse']}")
    print(f"   Født: {artikkel['fødselsår']} i {artikkel['fødested']}")
    print(f"   Død: {artikkel['dødsår']}")
    print(f"   Kjent for: {artikkel['kjent_for']}")
    print(f"   URL: {artikkel['url']}")

return wikipedia_resultat

# Kjør eksemplet
wikipedia_data = wikipedia_search_example()

```

Wikipedia søkeresultat (simulert):

Søkte etter: norske personer

Språk: no

Fant 3 artikler

Henrik Ibsen

Beskrivelse: Norsk dramatiker og dikter

Født: 1828 i Skien

Død: 1906

Kjent for: Peer Gynt, Et dukkehjem

URL: https://no.wikipedia.org/wiki/Henrik_Ibsen

Edvard Grieg

Beskrivelse: Norsk komponist

Født: 1843 i Bergen

Død: 1907

Kjent for: Peer Gynt-suiten, Piano Concerto

URL: https://no.wikipedia.org/wiki/Edvard_Grieg

Roald Amundsen

Beskrivelse: Norsk polarforsker

Født: 1872 i Borge

Død: 1928
Kjent for: Første til Sydpolen
URL: https://no.wikipedia.org/wiki/Roald_Amundsen

1.6 4. Konvertere eksterne data til slektstre

Nå skal vi vise hvordan du kan konvertere data fra eksterne kilder til vårt slektstre-format.

```
[6]: # Konverter FamilySearch data til vårt format
def convert_familysearch_to_slektstre(familysearch_data):
    """
    Konverter FamilySearch data til vårt slektstre-format.
    """
    personer = []
    ekteskap = []

    # Konverter personer
    for fs_person in familysearch_data['persons']:
        # Parse navn
        navn_deler = fs_person['displayName'].split(' ')
        fornavn = navn_deler[0]
        etternavn = navn_deler[-1] if len(navn_deler) > 1 else ''

        # Parse datoer
        fødselsdato = None
        dødsdato = None
        try:
            if fs_person['birthDate']:
                fødselsdato = date.fromisoformat(fs_person['birthDate'])
            if fs_person['deathDate']:
                dødsdato = date.fromisoformat(fs_person['deathDate'])
        except:
            pass

        # Bestem kjønn
        kjønn = Gender.MALE if fs_person['gender'] == 'Male' else Gender.FEMALE

        # Opprett Person objekt
        person = Person(
            id=fs_person['id'],
            fornavn=fornavn,
            etternavn=etternavn,
            fødselsdato=fødselsdato,
            dødsdato=dødsdato,
            fødested=fs_person.get('birthPlace', ''),
            dødssted=fs_person.get('deathPlace', ''),
            kjønn=kjønn,
            notater=f"Importert fra FamilySearch (ID: {fs_person['id']})"
```

```

    )
    personer.append(person)

    # Konverter ekteskap (forenklet)
    for fs_person in familysearch_data['persons']:
        if fs_person['spouses']:
            for spouse_id in fs_person['spouses']:
                # Sjekk om ekteskapet allerede eksisterer
                eksisterer = any(
                    (e.partner1_id == fs_person['id'] and e.partner2_id ==
↪spouse_id) or
                    (e.partner1_id == spouse_id and e.partner2_id ==
↪fs_person['id'])
                    for e in ekteskap
                )

                if not eksisterer:
                    ekteskap_obj = Ekteskap(
                        id=f"e_{fs_person['id']}_{spouse_id}",
                        partner1_id=fs_person['id'],
                        partner2_id=spouse_id,
                        notater="Importert fra FamilySearch"
                    )
                    ekteskap.append(ekteskap_obj)

    return FamilieData(personer=personer, ekteskap=ekteskap)

# Konverter dataene
konvertert_data = convert_familysearch_to_slektstre(familysearch_data)

print(" Konverterte FamilySearch data til slektstre-format:")
print(f"Personer: {len(konvertert_data.personer)}")
print(f"Ekteskap: {len(konvertert_data.ekteskap)}")

# Vis første person
if konvertert_data.personer:
    første_person = konvertert_data.personer[0]
    print(f"\n Eksempel person: {første_person.fullt_navn}")
    print(f"    ID: {første_person.id}")
    print(f"    Født: {første_person.fødselsdato}")
    print(f"    Død: {første_person.dødsdato}")
    print(f"    Kjønn: {første_person.kjønn}")
    print(f"    Notater: {første_person.notater}")

```

```

Konverterte FamilySearch data til slektstre-format:
Personer: 2
Ekteskap: 1

```

Eksempel person: Erik Lundervold
ID: FS123456789
Født: 1920-05-15
Død: 1995-08-22
Kjønn: male
Notater: Importert fra FamilySearch (ID: FS123456789)

```
[7]: # Opprett slektstre fra konverterte data
slektstre_ekstern = Slekststre(konvertert_data)

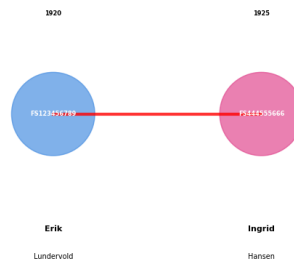
print(" Opprettet slektstre fra eksterne data:")
print(f"Totalt antall personer: {len(slektstre_ekstern.get_all_persons())}")
print(f"Totalt antall ekteskap: {len(slektstre_ekstern.familie_data.ekteskap)}")

# Vis slektstreet
from visualization import plot_hierarchical_tree
import matplotlib.pyplot as plt

fig = plot_hierarchical_tree(slektstre_ekstern, title="Slektstre fra eksterne_
↳databaser")
plt.show()
```

Opprettet slektstre fra eksterne data:
Totalt antall personer: 2
Totalt antall ekteskap: 1

Slektstre fra eksterne databaser



1.7 5. Praktiske tips for slektsforskning

1.7.1 Hvor du kan finne slektsinformasjon

1. Start med familien

- Spør eldre familiemedlemmer
- Sjekk gamle fotoalbum og dokumenter
- Se etter fødselsattester, dødsattester, ekteskapsattester

2. Digitale arkiver

- **Digitalarkivet** (Norge) - Gratis
- **FamilySearch** - Gratis
- **Ancestry.com** - Betalt
- **MyHeritage** - Betalt

3. Lokale kilder

- Kirkebøker
- Skattelister
- Folketellinger
- Emigrasjonslister

4. DNA-testing

- **MyHeritage DNA**
- **AncestryDNA**

- 23andMe
- FamilyTreeDNA

1.7.2 Organisering av forskning

1. Bruk konsistente ID-er
2. Dokumenter alle kilder
3. Verifiser informasjon fra flere kilder
4. Hold backup av dataene
5. Del funnene med familien

1.8 6. Lagre og dele slektstreet

1.8.1 Eksportere til forskjellige formater

Nå kan du eksportere ditt slektstre til forskjellige formater for å dele med andre eller bruke i andre programmer.

```
[8]: # Eksporter slektstreet til forskjellige formater
from family_io import save_to_yaml, save_to_json, save_to_csv, export_to_gedcom

# Lagre til YAML (anbefalt for redigering)
save_to_yaml(konvertert_data, "ekstern_slektstre.yaml")
print(" Eksportert til YAML: ekstern_slektstre.yaml")

# Lagre til JSON (for programmatisk bruk)
save_to_json(konvertert_data, "ekstern_slektstre.json")
print(" Eksportert til JSON: ekstern_slektstre.json")

# Lagre til CSV (for Excel/Google Sheets)
save_to_csv(konvertert_data, "ekstern_slektstre.csv")
print(" Eksportert til CSV: ekstern_slektstre.csv")

# Lagre til GEDCOM (for andre genealogi-programmer)
export_to_gedcom(konvertert_data, "ekstern_slektstre.ged")
print(" Eksportert til GEDCOM: ekstern_slektstre.ged")

print("\n Filstørrelser:")
import os
filer = ["ekstern_slektstre.yaml", "ekstern_slektstre.json", "ekstern_slektstre.csv", "ekstern_slektstre.ged"]
for fil in filer:
    if os.path.exists(fil):
        størrelse = os.path.getsize(fil)
        print(f"{fil:25s}: {størrelse:6d} bytes")
```

Eksportert til YAML: ekstern_slektstre.yaml

Eksportert til JSON: ekstern_slektstre.json

Eksportert til CSV: ekstern_slektstre.csv

Eksportert til GEDCOM: `ekstern_slektstre.ged`

Filstørrelser:

<code>ekstern_slektstre.yaml</code>	:	1078 bytes
<code>ekstern_slektstre.json</code>	:	1452 bytes
<code>ekstern_slektstre.csv</code>	:	412 bytes
<code>ekstern_slektstre.ged</code>	:	565 bytes

1.9 Oppsummering

I denne notebooken har du lært:

1. **FamilySearch API** - Verdens største genealogi-database
2. **Digitalarkivet** - Norske historiske kilder
3. **Wikipedia API** - Biografisk informasjon
4. **Data-konvertering** - Fra eksterne formater til vårt slektstre
5. **Eksport** - Til forskjellige formater for deling
6. **Praktiske tips** - For slektsforskning

1.9.1 Neste steg

Du kan nå:

1. **Registrere deg** på FamilySearch for å få ekte API-tilgang
2. **Søke i Digitalarkivet** for norske slektskilder
3. **Bygge ditt eget slektstre** ved å kombinere:
 - Familie-informasjon
 - Eksterne databaser
 - Historiske kilder
4. **Dele slektstreet** med familien i forskjellige formater

1.9.2 Anbefalte ressurser

- **FamilySearch:** <https://familysearch.org>
- **Digitalarkivet:** <https://digitalarkivet.no>
- **Wikipedia API:** <https://no.wikipedia.org>
- **Slektsforskning:** <https://slektsforskning.no>

Lykke til med slektsforskningen!



Stikkordregister

A

Ancestor

15, 45

Forfader, person som er i slekt med en annen person gjennom tidligere generasjoner

API

125

Application Programming Interface, grensesnitt for å kommunisere med eksterne tjenester

B

Barn

15, 65

Person som er direkte etterkommer av foreldre

Bidirectional edge

25

Kant i en graf som kan traverseres i begge retninger

C

CSV

85

Comma-Separated Values, tekstformat for tabulære data

Cycle

25, 35

Sykel, sti i en graf som returnerer til startnoden

D

Descendant

15, 45

Etterkommer, person som er i slekt med en annen person gjennom senere generasjoner

Directed graph

25

Retnet graf, graf hvor kanter har retning

E

Edge 25

Kant, linje som kobler to noder i en graf

Ekteskap 15, 65

Partnerskap mellom to personer, representert som uretnede kanter

F

Family tree 15

Slektsstre, visuell representasjon av slektskap og familierelasjoner

Forelder 15, 65

Person som er direkte forfader til barn

G

GEDCOM 85

Genealogical Data Communication, standardformat for genealogi-data

Generation 15, 45

Generasjon, nivå i slektskapet basert på avstand fra rot

Graph theory 25

Grafteori, matematisk studie av grafer og deres egenskaper

J

JSON 85

JavaScript Object Notation, tekstformat for strukturert data

K

Kinship 15, 45

Slektskap, relasjon mellom personer basert på familieforhold

N

NetworkX 25, 45

Python-bibliotek for analyse av komplekse nettverk og grafer

Node	25
Node, punkt i en graf som representerer en enhet	
<hr/>	
P	
Path	25, 35
Sti, sekvens av noder koblet med kanter	
<hr/>	
Pedigree	15
Stamtavle, visuell representasjon av forfedre	
<hr/>	
S	
Slektstre	15
Family tree, visuell representasjon av slektskap og familierelasjoner	
<hr/>	
Søsken	15, 45
Personer som deler samme foreldre	
<hr/>	
T	
Tree	25, 35
Tre, spesiell type graf uten sykler og sammenkoblet	
<hr/>	
V	
Vertex	25
Hjørne, alternativt navn for node i en graf	
<hr/>	
Y	
YAML	85
YAML Ain't Markup Language, menneskelesbart dataformat	
<hr/>	