



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)

11 Oktober 2025

Versjon 1.0



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
Alfabetskifte 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

[SPILL AV PODCASTEN](#)



[Last ned podcasten direkte](#)

Klikk på knappen over for å spille av podcasten i nettleseren, eller last ned filen for lokal avspilling.



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.

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:** Fokusperson i midten
- **Statistikk-diagrammer:** Kjønnsfordeling, 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



 [Download podcast directly](#)

Click the button above to play the podcast in your browser, or download the file for local playback.

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                            # Slektstre-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.

Bidrag / Contributing

Bidrag er velkommen! Vennligst opprett en issue eller pull request. / Contributions are welcome! Please create an issue or 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
```

Kontakt / Contact

Arvid Lundervold - [GitHub](#)



Slektstrær og Grafer - En Introduksjon



Family Trees and Graphs - An Introduction

Norsk | English

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.

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 programering
- 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

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
-

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

Slektstrær - En historisk oversikt

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

Hvorfor er slektstrær 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

Type slektstrær

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
- Fokusperson i midten
- Gir komplett oversikt

4. Vifte-diagram (Fan Chart)

- Sirkulær visning av forfedre
- Estetisk tiltalende
- Populær i moderne genealogi-programmer

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

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?
-

🇮🇹 Del 2: Grunnleggende grafteori / Part 2: Basic Graph Theory

🇳🇴 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)
- **Retnet graf:** Kanter har retning ($A \rightarrow B$)
- **Uretnet graf:** Kanter har ingen retning ($A \leftrightarrow B$)
- **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 ($A \leftrightarrow B$)
- **Tree:** A special type of graph without cycles

🇳🇴 Eksempler med NetworkX / Examples with NetworkX

La oss se på noen enkle eksempler:

```
In [1]: # Importer nødvendige biblioteker
import sys
sys.path.append('../src')

import networkx as nx
import matplotlib.pyplot as plt
from models import Person, Gender
from tree import Slektstre
from datetime import date

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

✓ Biblioteker importert!
✓ Libraries imported!

```
In [3]: # 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
G.add_node("A", name="Alice")
G.add_node("B", name="Bob")
G.add_node("C", name="Charlie")

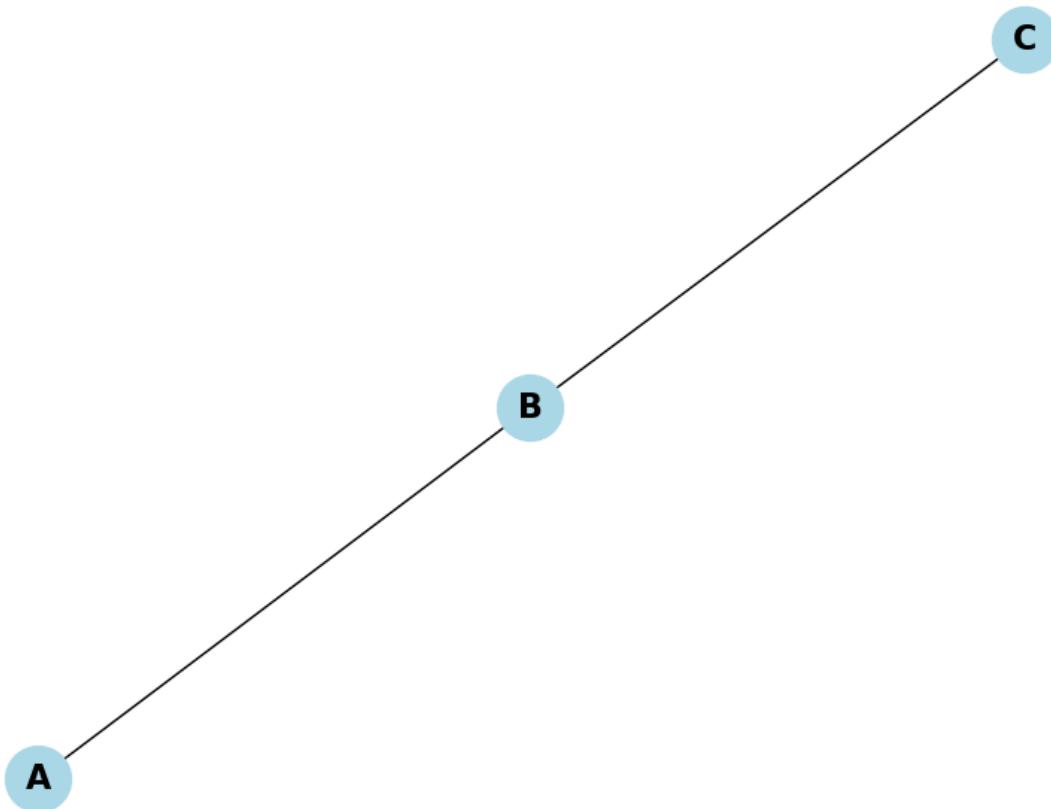
# Legg til kanter
G.add_edge("A", "B")
G.add_edge("B", "C")

# Visualiser
plt.figure(figsize=(8, 6))
pos = nx.spring_layout(G)
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()}")
```

🇳🇴 Eksempel 1: Enkel urettet graf
🇬🇧 Example 1: Simple undirected graph

Enkel urettet graf / Simple undirected graph



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

```
In [6]: # 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
plt.figure(figsize=(8, 6))
pos = nx.spring_layout(D)

# 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')

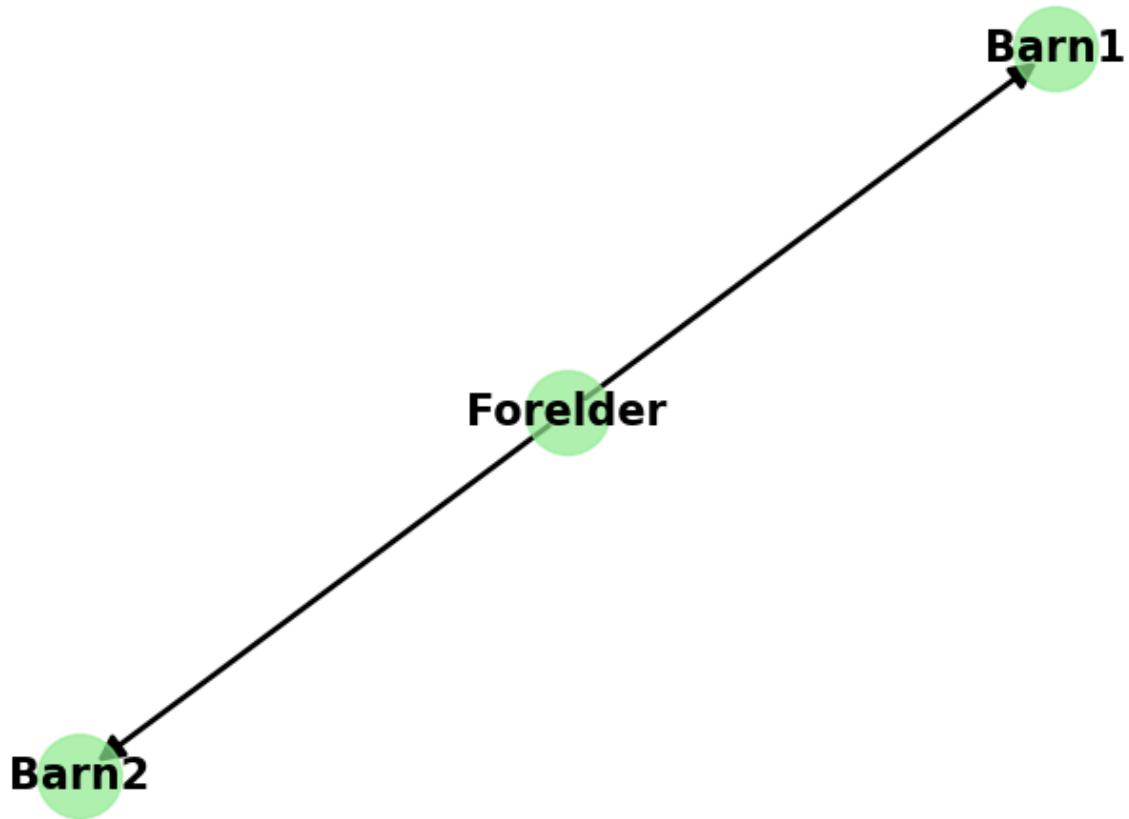
plt.title("Rettet graf (forelder-barn) / Directed graph (parent-child)")
plt.axis('off')
plt.show()

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

- 🇳🇴 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 farfar).

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

```
In [7]: # 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
pos1 = nx.spring_layout(tree)
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)")

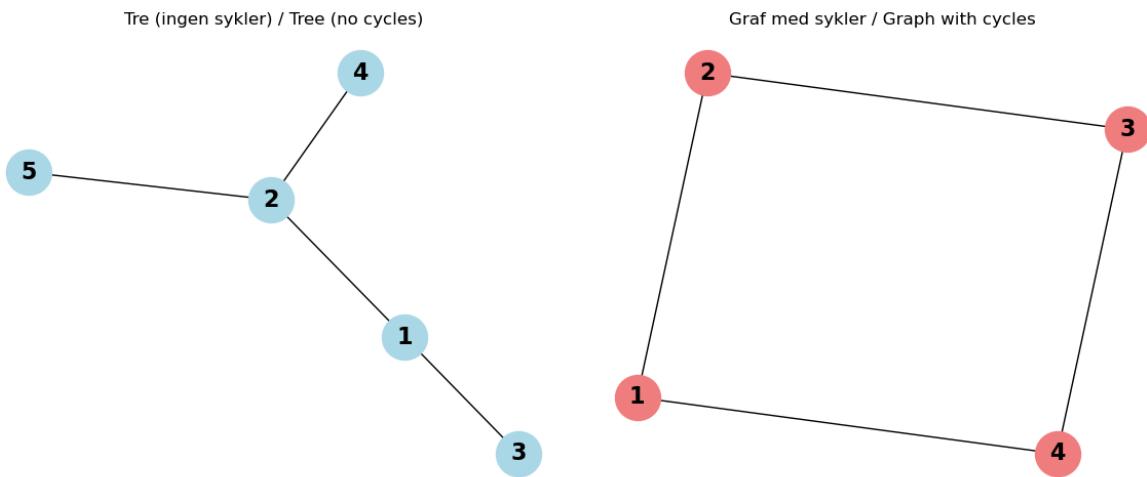
# Tegn grafen med sykler
pos2 = nx.spring_layout(cycle_graph)
nx.draw(cycle_graph, pos2, with_labels=True, node_color='lightcoral',
        node_size=1000, font_size=16, font_weight='bold', ax=ax2)
ax2.set_title("Graf med sykler / Graph with cycles")

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)}")

```

- FLAG: Eksempel 3: Tre vs. Graf med sykler
 FLAG: Example 3: Tree vs. Graph with cycles



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

Del 3: Slektstrær SOM grafer / Part 3: Family Trees AS Graphs

FLAG: 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
Sleksrelasjon / Family relationship	Kant / Edge
Forelder-barn / Parent-child	Retnet kant / Directed edge
Ekteskap / Marriage	Uretnet 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. **Analysse:** 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

```
In [9]: # Eksempel 4: Bygg et enkelt slektstre som graf / Build a simple family tree
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
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(family_graph, k=3, iterations=50)

# 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')

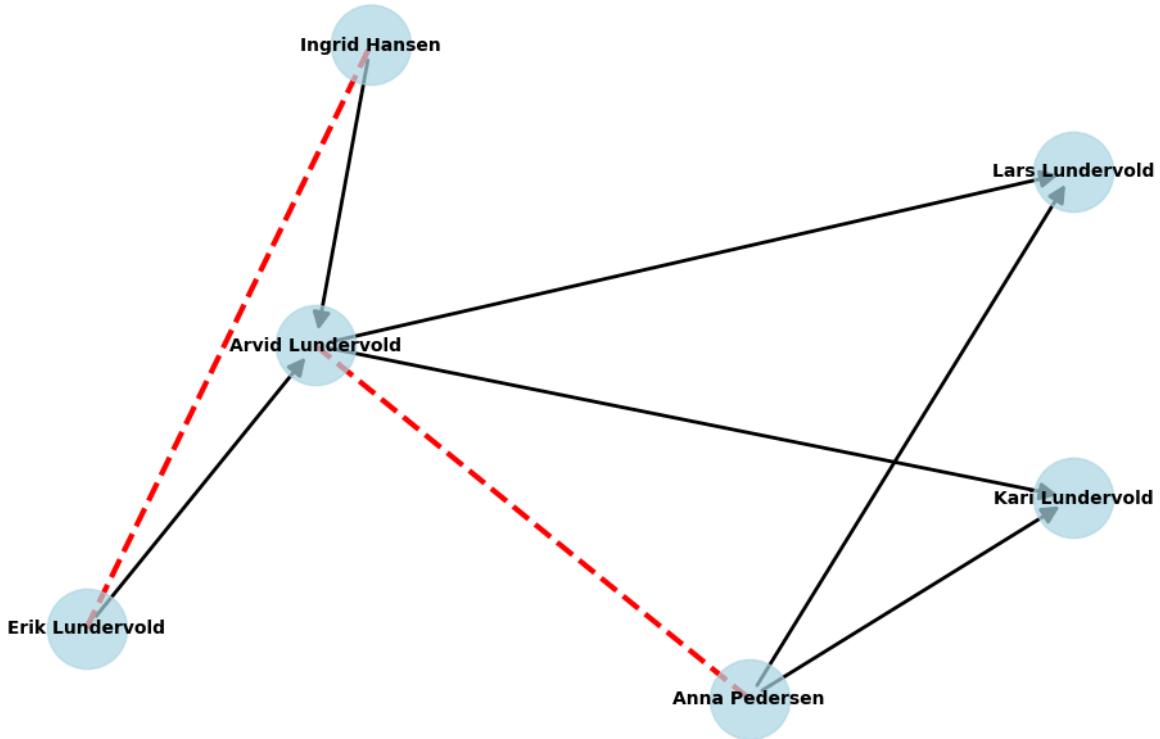
plt.title("Slektstre som graf / Family tree as graph")
plt.axis('off')
plt.show()

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()}")

```

🇳🇴 Eksempel 4: Bygg et enkelt slektstre som graf

🇬🇧 Example 4: Build a simple family tree as graph



Antall personer: 6

Antall relasjoner: 8

Number of people: 6

Number of relationships: 8

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

🇳🇴 Refleksjonsoppgaver / Reflection Questions

1. Slektkap 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 lengre tilbake i tid du går?
- Why do family trees become more complex the further back in time you go?

🇳🇴 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.

In [10]:

```
# 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
exercise_tree.add_node("Parent", name="Parent Person", role="parent")
exercise_tree.add_node("Child1", name="First Child", role="child")
exercise_tree.add_node("Child2", name="Second Child", role="child")

# Legg til relasjoner
exercise_tree.add_edge("Parent", "Child1", relation="parent-child")
exercise_tree.add_edge("Parent", "Child2", relation="parent-child")
exercise_tree.add_edge("Child1", "Child2", relation="siblings")

# Visualiser
plt.figure(figsize=(10, 6))
pos = nx.spring_layout(exercise_tree, k=2)

# 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)

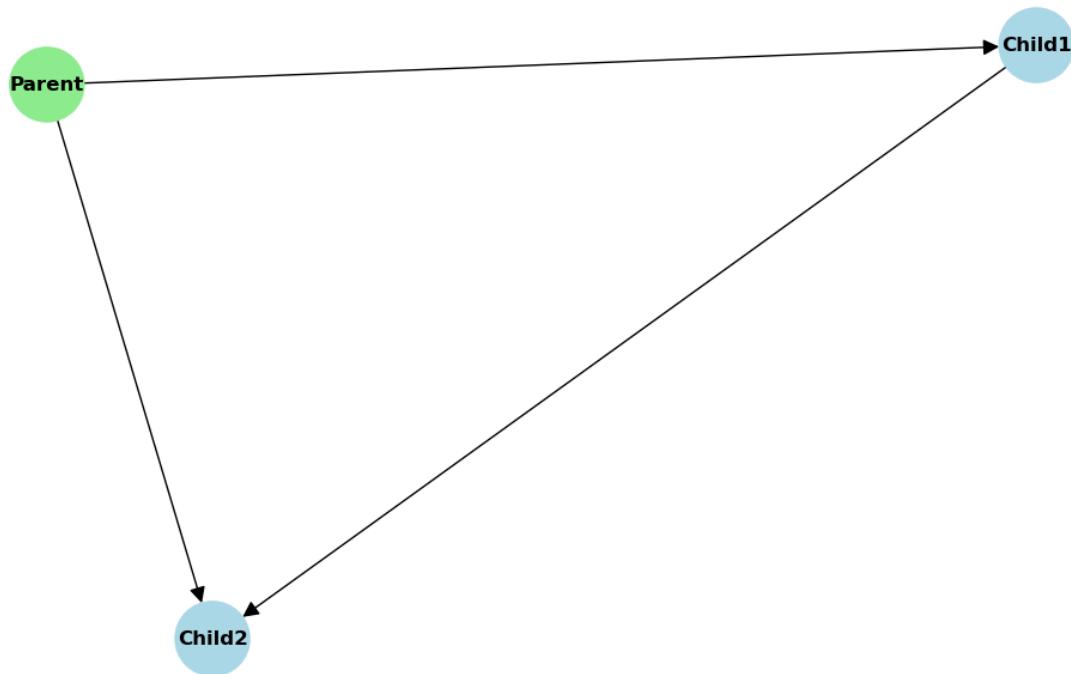
plt.title("3-personers slektstre / 3-person family tree")
plt.show()

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()}")
```

🇳🇴 Løsning til Oppgave 1 / Solution to Exercise 1

🇬🇧 Solution to Exercise 1

3-personers slektstre / 3-person family tree



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

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.

```
In [11]: # LØSNING / SOLUTION
print("FLAG Løsning til Oppgave 2 / Solution to Exercise 2")
print("UK 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_gra
```

```

print(f"Paths from {family_graph.nodes[start_person]['name']} to {family_gra

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_
    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 → La
rs Lundervold

Korteste sti: Erik Lundervold → Arvid Lundervold → Lars Lundervold

Shortest path: Erik Lundervold → Arvid Lundervold → Lars Lundervold

Del 5: Praktiske anvendelser / Part 5: Practical Applications

🇳🇴 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:

```
In [12]: # Last inn eksempel-familien og vis hvordan Slektstre-klassen bruker Network
print("🇳🇴 Hvordan Slektstre-klassen bruker NetworkX")
print("🇬🇧 How the Slektstre class uses NetworkX")

# Last inn eksempel-familien
from family_io import load_from_yaml

familie_data = load_from_yaml('../data/eksempel_familie.yaml')
slektstre = Slektstre(familie_data)

print(f"\nFamilie lastet med {len(familie_data.personer)} personer og {len(f
```

```

print(f"Family loaded with {len(familie_data.personer)} people and {len(famili
# Vis hvordan grafen brukes internt
print(f"\nInternt graf-objekt / Internal graph object:")
print(f" Type: {type(slektstre.graph)}")
print(f" Antall noder: {slektstre.graph.number_of_nodes()}")
print(f" Number of nodes: {slektstre.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
```

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

🇳🇴 Hva har du lært? / What have you learned?

I denne notebooken har du lært:

In this notebook you have learned:

- 🌳 **Slektstrær:** Hva de er, hvorfor de er viktige, og forskjellige typer
- 📊 **Grafteori:** Grunnleggende konsepter som noder, kanter, og treer
- 🔗 **Kobling:** Hvordan slektstrær kan representeres som grafer
- 💻 **Praksis:** Hvordan bruke NetworkX til å bygge og analysere slektstrær
- ⌚ **Øvelser:** Både refleksjon og programmering for å forstå begrepene

English:

- 🌳 **Family trees:** What they are, why they're important, and different types
- 📊 **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

Ordliste / Vocabulary

Norsk	English	Grafteori	Forklaring
Slektstre	Family tree	Tree	Spesiell type graf uten sykler
Person	Person	Node/Vertex	Et punkt i grafen
Sleksrelasjon	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
Sammenheng	Connectivity	Connected	Alle noder kan nås
Komponent	Component	Component	Sammenhengende del av graf

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

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

🇳🇴 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.

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.

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

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

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
```

```
In [1]: # Importer nødvendige biblioteker  
import sys  
import os  
sys.path.append('../src')  
  
# Importer modulene direkte fra src-mappen  
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, plt  
from localization import t, get_available_languages
```

```
# Importer også standardbiblioteker
import matplotlib.pyplot as plt
import plotly.express as px
from datetime import date
import pandas as pd

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

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

Grunnleggende konsepter

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

Ekteskap-modellen

Et `Ekteskap` kobler to personer sammen:

- **Partnere:** referanser til to person-IDer
- **Datoer:** ekteskapsdato, skilsmisse_dato
- **Metadata:** ekteskapssted, type, notater

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

```
In [2]: # 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

Last eksempeldata

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

```

In [3]: # Last eksempel-familie
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.fødsel})

```

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)

Test visualisering

La oss teste en enkel visualisering:

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. søsknen)

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

```
In [4]: # 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
    print(f"\n📊 Totalt: {len(slektstre.get_all_persons())} personer")
```

👤 Alle personer i slektstreet:

ID:				
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

```
In [5]: # 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
        status = "Aktivt" if ekteskap.er_aktivt else "Skilt"
```

```

print(f"ID: {ekteskap.id:3} | {partner1.fullt_navn:20} ↔ {partner2.f
print(f"\n===== Totalt: {len(slektstre.familie_data.ekteskap)} ekteskap")

```

===== Alle ekteskap i slektstreet:

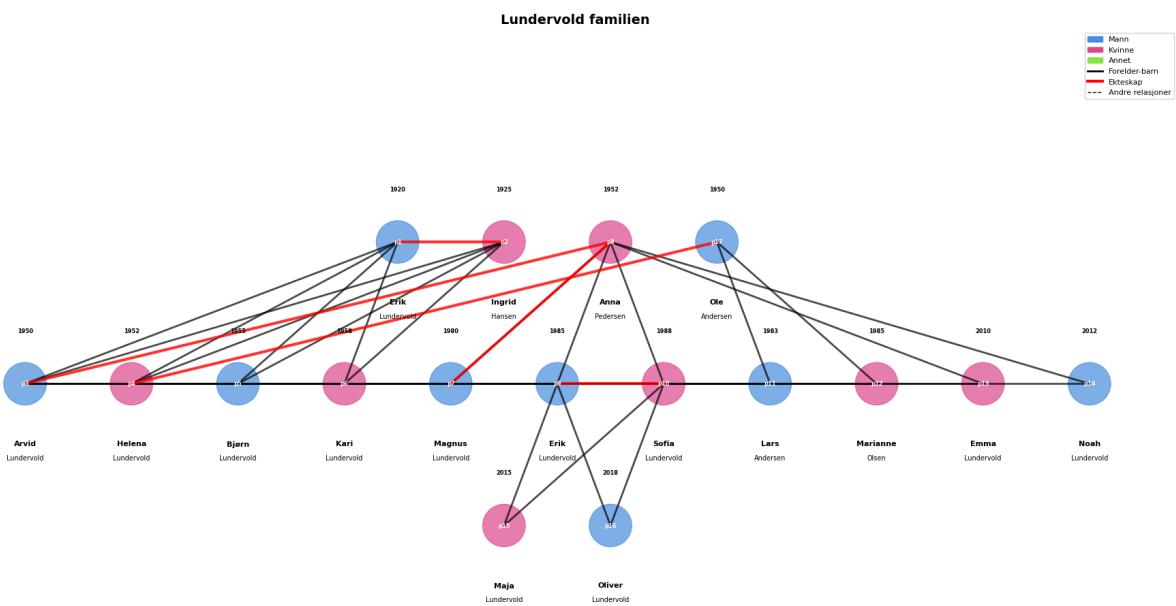
```

=====
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

In [6]: # Test hierarkisk slektstre
`fig = plot_hierarchical_tree(slektstre, title="Lundervold familien")
plt.show()`



In []:

In []:

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.

Importer biblioteker

```
In [11]: # Importer biblioteker
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!")
```

✅ Biblioteker importert!

Opprett et tomt slektstre

```
In [12]: # Opprett et tomt slektstre
slektstre = Slektstre()

print(f"Tomt slektstre opprettet med {len(slektstre.get_all_persons())} pers")
```

Tomt slektstre opprettet med 0 personer

Legg til personer

La oss legge til personer fra tre generasjoner:

```
In [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"Legt til: {bestefar.fullt_navn}")
```

Lagt til: Erik Lundervold

```
In [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
```

```
In [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

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

```
In [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}")

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!
```

Opprett relasjoner

Nå må vi koble personene sammen med relasjoner:

```
In [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

```
In [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

```
In [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}")
```

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

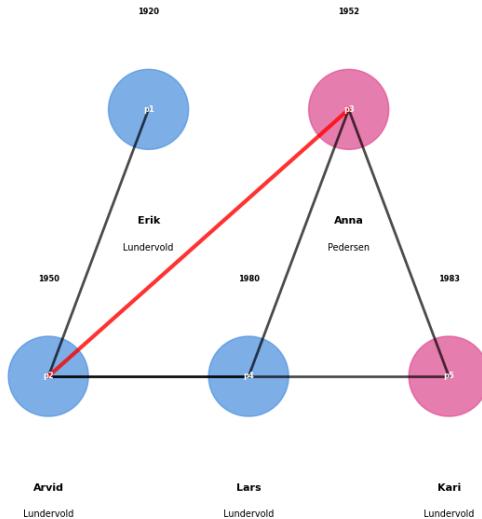
Visualiser slektstreet

```
In [20]: from visualization import plot_hierarchical_tree

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

Manuelt bygget slektstre

■	Mann
■	Kvinne
■	Annet
—	Forelder-barn
—	Ekteskap
- - -	Andre relasjoner



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.

Bygge slektstre manuelt

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

Importer biblioteker

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

# Importer biblioteker
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!")

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

# SJEKK AT SLEKTSTREET ER HELT TOMT
alle_personer = manuelt_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}")
else:
    print("✓ Slektstreet er tomt – vi kan begynne!")
```

```
In [ ]: # 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
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!")
```

```
In [ ]: # 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!")
```

```
In [ ]: # 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!")
```

Opprett et tomt slektstre

```
In [ ]: # 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")

```

Rydd opp i eksisterende data

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

```

In [ ]: # 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 = Slektstre()
    print("✅ Nytt tomt slektstre opprettet")
else:
    print("✅ Ingen eksisterende data å rydde opp i")

```

```

In [ ]: # 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-cellene på nytt.")
    raise Exception("Slektstreet er ikke tomt! Kjør opprydding-cellene på nytt")
else:
    print("✅ Slektstreet er tomt – vi kan begynne!")

```

Legg til personer

La oss legge til personer fra tre generasjoner:

```

In [ ]: # 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}")
```

```
In [ ]: # 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}")
```

```
In [ ]: # 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"Legt til: {barn1.fullt_navn}")
print(f"Legt til: {barn2.fullt_navn}")
```

Legg til personer

La oss bygge en enkel familie med 3 generasjoner:

```
In [ ]: # 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"Legt til: {bestefar.fullt_navn} og {bestemor.fullt_navn}")
```

```
In [ ]: # 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.strftime('%Y-%m-%d')}")

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

```
In [ ]: # 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.strftime('%Y-%m-%d')}")

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

```
In [ ]: # 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}")
```

```
In [ ]: # 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!")
```

```
In [ ]: # 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!")

```

```

In [ ]: # FINAL BEKREFTELSE – Vis alle personer med deres ID-er
print("FINAL BEKREFTELSE: Alle personer i slektstre:")
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!")

```

Opprett relasjoner

Nå må vi koble personene sammen med relasjoner:

```

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

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

```

```

In [ ]: # 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}")

```

```
In [ ]: # 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}")

```

Analyser slektstreet

La oss se på slektskap og relasjoner:

```
In [ ]: # Hent søsknen
søsknen = slektstre.get_siblings(barn1.id)
print(f"Søsknen til {barn1.fullt_navn}:")
for søsknen_person in søsknen:
    print(f"- {søsknen_person.fullt_navn}")
```

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

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

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

Visualiser slektstreet

```
In [ ]: from visualization import plot_hierarchical_tree

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

Statistikk

```
In [ ]: # 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']}")
```

Validere slektstreet

La oss sjekke om det er noen problemer med slektstreet:

```
In [ ]: # 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!")
```

Lagre slektstreet

Du kan lagre slektstreet til fil:

```
In [ ]: from family_io import save_to_yaml

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

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.

Import/eksport av data

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

Støttede formater

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

```
In [1]: # Importer nødvendige biblioteker
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
)
from datetime import date
import pandas as pd

print("✅ Alle biblioteker importert!")
```

✅ Alle biblioteker importert!

1. YAML Format (Anbefalt)

YAML er det mest lesbare formatet for familie-data:

```
In [2]: # Last eksempel-familie fra YAML
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(fam
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_perso
```

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

Eksempel person: Erik Lundervold (ID: p1)

```
In [3]: # 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

2. JSON Format

JSON er godt for programmatisk bruk og kompatibilitet:

```
In [4]: # 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

3. CSV Format

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

```
In [5]: # 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 partner
e
0 Flyktet fra Norge under krigen|Bygde sitt eget...      NaN  NaN  NaN
N
1 Møtte Erik på dans i 1947|Spilte piano og sang...      NaN  NaN  NaN
N
2 Doktorgrad fra MIT|Grunnla flere teknologisels...  p1|p2  NaN  NaN
N
3 Designet flere kjente bygninger i Bergen|Malte...  p1|p2  NaN  NaN
N
4 Spilte fotball på høyt nivå i ungdommen|Trente...  p1|p2  NaN  NaN
N
```

```
In [6]: # 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

4. GEDCOM Format

GEDCOM er standarden for genealogi-programmer:

```
In [7]: # 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 10 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/
```

5. Sammenligning av formater

La oss sammenligne størrelsen og kompleksiteten:

```
In [8]: import os

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

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

6. Validering av importerte data

La oss sjekke at alle formater gir samme resultat:

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.

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
```

GEDCOM-tagger

Person-tagger:

- **INDI** - Individ (person)
- **NAME** - Navn
- **GIVN** - Fornavn
- **SURN** - Etternavn
- **SEX** - Kjønn (M/F)

- **BIRT** - Fødsel
- **DEAT** - Død
- **MARR** - Ekteskap
- **DIV** - Skilsmisses

Familie-tagger:

- **FAM** - Familie
- **HUSB** - Ektefelle
- **WIFE** - Ektefelle
- **CHIL** - Barn

Metadata-tagger:

- **DATE** - Dato
- **PLAC** - Sted
- **NOTE** - Notater
- **SOUR** - Kilde

Fordeler med GEDCOM

- Standardisert** - Fungerer med alle genealogi-programmer
- Portabel** - Enkelt å dele mellom systemer
- Komplett** - Støtter alle typer genealogiske data
- Lesbar** - Menneske-lesbart tekstformat

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

```
In [9]: # 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 10 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

```

In [10]:

```

# Sammenligne 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}")

# JSON
json_size = os.path.getsize("eksport_familie.json")
print(f"JSON        {json_size:8d} bytes    {len(familie_data.personer):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}")

# GEDCOM

```

```

gedcom_size = os.path.getsize("eksport_familie.ged")
print(f"GEDCOM      {gedcom_size:8d} bytes      {len(familie_data.personer):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
<hr/>			
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

```
In [11]: # 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
<hr/>		
YAML	17	5
JSON	17	5
CSV	17	5

✅ Alle formater har samme antall personer!

7. Rydde opp

La oss slette de midlertidige filene:

```
In [12]: # 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"\n✅ Slettet {fil}")

print("\n\n✅ Opprydding fullført!")
```

```
✅ Slettet eksport_familie.yaml
✅ Slettet eksport_familie.json
✅ Slettet eksport_familie.csv
✅ Slettet eksport_familie.ged
```

```
✅ Opprydding fullført!
```

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.

Visualisering av slektstre

I denne notebooken utforsker vi alle tilgjengelige visualiseringsalternativer for slektstre.

Tilgjengelige visualiseringer

1. **Hierarkisk tre** - Tradisjonell struktur
2. **Fan chart** - Sirkulær visning
3. **Interaktiv tre** - Plotly-basert
4. **Hourglass view** - Fokusperson i midten
5. **Statistikk** - Grafer og diagrammer

```
In [1]: # Importer nødvendige biblioteker
import sys
sys.path.append('../src')

from models import Person, Ekteskap, FamilieData, Gender
from tree import Slektstre
from family_io import load_from_yaml
from visualization import (
    plot_hierarchical_tree,
    plot_fan_chart,
    plot_interactive_tree,
    plot_statistics,
    plot_hourglass_view
)
import matplotlib.pyplot as plt
import plotly.express as px
from datetime import date

print("✅ Alle biblioteker importert!")
```

✅ Alle biblioteker importert!

```
In [2]: # Last eksempel-familie
familie_data = load_from_yaml('../data/eksempel_familie.yaml')
slektstre = Slektstre(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("\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

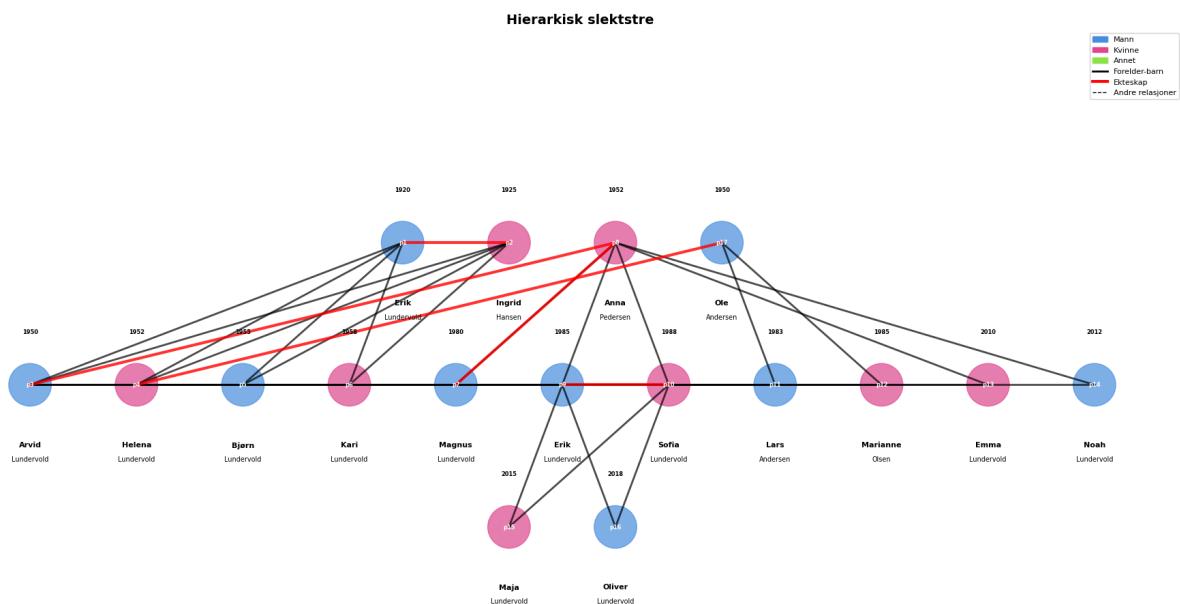
Gjennomsnittsalder: 49.2 år

Antall ekteskap: 5

1. Hierarkisk tre

Tradisjonell struktur med generasjoner fra topp til bunn:

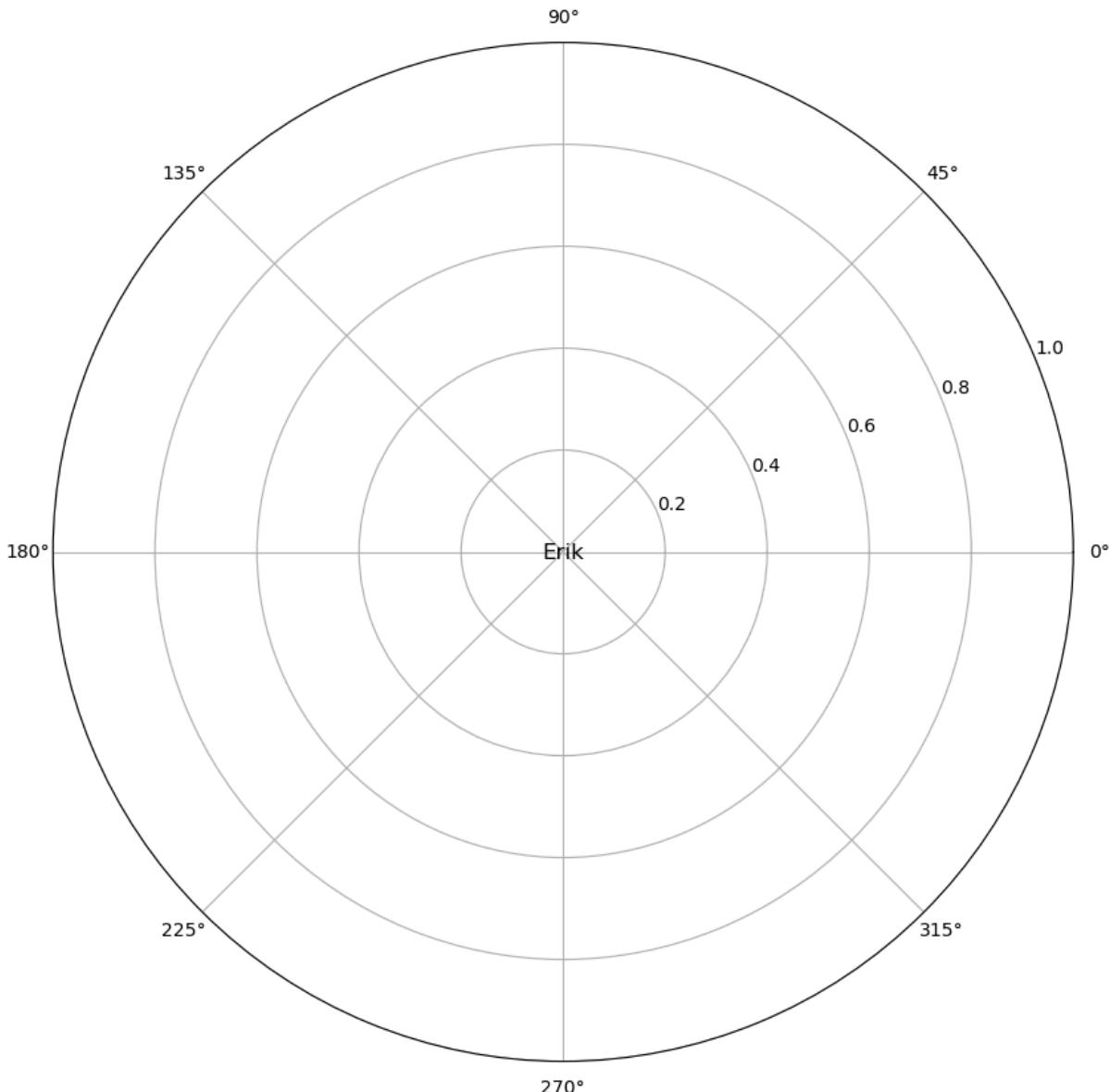
```
In [3]: # Plott hierarkisk tre
fig = plot_hierarchical_tree(slektstre, title="Hierarkisk slektstre")
plt.show()
```



2. Fan Chart

Sirkulær visning med eldste generasjon i midten:

```
In [4]: # 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!")
```



3. Interaktiv tre

Plotly-basert interaktiv visualisering med hover-info:

```
In [5]: # Plott interaktiv tre
fig = plot_interactive_tree(slektstre, title="Interaktiv slektstre")
fig.show()
```

4. Hourglass View

Fokusperson i midten med forfedre over og etterkommere under:

```
In [6]: # Velg en fokusperson (Arvid Lundervold)
fokusperson_id = "p3" # Arvid Lundervold
fokusperson = slektstre.get_person(fokusperson_id)
```

```

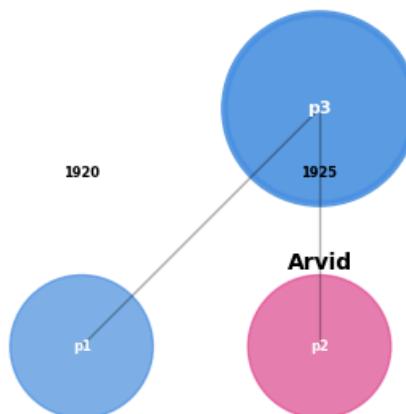
if fokusperson:
    print(f"Fokusperson: {fokusperson.fullt_navn} (ID: {fokusperson.id})")

    # Plott hourglass view
    fig = plot_hourglass_view(slektstre, fokusperson_id, title=f"Hourglass \n"
                               plt.show()
else:
    print("Fokusperson ikke funnet!")

```

Fokusperson: Arvid Lundervold (ID: p3)

Hourglass View - Arvid Lundervold



Erik Lundervold	Ingrid Hansen
---------------------------	-------------------------

5. Statistikk og diagrammer

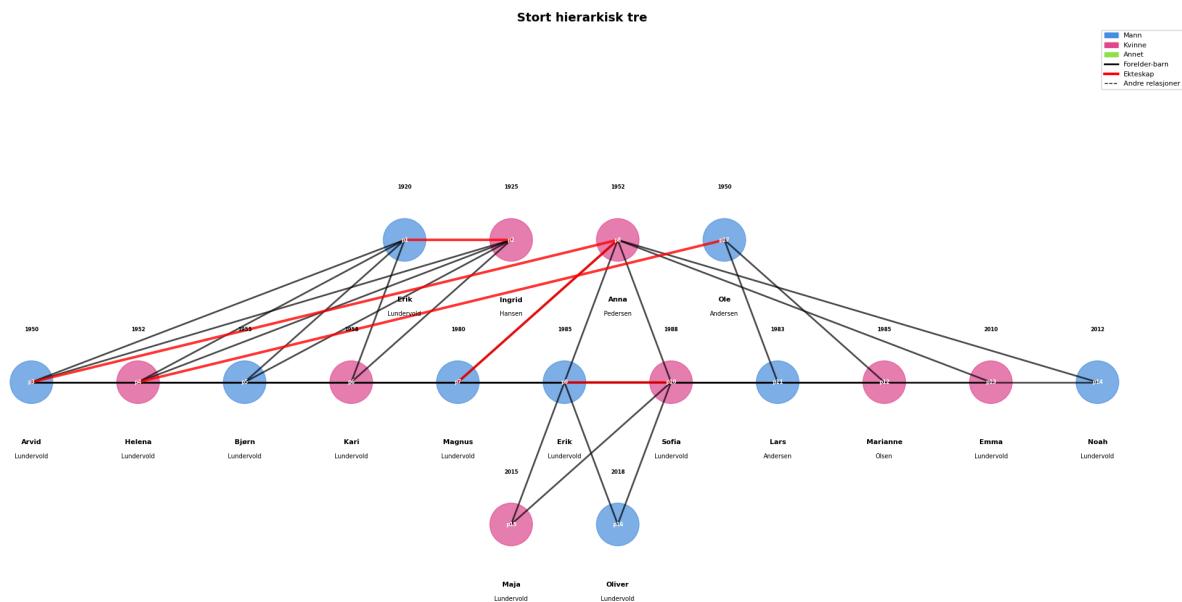
Vis ulike statistikk-diagrammer:

```
In [7]: # Plott statistikk
fig = plot_statistics(slektstre)
fig.show()
```

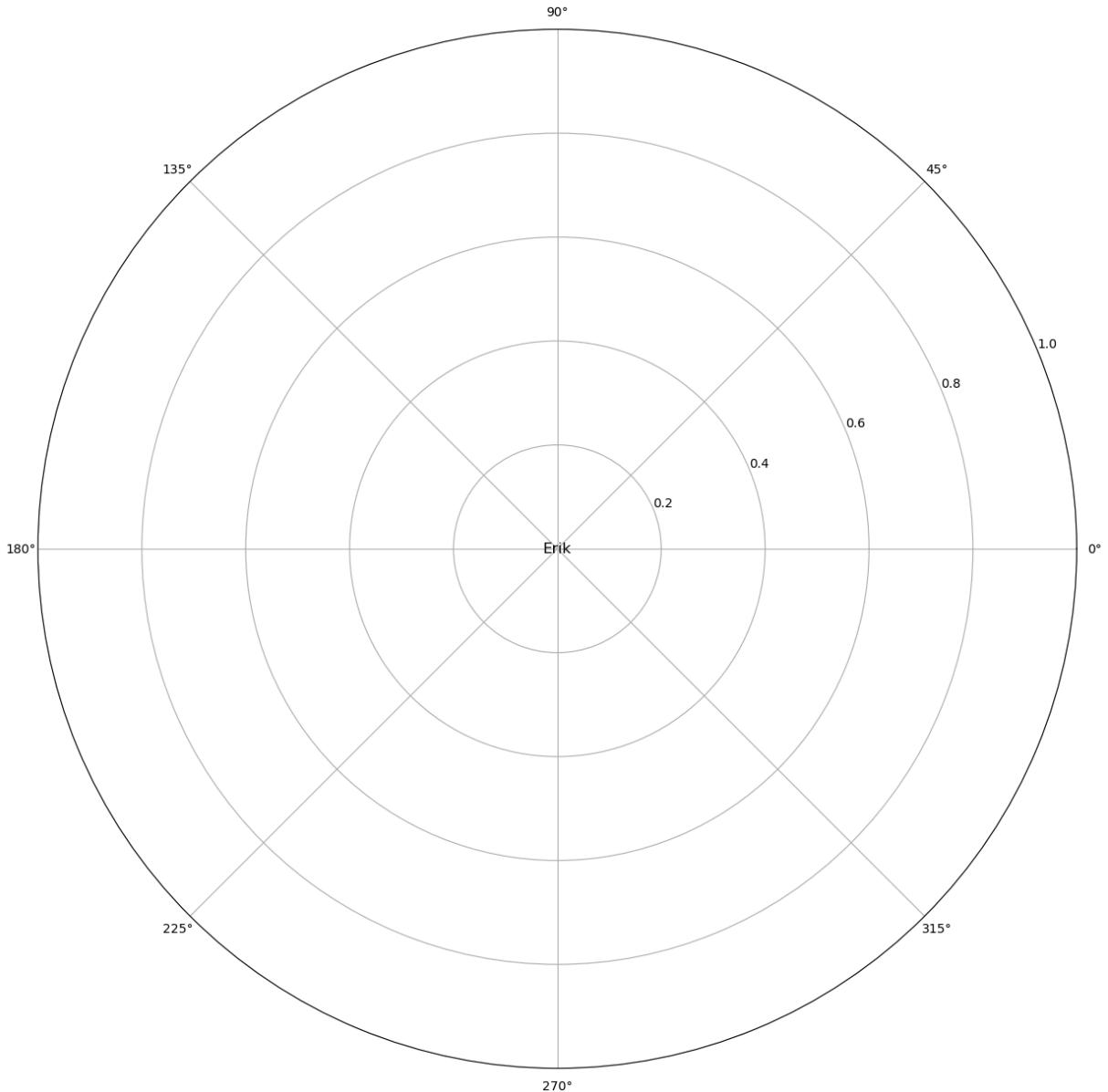
6. Tilpassede visualiseringer

La oss eksperimentere med forskjellige parametere:

```
In [8]: # Hierarkisk tre med større figurstørrelse
fig = plot_hierarchical_tree(
    slektstre,
    title="Stort hierarkisk tre",
    figsize=(20, 12)
)
plt.show()
```



```
In [9]: # 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!")
```



7. Sammenligning av visualiseringer

La oss sammenligne forskjellige visualiseringer side ved side:

```
In [10]: # 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()

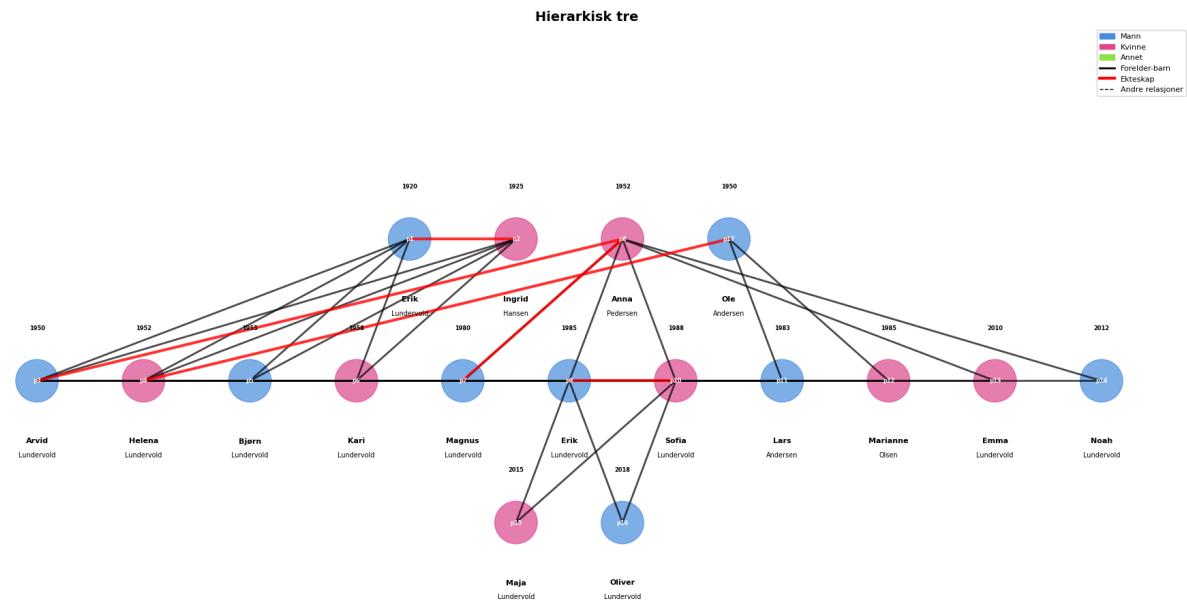
# Fan chart
alle_personer = slektstre.get_all_persons()
if alle_personer:
    root_person_id = alle_personer[0].id
    fig2 = plot_fan_chart(slektstre, root_person_id, title="Fan Chart")
    plt.show()
```

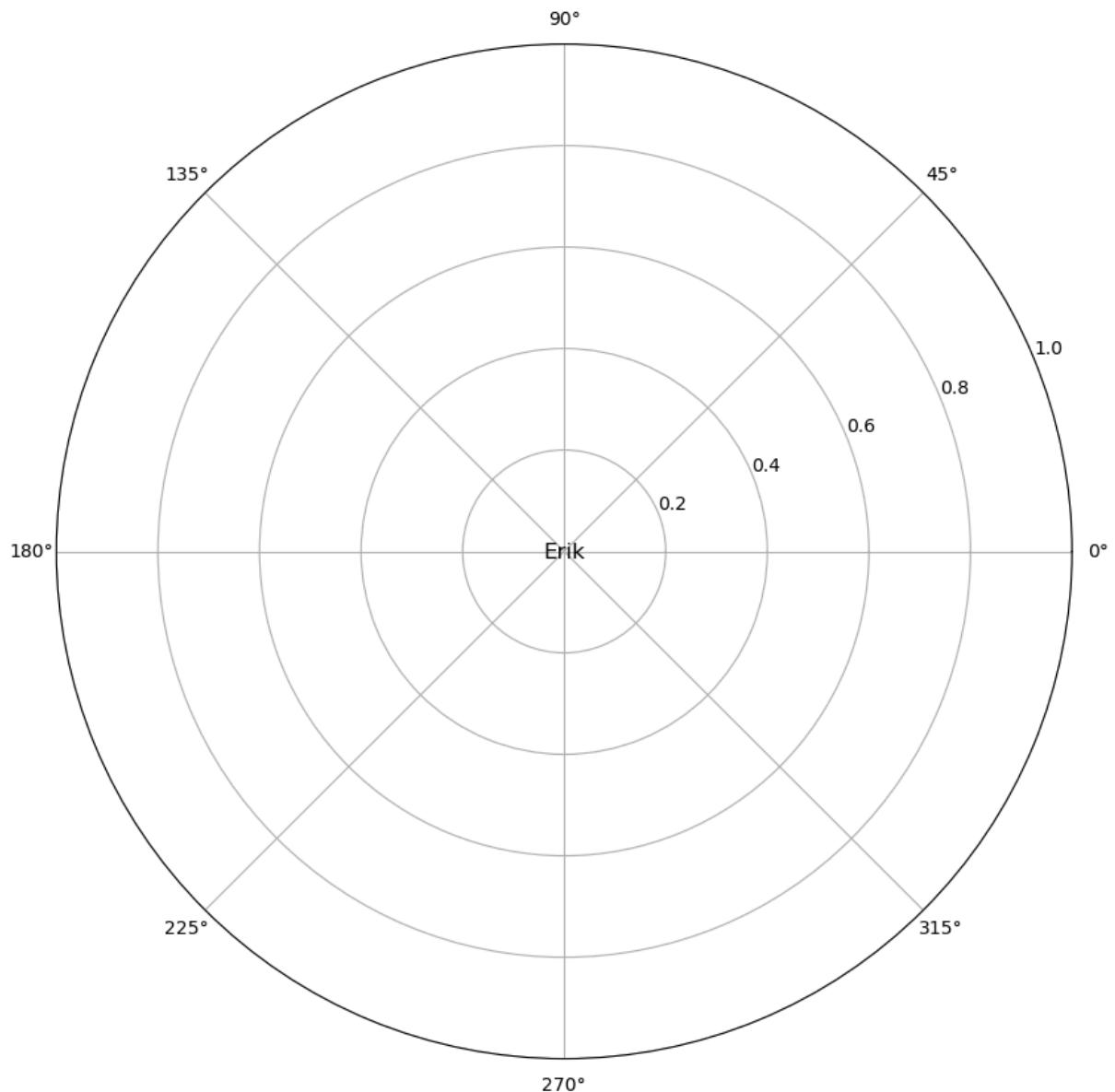
```

# Hourglass view
fig3 = plot_hourglass_view(slektstre, "p3", title="Hourglass View")
plt.show()

# Statistikk
fig4 = plot_statistics(slektstre)
fig4.show()

```





Hourglass View



8. Eksport av visualiseringer

La oss lagre visualiseringer til filer:

```
In [11]: # Lagre visualiseringer til filer
import os

# Opprett mappe for eksporterte bilder
os.makedirs("eksporterte_bilder", exist_ok=True)

# Lagre hierarkisk tre
fig = plot_hierarchical_tree(slektstre, title="Hierarkisk slektstre")
fig.savefig("eksporterte_bilder/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("eksporterte_bilder/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("eksporterte_bilder/hourglass_view.png", dpi=300, bbox_inches='tight')
plt.close(fig)

# Lagre statistikk (Plotly-figur)
fig = plot_statistics(slektstre)
fig.write_image("eksporterte_bilder/statistikk.png", width=800, height=600,
# Plotly-figurer trenger ikke plt.close()

print("✅ Alle visualiseringer lagret til 'eksporterte_bilder/' mappen")
print("📁 Filene:")
for fil in os.listdir("eksporterte_bilder"):
    print(f" - {fil}")

✓ Alle visualiseringer lagret til 'eksporterte_bilder/' mappen
📁 Filene:
- fan_chart.png
- hourglass_view.png
- hierarkisk_tre.png
- statistikk.png

```

9. Rydde opp

La oss slette de eksporterte bildene:

```

In [12]: # Slett eksporterte bilder
import shutil

if os.path.exists("eksporterte_bilder"):
    shutil.rmtree("eksporterte_bilder")
    print("📁 Slettet 'eksporterte_bilder/' mappen")

print("✅ Opprydding fullført!")

📁 Slettet 'eksporterte_bilder/' mappen
✓ Opprydding fullført!

```

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** - Fokusperson 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 fullført alle notebookene! Du kan begynne å bygge ditt eget slektstre.

Eksterne genealogi-databaser og API-er

I denne notebooken lærer du hvordan du kan hente slektsinformasjon fra eksterne databaser og integrere dem med ditt slektstre-program.

Tilgjengelige databaser

1. FamilySearch API (Gratis)

- Verdens største genealogi-database
- Over 1 milliard personer
- Gratis API med registrering
- Støtter GEDCOM-import/eksport

2. MyHeritage API (Betalt)

- Kommersiell genealogi-tjeneste
- DNA-analyse og slektsforskning
- API tilgjengelig for utviklere

3. Ancestry.com API (Betalt)

- Største kommersielle genealogi-tjeneste
- Begrenset API-tilgang
- Hovedsakelig for partnere

4. Nasjonale arkiver

- **Digitalarkivet** (Norge) - Gratis
- **Riksarkivet** (Norge) - Gratis
- **Arkivverket** (Norge) - Gratis

5. Wikipedia/Wikidata

- Biografisk informasjon
- Gratis og åpen tilgang
- Begrenset genealogisk data

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

```
In [1]: # Importer nødvendige biblioteker
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 datetime import date
import requests
import json
import time
import os

print("✅ Alle biblioteker importert!")
print("📖 Klar for å utforske eksterne databaser!")
```

✅ Alle biblioteker importert!
📖 Klar for å utforske eksterne databaser!

1. FamilySearch API

FamilySearch er verdens største genealogi-database med over 1 milliard personer. De tilbyr et gratis API for utviklere.

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

API-endepunkter

- **Personer:** /platform/tree/persons
- **Familier:** /platform/tree/families
- **Kilder:** /platform/tree/sources
- **Søk:** /platform/tree/search

Eksempel: Søke etter personer

```
In [2]: # FamilySearch API eksempel (simulert)
# MERK: Dette er et eksempel – du trenger ekte API-nøkler for å bruke Family
```

```

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

2. Digitalarkivet (Norge)

Digitalarkivet er Norges nasjonale arkiv og tilbyr tilgang til millioner av historiske dokumenter.

Tilgjengelige kilder

- **Folketellinger** (1801-1910)
- **Kirkebøker** (døpte, konfirmerte, gift, døde)
- **Skattelister** og matrikkler
- **Emigrasjonslister**
- **Militære arkiver**

API-tilgang

Digitalarkivet har ikke et offisielt API, men tilbyr:

- **REST API** for søk
- **CSV-eksport** av søkeresultater
- **GEDCOM-eksport** for slektsforskning

Eksempel: Søke i kirkebøker

```
In [3]: # Digitalarkivet søkeresultat (simulert)
def digitalarkivet_search_example():
    """
    Eksempel på hvordan man bruker Digitalarkivet API.
    Dette er simulert data basert på ekte arkivstruktur.
    """
    # Søk etter person med ID FS123456789
    result = digitalarkivet_search("FS123456789")
    print(result)
```

```

# 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

3. Wikipedia API

Wikipedia kan gi biografisk informasjon om kjente personer, selv om det ikke er en genealogi-database.

Wikipedia API

- **Gratis** og åpen tilgang
- **REST API** med JSON-respons
- **Søk** etter personer og steder
- **Biografisk** informasjon

Eksempel: Søke etter norske personer

```
In [4]: # 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 artikkkel in wikipedia_resultat['resultater']:
    print(f"\n■ {artikkkel['tittel']}")  

    print(f"  Beskrivelse: {artikkkel['beskrivelse']}")  

    print(f"  Født: {artikkkel['fødselsår']} i {artikkkel['fødested']}")  

    print(f"  Død: {artikkkel['dødsår']}")  

    print(f"  Kjent for: {artikkkel['kjent_for']}")  

    print(f"  URL: {artikkkel['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

4. Konvertere eksterne data til slektstre

Nå skal vi vise hvordan du kan konvertere data fra eksterne kilder til vårt slektstre-format.

In [5]:

```
# 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'])

            personer.append({
                'navn': f'{fornavn} {etternavn}',
                'fødselsdato': fødselsdato,
                'dødsdato': dødsdato
            })
        except:
            pass

    # Konverter ekteskap
    for fs_ekteskap in familysearch_data['relationships']:
        ekteskap.append({
            'person1': personer[fs_ekteskap['person1Index']],
            'person2': personer[fs_ekteskap['person2Index']],
            'type': fs_ekteskap['type']
        })

    return {
        'personer': personer,
        'ekteskap': ekteskap
    }
```

```

except:
    pass

# Bestem kjønn
kjønn = Gender.MALE if fs_person['gender'] == 'Male' else Gender.FEM

# 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 (foreklet)
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 == s
                 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)

```
In [6]: # Opprett slektstre fra konverterte data
slektstre_ekstern = Slektstre(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

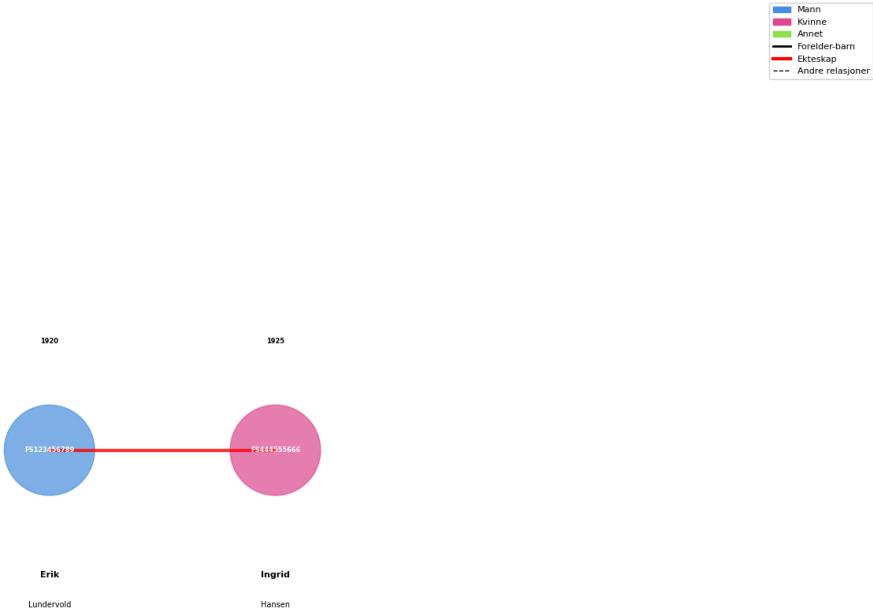
fig = plot_hierarchical_tree(slektstre_ekstern, title="Slektstre fra ekstern")
plt.show()
```

🌳 Opprettet slektstre fra eksterne data:

Totalt antall personer: 2

Totalt antall ekteskap: 1

Slektstre fra eksterne databaser



5. Praktiske tips for slektsforskning

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

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**

6. Lagre og dele slektstreet

Eksportere til forskjellige formater

Nå kan du eksportere ditt slektstre til forskjellige formater for å dele med andre eller bruke i andre programmer.

```
In [7]: # Eksporter slektstreet til forskjellige formater
from family_io import save_to_yaml, save_to_json, save_to_csv, export_to_ged

# 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("\nFilstørrelser:")
import os
filer = ["ekstern_slektstre.yaml", "ekstern_slektstre.json", "ekstern_slektstre.csv"]
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:

ekstern_slektstre.yaml	:	1078 bytes
ekstern_slektstre.json	:	1452 bytes
ekstern_slektstre.csv	:	412 bytes
ekstern_slektstre.ged	:	565 bytes

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

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

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**

Slektstre, 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**

Grafeori, 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

P**Path****25, 35**

Sti, sekvens av noder koblet med kanter

Pedigree**15**

Stamtabl, visuell representasjon av forfedre

S**Slektstre****15**

Family tree, visuell representasjon av slektskap og familierelasjoner

Søsken**15, 45**

Personer som deler samme foreldre

T**Tree****25, 35**

Tre, spesiell type graf uten sykler og sammenkoblet

V**Vertex****25**

Hjørne, alternativt navn for node i en graf

Y**YAML****85**

YAML Ain't Markup Language, menneskelesbart dataformat