Intro to Mathematical Modelling and ABM in python for your standard social scientist

2026-01-22

Peter Steiglechner

All code on: https://github.com/PeterSteiglechner/workshop_mathmodels.git

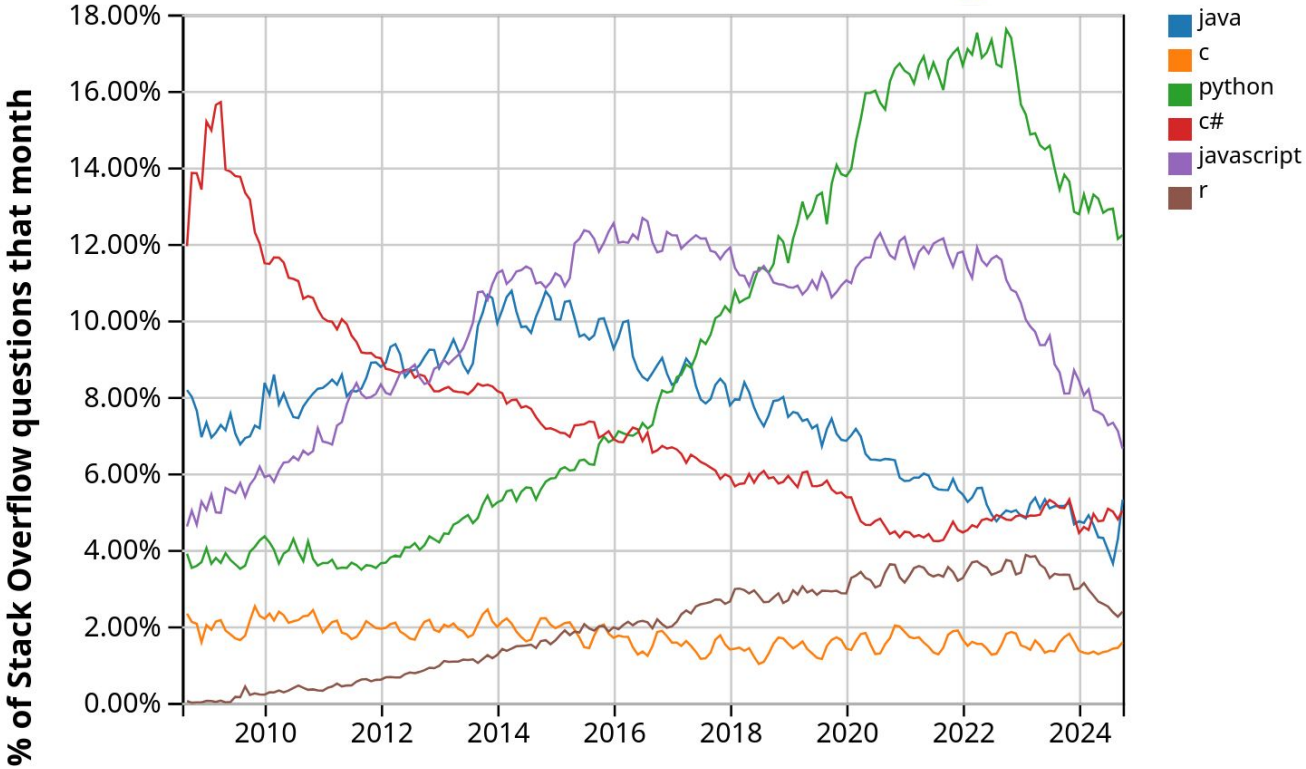
Overview

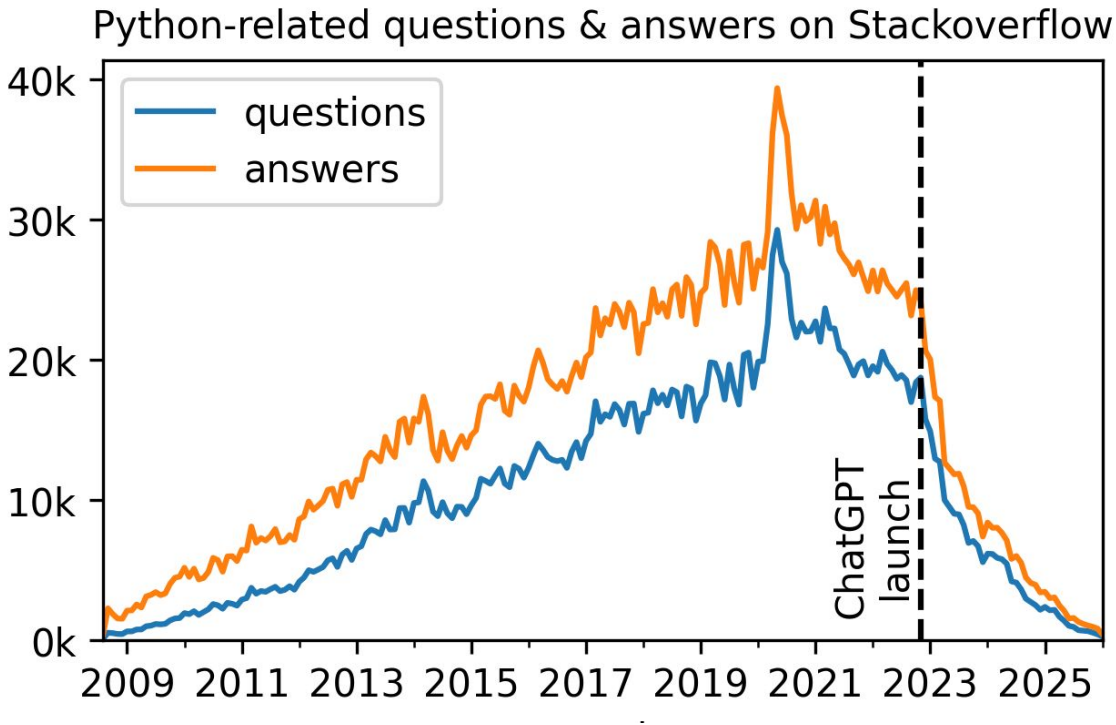
1. Python [1min]
2. Mathematical Modelling [4min]
3. System Dynamics Models with Python [30-45min]
    a. Structure
    b. SIR model (with playtime)
4. ABM with Python [60++min]
    a. Structure
    b. SIR model (with playtime)
    c. Opinion Dynamics Model (with playtime)
    d. Sensitivity Analysis
    e. (Schelling Model)
5. (optional) Visualisation – core parts of python figures [5min]
6. Au revoir and Summary

What is python? Why python?

+ GENERAL-PURPOSE PROGRAMMING LANGUAGE
+ OPEN-SOURCE
+ EASY: high readability and interpretability
+ COMMUNITY: vast support
+ STEEP LEARNING CURVE
+ FAST (C++ backend)
+ INDEPENDENT OF MACHINE/OS/…
+ …

The rise of python





I queried data on 2026-01-20 from:

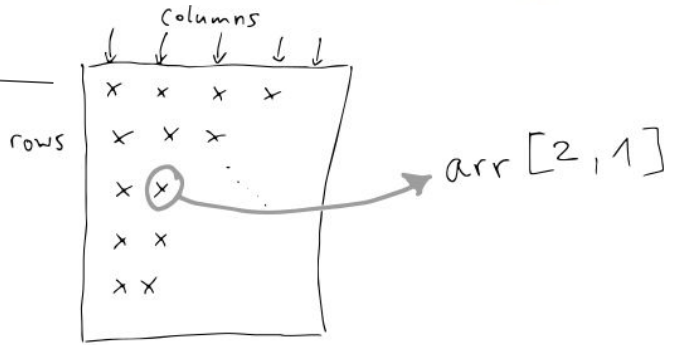
https://data.stackexchange.com/stackoverflow/query/1933718/nr-python-questionsanswers-over-time

No (labelled) training data = No LLM training???

Python Syntax Basics

- basic syntax (operators)
- dictionaries, lists, arrays
- If-else condition
- for-loops, while-loops
- list comprehension
- defining functions
- class and objects (today)

NumPy



- Library for fast **numeric** calculations.

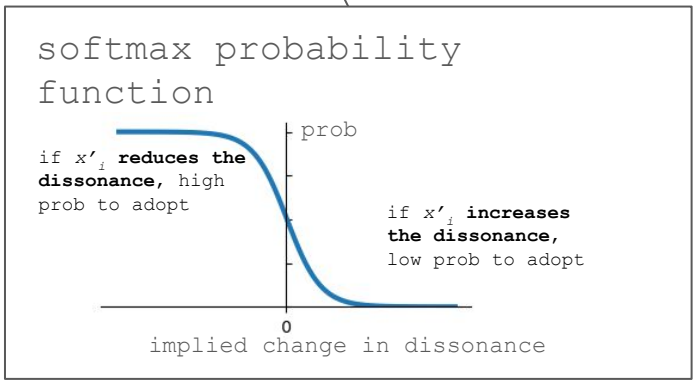
Examples:
- remove red colour from pixel images;
- linear algebra of adjacency matrix

Modelling paradigms

KISS — *Keep it simple, stupid!*

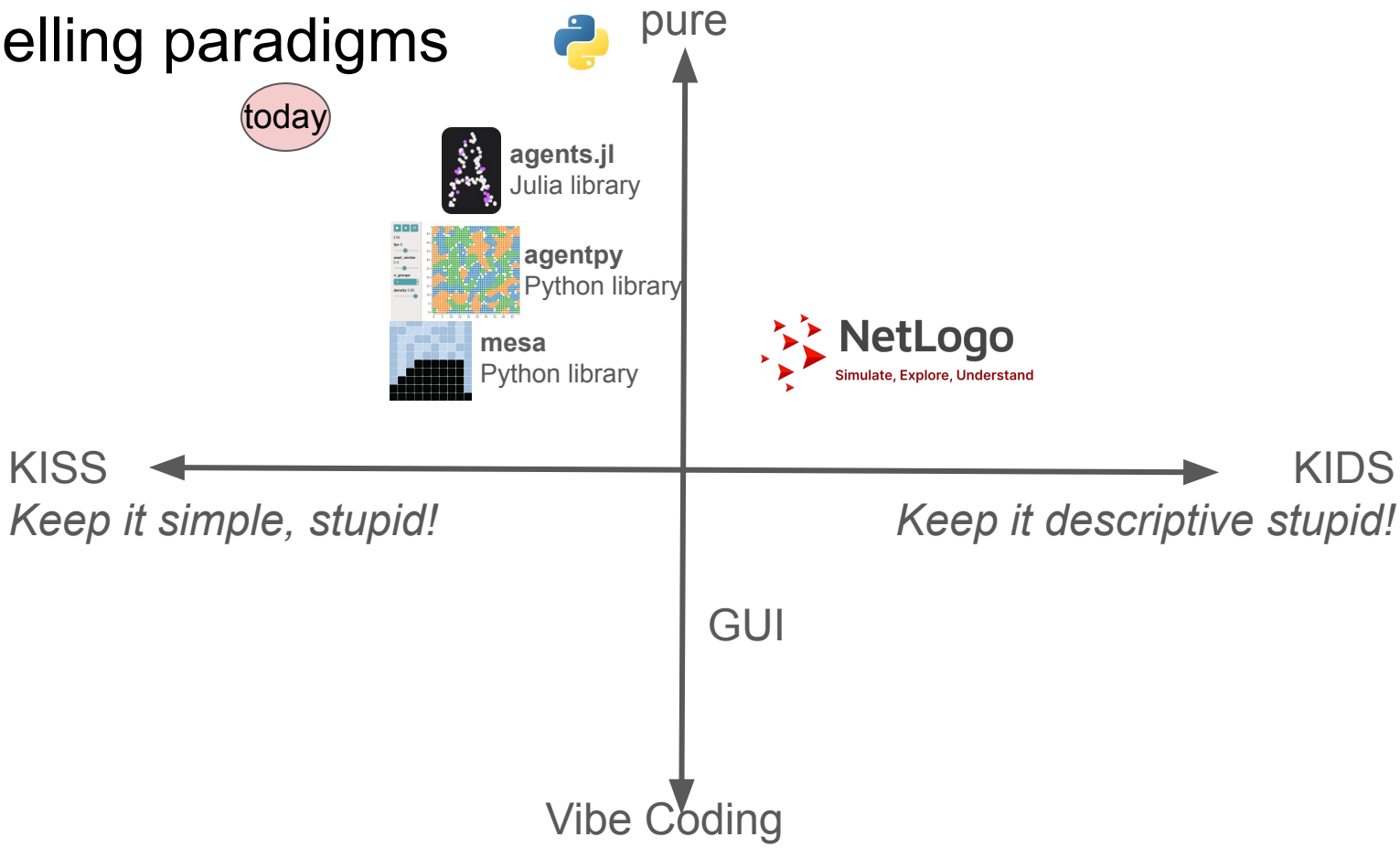
KIDS — *Keep it descriptive simple!*

```
# HEURISTIC
if D(newBelief, agent.BN)  > D(agent.x,
agent.BN):
   pass # reject;
else:
   agent.x = newBelief
```



Edmonds, B. *et al.* (2019) “Different Modelling Purposes,” *Journal of Artificial Societies and Social Simulation*, 22(3), p. 6.

Modelling paradigms



Mathematical modelling for the social sciences

The idea:

Formalise a verbal theory with all its assumptions into a transparent, precise language and simulate the consequences of the theory's explicit and implicit assumptions on some outcome.

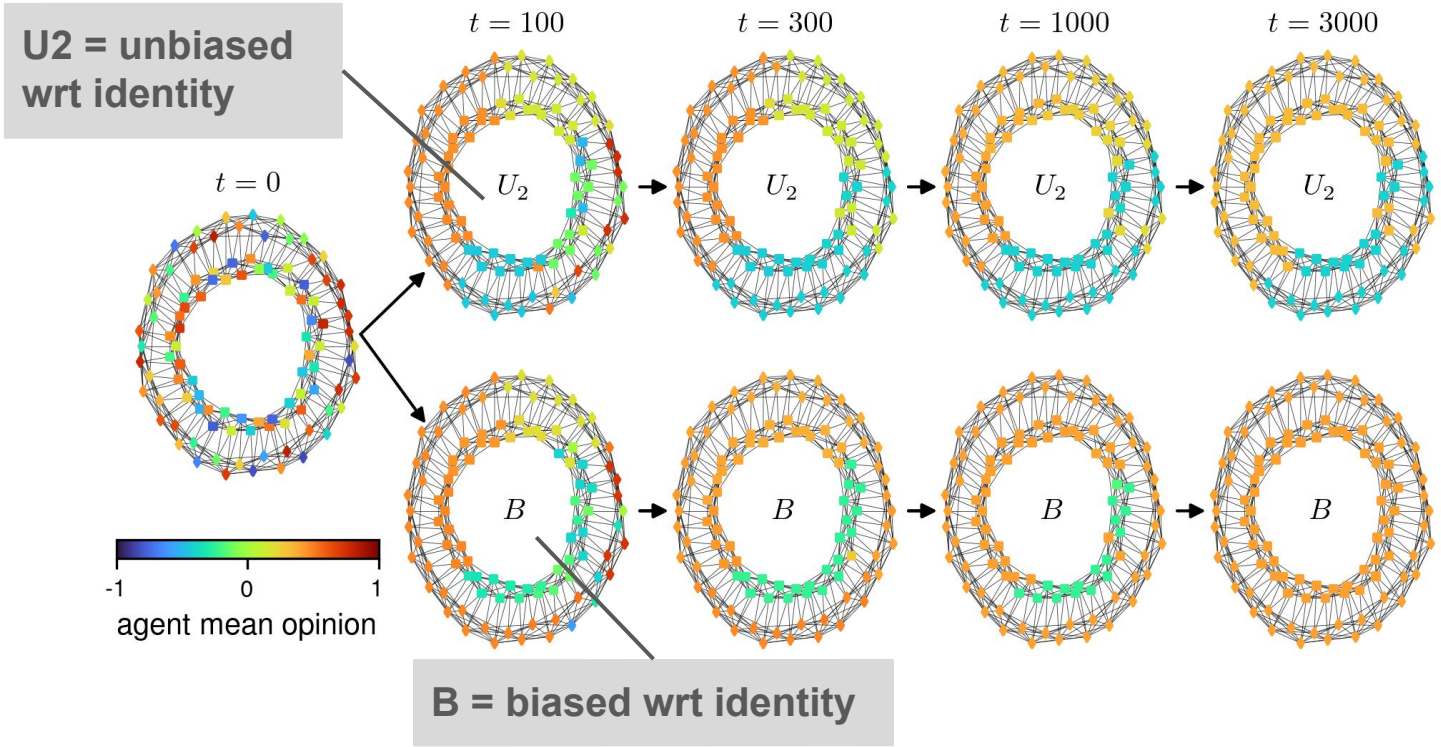
"What if" or "How possibly" science

Example:

>>> Hyp: "In-group bias leads to opinion polarisation between identity groups".

>>> When is this true? How could it possibly *not* be true?

“What if” or “How possibly” science with opinion dynamics models



Consensus *due to* in-group bias?

Steiglechner, P. *et al.* (2023) ‘Social identity bias and communication network clustering interact to shape patterns of opinion dynamics’, *Journal of The Royal Society Interface*, 20(209), p. 20230372. Available at: https://doi.org/10.1098/rsif.2023.0372.

Yeah, modelling is great. But how?

1. Python [1min] ✅
2. Mathematical Modelling [4min] ✅
3. System Dynamics Models with Python [30-45min]
    a. Structure
    b. SIR model (with playtime)
4. ABM with Python [60++min]
    a. Structure
    b. SIR model (with playtime)
    c. Opinion Dynamics Model (with playtime)
    d. Sensitivity Analysis
5. (optional) Visualisation – core parts of python figures [5min]
6. Au revoir and Summary

SYSTEM DYNAMICS MODELS

Differential equations describe system-level variable(s)

(think: modelling temperature dynamics in a heated room)

AGENT BASED MODELS

Heuristic rules/equations describe individual-level variables
Aggregate to get (complex) system dynamics.

(think: modelling opinion dynamics in a human society)

System Dynamics Models

PLAYTIME

Let us solve the equation

*dy/dt = - 2 * y*

over time t=0 to 4 with initial condition y(t=0)=1

Define derivative function

```
def derivative(y , t):
    dydt = -2 * y
    return dydt
```

Use *odeint* to integrate

```
import numpy as np
from scipy.integrate import odeint
t = np.linspace(start=0 , stop=4.0, num=401)
y0 = 1.0
y = odeint(derivative, y0, t)
```

Solve the system:

$$\frac{dx}{dt} = \sigma(y - x),$$

$$\frac{dy}{dt} = x(\rho - z) - y,$$

$$\frac{dz}{dt} = xy - \beta z.$$

over time t with initial conditions x0, y0, z0

Tip:

```
sigma = …
…
def derivative(S , t):
    x,y,z = S # S is a list or array
    dxdt = …
    dydt = …
    …
    return dSdt # list or array
```

Code: 01_sysdyn_models.py

System Dynamics Models

PLAYTIME

Let us solve the equation

*dy/dt = - 2 * y*

over time t=0 to 4 with initial condition y(t=0)=1

Define derivative function

```
def derivative(y , t):
    dydt = -2 * y
    return dydt
```

Use *odeint* to integrate

```
import numpy as np
from scipy.integrate import odeint
t = np.linspace(start=0 , stop=4.0, num=401)
y0 = 1.0
y = odeint(derivative, y0, t)
```

Code: 02_lorenz.py

Solve the system:

$$\frac{dx}{dt} = \sigma(y - x),$$

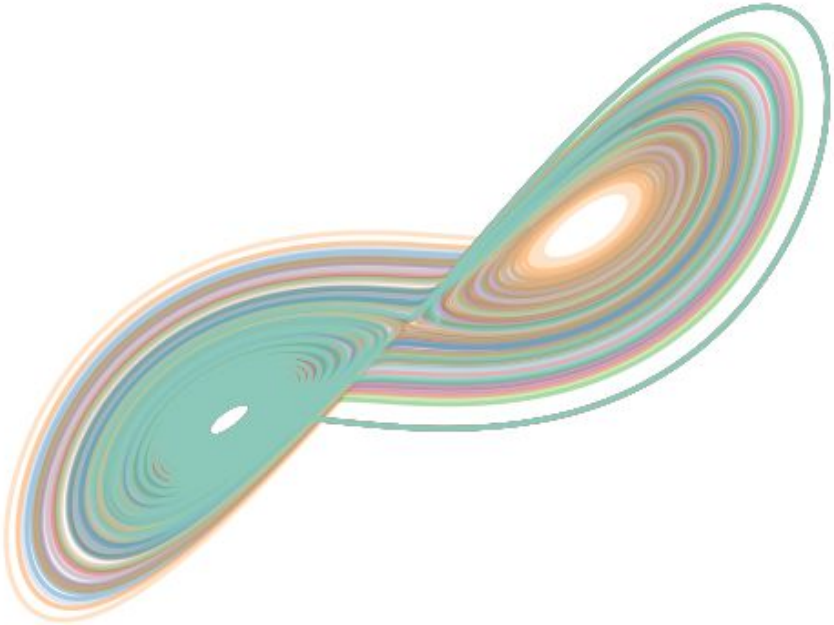
$$\frac{dy}{dt} = x(\rho - z) - y,$$

$$\frac{dz}{dt} = xy - \beta z.$$

over time t with initial conditions x0, y0, z0

```
sigma=1; beta=1; rho=1;
def lorenzsystem(s, t):
    x,y,z = s
    dxdt = sigma * (y-x)
    dydt = x * (rho - z) - y
    dzdt = x*y - beta* z
    system_deriv = [dxdt, dydt, dzdt]
    return system_deriv
t = np.linspace(start=0 , stop=30,
num=10001)
s0 = np.array([0.9,0.,0.])
s = odeint(lorenzsystem, s0, t)
```

Lorenz Butterfly 💙



Code: 02_lorenz.py

```
import matplotlib.pyplot as plt
plt.plot(s[:,0], s[:,1])
→ With different initial conditions
```

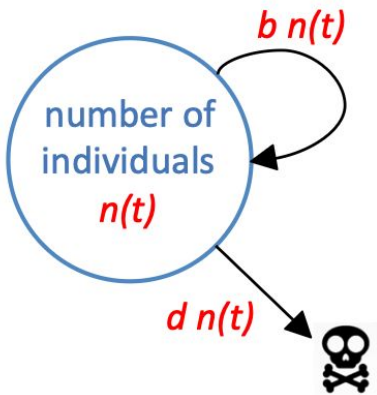
How to construct a model – recipe

1. formulate the question;
2. determine the basic ingredients;
3. qualitatively describe the relevant system;
4. quantitatively describe the relevant system;
5. analyse the equations;
6. checks and balances;
7. relate the results back to the question.

1. How does a population grow?
2. Variable
   n(t)
   Parameters:
   death rate, birth rate
3. *n(t)* grows when existing people reproduce, n(t) reduces when existing people die
4. …
5. …
6. …
7. …

Models of population growth

exponential growth model - flow diagram and equation



$$\frac{dn(t)}{dt} = b\,n(t) - d\,n(t)$$

$$\frac{dn(t)}{dt} = (b - d)\,n(t)$$

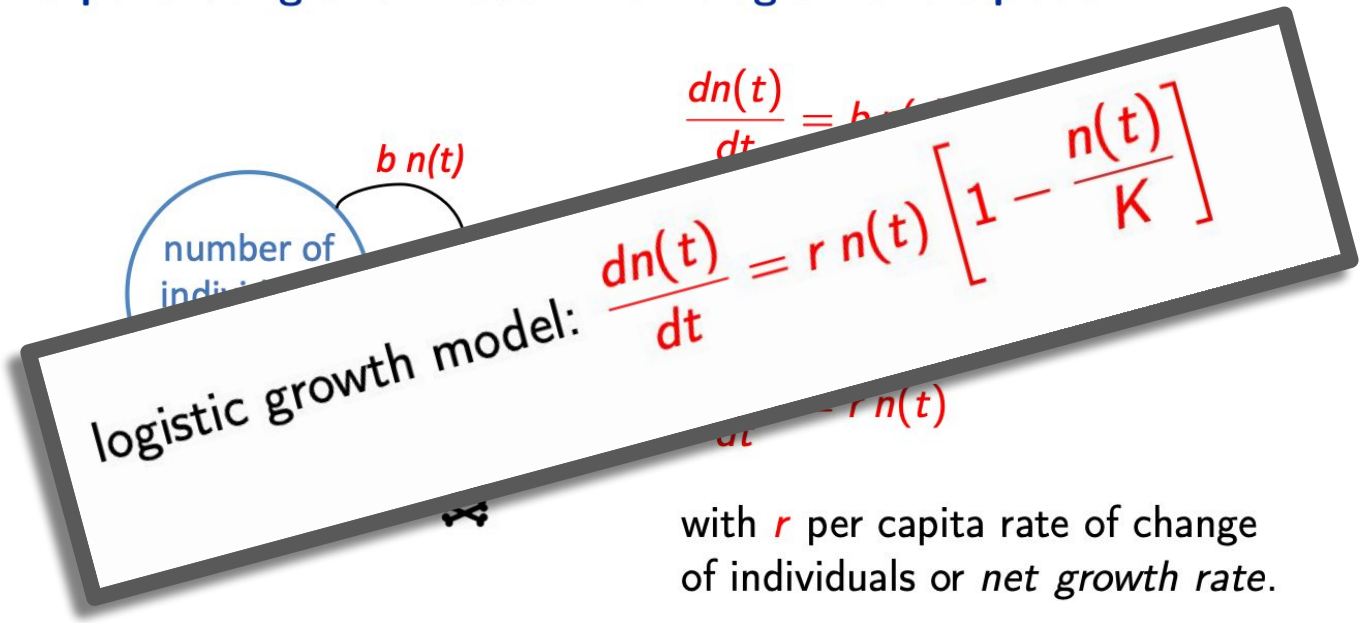
$$\frac{dn(t)}{dt} = r\,n(t)$$

with r per capita rate of change of individuals or *net growth rate*.

Slides from Ago Merico

Models of population growth

exponential growth model - flow diagram and equation



with r per capita rate of change of individuals or *net growth rate*.

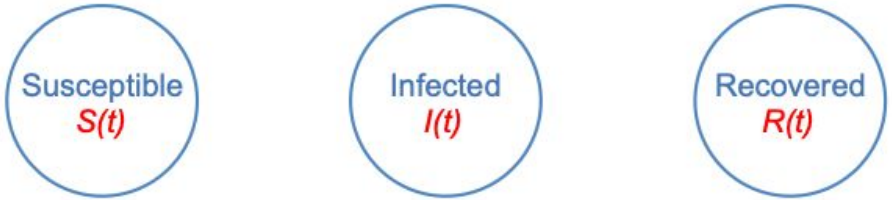
Slides from Ago Merico

Exercise: SIR Model

1. formulate the question;
2. determine the basic ingredients;
3. qualitatively describe the relevant system;
4. quantitatively describe the relevant system;
5. analyse the equations;
6. checks and balances;
7. relate the results back to the question.

Epidemiological model of disease spreading

Susceptible-Infected-Recovered (SIR) model



$$\frac{dS}{dt}$$

$$\frac{dI}{dt}$$

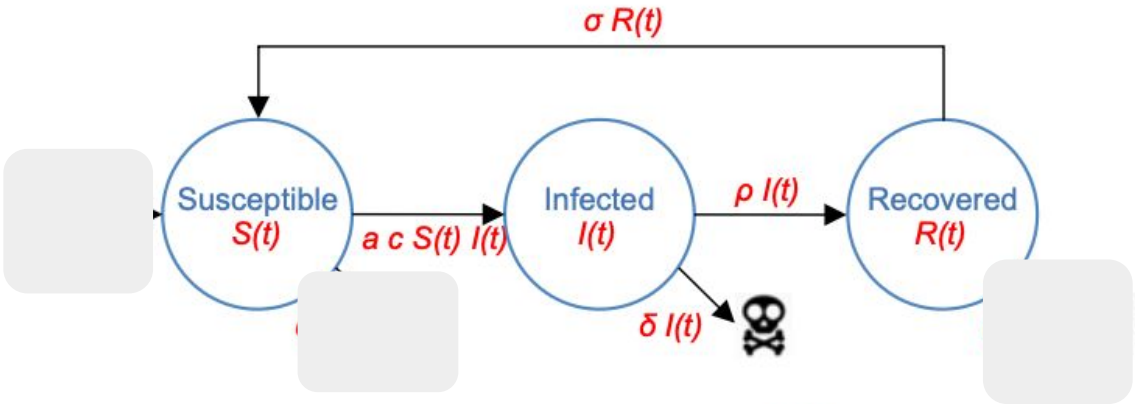
$$\frac{dR}{dt}$$

Slides from Ago Merico

Epidemiological model of disease spreading

Susceptible-Infected-Recovered (SIR) model

PLAYTIME?



$$\frac{dS}{dt} = -a\,c\,S(t)\,I(t) + \sigma\,R(t)$$

$$\frac{dR}{dt} = \rho\,I(t) - \sigma\,R(t)$$

$$\frac{dI}{dt} = a\,c\,S(t)\,I(t) - \delta\,I(t) - \rho\,I(t)$$

c*I(t)= prob contact, a = infection prob, sigma=R2S rate, rho = recovery, delta = mortality

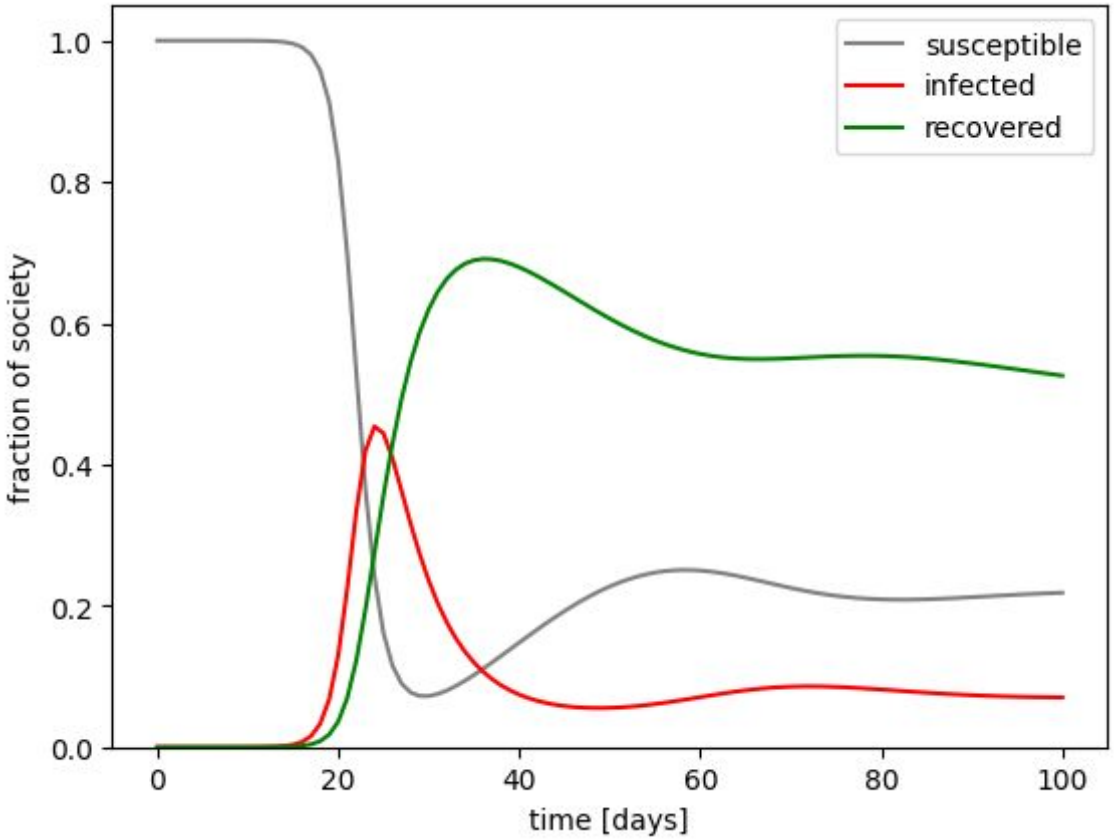
Slides from Ago Merico

SIR model

"Covid"-style epidemic:

```
a = 0.5  # infection prob 50%
c = 2  # meet 2 random people per day
rho = 0.2  # recovery nearly complete
after 14 days
sigma = 0.03  # re-infection (R2S rate)
relatively slow
delta = 0.02  # mortality
```

Code: 03_si_sysdyn.py



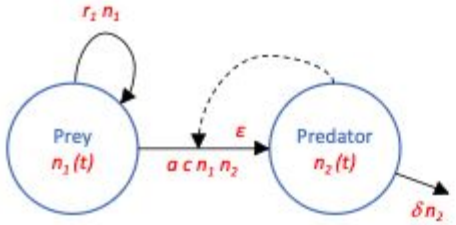
Predator Prey

~ SI - model

Species interaction

predator-prey (or Lotka-Volterra) model - equations

the equations can follow easily from the flow diagram:



$$\frac{dn_1}{dt} = r\,n_1 - a\,c\,n_1\,n_2 \quad \text{(prey)}$$

$$\frac{dn_2}{dt} = \epsilon\,a\,c\,n_1\,n_2 - \delta\,n_2 \quad \text{(predator)}$$

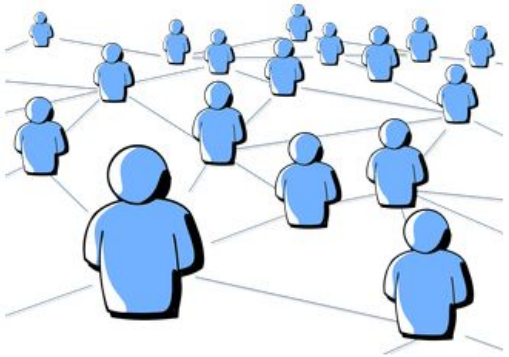
this model produces a very rich set of dynamical behaviours, including cycles over time, which may help explaining the cyclic dynamics of interacting species observed at times in nature.

Slides from Ago Merico

System Dynamics 😮‍💨 → ABM 😌

1. Python [1min] ✅
2. Mathematical Modelling [4min] ✅
3. System Dynamics Models with Python [30-45min] ✅
   a. Structure ✅
   b. SIR model (with playtime) ✅
4. ABM with Python [60++min]
   a. Structure
   b. SIR model (with playtime)
   c. Opinion Dynamics Model (with playtime)
   d. Sensitivity Analysis
   e. (Schelling Model)
5. (optional) Visualisation – core parts of python figures [5min]
6. Au revoir and Summary

Agent Based modelling



Agents (= human individuals)

…with variable attributes

…interact with their environment

…interact with other agents (learning, influencing, adapting)

Used to study the emergence of complex social phenomena.

Simulate conditions which lead to one phenomena or another.

Most Simple Structure of an ABM

- **Define the agent (class)**
- **Initialise agents**
- **Update agents**
- **Observe agents**

and define main loop

Most Simple Structure of an ABM

An agent-based SI model

Code: 06_si_abm.py

Consider *n* agents

Social Network:
Agents have links to other agents. Connected agents may infect each other.

Agent:
An agent is a person that can be susceptible (*S*) or infectious (*I*)

Update rule:
Connected agents meet randomly. When one of them is infectious, there's a probability that the other becomes infected as well.
An infectious agent recovers after t_R time steps.

Class, Objects and ABM structure

Code: 04_abm_structure.py

```
class agent:
    pass

def initialise(params) :
    agents = []
    # ...
    return agents

def update(t, agents, params) :
    # ...
    return agents

def observe(t, agents):
    # ... plot / store the results
    return snap
```

Main loop

```
if __name__ == "__main__":
    t = 0
    T = 100
    params = dict(a=1, b=2, seed=2026)
    np.random.seed(params["seed"])

    agents = initialise(params)
    results = observe(t, agents)
    for t in range(1, T):
        agents = update(t, agents, params)
        results.extend(observe(t, agents))
```

Class definitions (simpler vs. better)

```
class agent:
   pass


ag = agent()
ag.id = 0
ag.state = 1
ag.nbs = [0,17,18,99]
```

Define an empty class

Create an object and store a reference to it in variable *ag*

Access the attribute *ag.id,* then write with *0*

…

Objects can have attributes (age, opinion, …) and functions (increase age of *self*)

Class definitions (simpler vs. better)

```
class agent:
   pass


ag = agent()
ag.id = 0
ag.state = 1
ag.nbs = [0,17,18,99]
```

```
class agent:
   def __init__(self, id, nbs):
       self.id = id
       self.state = 0
       self.nbs = nbs


ag = agent(0, [0,17,18,99])
```

Initialise

```
def initialise(params):
    agents = []
    # ...
    return agents
```

Create a list *agents* of *params["n"]* agents (ie., objects of class *agent*) with unique identifiers *id*

Lv 1:

Connect them to the "next agent" (*ag.nbs = [id+1]*)

Lv 2:

Connect them in a random network (*ag.nbs = [...]*)

Lv 3:

Connect them in a network using *networkx*'s
G = nx.?? ; ag.nbs = list(G.neighbours(n))

Update

```
def update(t, agents, params) :
    # ...
    return agents
```

Lv 1:

Select an agent and update its attribute

Lv 2:

Select an agent and one of its neighbour
Update the attribute of the first/both agents

Lv 3:

Iterate through the list of agents randomly
For each agent, select a neighbour and update their attributes

Observe / Main Loop

```
def observe(t, agents):
    snap = [
      [t, ag.id, ag.state]
      for ag in agents
    ]
    return snap
```

Observe should return a *list* of all agent variables you are interested in, together with current *time* and the *id*s of the respective agents.

```
np.random.seed(params["seed"])
agents = initialise(params)
results = observe(t, agents)
for t in range (1,T):
     agents = update(t, agents, params)
     results.extend(observe(t, agents))
resultColumns = ["t", "id", "state"]
df = pd.DataFrame(results, columns=resultColumns)
```

~ decent setup to manage simulated data

SUMMARY

Class, Objects and ABM structure

```
class agent:
    pass

def initialise(params) :
    agents = []
    # ...
    return agents

def update(t, agents, params) :
    # ...
    return agents

def observe(t, agents):
    # ... plot / store the results
    return snap
```

Main loop

```
if __name__ == "__main__":
    t = 0
    T = 100
    params = dict(a=1, b=2, seed=2026)
    np.random.seed(params["seed"])

    agents = initialise(params)
    results = observe(t, agents)
    for t in range(1, T):
        agents = update(t, agents, params)
        results.extend(observe(t, agents))
```

Class definitions (simpler vs. better)

```
class agent:
    pass
```

```
ag = agent()
ag.id = 0
ag.state = 1
ag.nbs = [0,17,18,99]
```

```
class agent:
    def __init__(self, id, nbs):
        self.id = id
        self.state = 0
        self.nbs = nbs
```

```
ag = agent(0, [0,17,18,99])
```

Initialise

```
def initialise(params):
    agents = []
    # ...
    return agents
```

Create a list *agents* of *params["n"]* agents (ie., objects of class *agent*) with unique identifiers *id*

Lv 1:
Connect them to the "next agent" (*ag.nbs = [id+1]*)

Lv 2:
Connect them in a random network (*ag.nbs = [...]*)

Lv 3:
Connect them in a network using *networkx*'s
G = nx.?? ; ag.nbs = list(G.neighbours(n))

Update

```
def update(t, agents,
params) :
    # ...
    return agents
```

Lv 1:
Select an agent and update its attribute

Lv 2:
Select an agent and one of its neighbour
Update the attribute of the first/both agents

Lv 3:
Iterate through the list of agents randomly
For each agent, select a neighbour and
update their attributes

Observe / Main Loop

```
def observe(t, agents):
    snap = [
      [t, ag.id, ag.state]
      for ag in agents
    ]
    return snap
```

Observe should return a *list* of the variables you are interested in, together with current *time* and *id* of the respective agents.

```
np.random.seed(params["seed"])
agents = initialise(params)
results = observe(t, agents)
for t in range (1,T):
    agents = update(t, agents, params)
    results.extend(observe(t, agents))
resultColumns = ["t", "id", "state"]
df = pd.DataFrame(results, columns=resultColumns)
```

This is a good setup to create dataframes

An agent-based SI model

Code: 06_si_abm.py

Consider *n* agents

Social Network:
Agents have links to other agents. Connected agents may infect each other.

Agent:
An agent is a person that can be susceptible (*S*) or infectious (*I*)

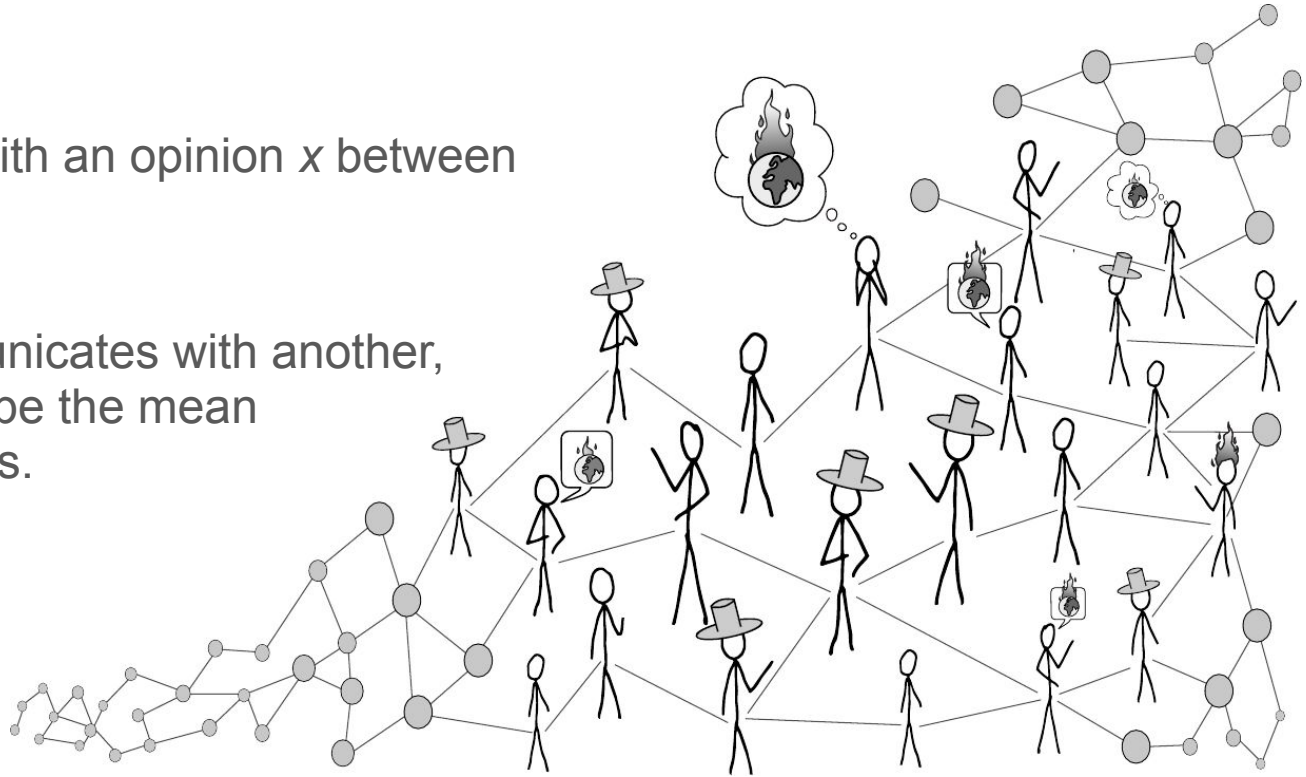
Update rule:
Connected agents meet randomly. When one of them is infectious, there's a probability that the other becomes infected as well.
An infectious agent recovers after t_R time steps.

A simple opinion dynamics model

Consider *n* agents

Agent:
An agent is a person with an opinion *x* between 0 and 1.

Update rule:
When an agent communicates with another, their new opinions will be the mean of the two prior opinions.



Break and Remaining things

1. Python [1min] ✅
2. Mathematical Modelling [4min] ✅
3. System Dynamics Models with Python [30-45min] ✅
    a. Structure ✅
    b. SIR model (with playtime) ✅
4. ABM with Python [60++min]
    a. Structure ✅
    b. SIR model (with playtime) ✅
    c. Opinion Dynamics Model (with playtime)
    d. Sensitivity Analysis
    e. (Schelling Model)
5. (optional) Visualisation – core parts of python figures [5min]
6. Au revoir and Summary

Opinion Dynamics Playtime extended

Code: 07_opiniondynamics_abm.py

Lv 1:

Create ABM structure, implement functions for simplest OD model, store dataframe

Lv 2:

Build ABM as above. Set up runs for multiple seeds and store as separate files.

Lv 3:

Think about science questions, implement extensions (e.g. using a boolean param)

Lv x:

Use the xarray or dataframes to plot results.

Sensitivity analysis

Code: 05_parallel.py

```
from joblib import Parallel, delayed
import multiprocessing
def run_simulation(params):
    # init
    # main loop
    filename = f"results_a{params['a']}_{params['seed']}.csv"
    # store dataframe as csv.
    return

params_SensAna = [dict(a = a_value ,seed=seed) for a_value in
[1,2,3] for seed in range(10)]
Parallel(n_jobs=max(1, multiprocessing.cpu_count() - 2))(
        delayed(run_simulation)(params)
        for params in params_SensAna
    )
```

Sensitivity analysis II

```
import xarray as xr
ds_all = []
for params in params_SensAna:
    filename = (
        f"results_r{params['recover_prob']}"
         f"_i{params['infect_prob']}"
         f"_{params['seed']}.csv"
    )
    df = pd.read_csv(filename, index_col=0)
    ds = xr.Dataset.from_dataframe(
         df.set_index(["t", "id"])
         )
    ds = ds.expand_dims({k: [v] for k, v in
params.items()})
    ds_all.append(ds)
ds_all = xr.combine_by_coords(ds_all)
```

xarray.Dataset

▶ Dimensions: (n: 1, infect_prob: 3, recover_prob: 3, infected0: 1, link_prob: 1, seed: 10, t: 100, id: 100)

▼ Coordinates:

n	(n)	int64	100
infect_prob	(infect_prob)	float64	0.05 0.1 0.2
recover_prob	(recover_prob)	float64	0.05 0.1 0.2
infected0	(infected0)	int64	2
link_prob	(link_prob)	float64	0.2
seed	(seed)	int64	0 1 2 3 4 5 6 ...
t	(t)	int64	0 1 2 3 4 5 6 ...
id	(id)	int64	0 1 2 3 4 5 6 ...

▼ Data variables:

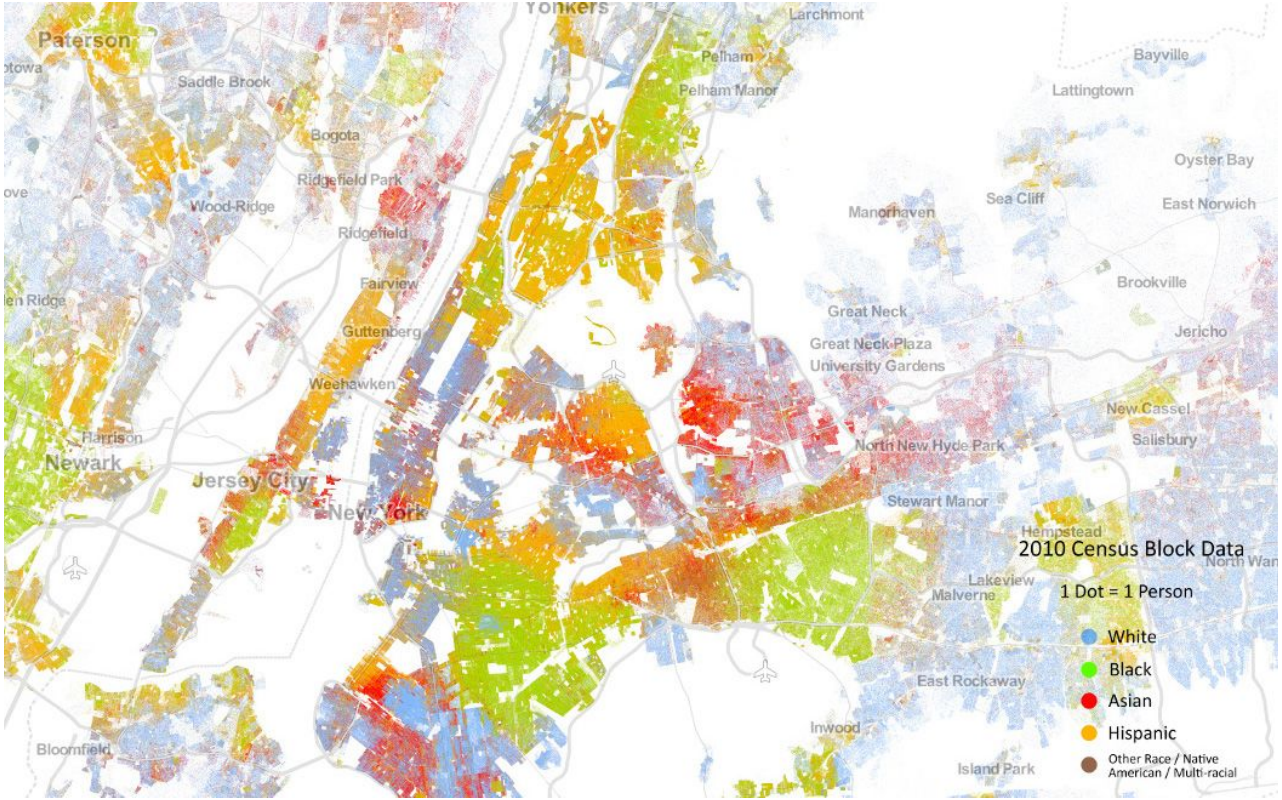
state	(n, infect_prob, recover_prob, infected0, link_prob, seed, t, id)	int64	0 0 0 0 0 0 0...

```
array([[[[[[[[0, 0, 0, ..., 0, 0, 0],
             [0, 0, 0, ..., 0, 0, 0],
             [0, 0, 0, ..., 0, 0, 0],
             ...,
             [0, 0, 0, ..., 0, 0, 0],
             [0, 0, 0, ..., 0, 0, 0],
             [0, 0, 0, ..., 0, 0, 0]],

            [[0, 0, 0, ..., 0, 0, 0],
             [0, 0, 0, ..., 0, 0, 0],
             [0, 0, 0, ..., 0, 0, 0],
             ...,
             [0, 0, 0, ..., 0, 0, 0],
             [0, 0, 0, ..., 1, 0, 0],
```

Code: 05_parallel.py

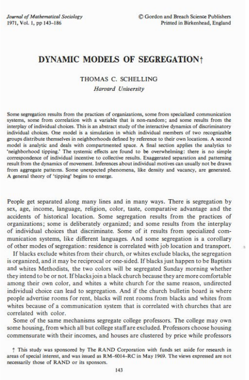
A model of segregation



Why is there so much spatial segregation between ethnicities?

Are humans simply extremely intolerant?

Thomas Schelling, 1971



Journal of Mathematical Sociology
1971, Vol. 1, pp 143–186

© Gordon and Breach Science Publishers
Printed in Birkenhead, England

DYNAMIC MODELS OF SEGREGATION†

THOMAS C. SCHELLING
Harvard University

Some segregation results from the practices of organizations, some from specialized communication systems, some from correlation with a variable that is non-random; and some results from the interplay of individual choices. This is an abstract study of the interactive dynamics of discriminatory individual choices. One model is a simulation in which individual members of two recognizable groups distribute themselves in neighborhoods defined by reference to their own locations. A second model is analytic and deals with compartmented space. A final section applies the analytics to 'neighborhood tipping.' The systemic effects are found to be overwhelming: there is no simple correspondence of individual incentive to collective results. Exaggerated separation and patterning result from the dynamics of movement. Inferences about individual motives can usually not be drawn from aggregate patterns. Some unexpected phenomena, like density and vacancy, are generated. A general theory of 'tipping' begins to emerge.

People get separated along many lines and in many ways. There is segregation by sex, age, income, language, religion, color, taste, comparative advantage and the accidents of historical location. Some segregation results from the practices of organizations; some is deliberately organized; and some results from the interplay of individual choices that discriminate. Some of it results from specialized communication systems, like different languages. And some segregation is a corollary of other modes of segregation: residence is correlated with job location and transport.

If blacks exclude whites from their church, or whites exclude blacks, the segregation is organized, and it may be reciprocal or one-sided. If blacks just happen to be Baptists and whites Methodists, the two colors will be segregated Sunday morning whether they intend to be or not. If blacks join a black church because they are more comfortable among their own color, and whites a white church for the same reason, undirected individual choice can lead to segregation. And if the church bulletin board is where people advertise rooms for rent, blacks will rent rooms from blacks and whites from whites because of a communication system that is correlated with churches that are correlated with color.

Some of the same mechanisms segregate college professors. The college may own some housing, from which all but college staff are excluded. Professors choose housing commensurate with their incomes, and houses are clustered by price while professors

† This study was sponsored by The RAND Corporation with funds set aside for research in areas of special interest, and was issued as RM-6014-RC in May 1969. The views expressed are not necessarily those of RAND or its sponsors.

143

A model of segregation – Schelling (1971)

- People have one of two types (“white”/”PoC”)
- They are happy with the place they live in as long as there are enough people of the same type in their neighbourhood. If they are unhappy, they will move.

RQ: How intolerant do people need to be to produce the observed strong segregation?

A simplified Schelling model

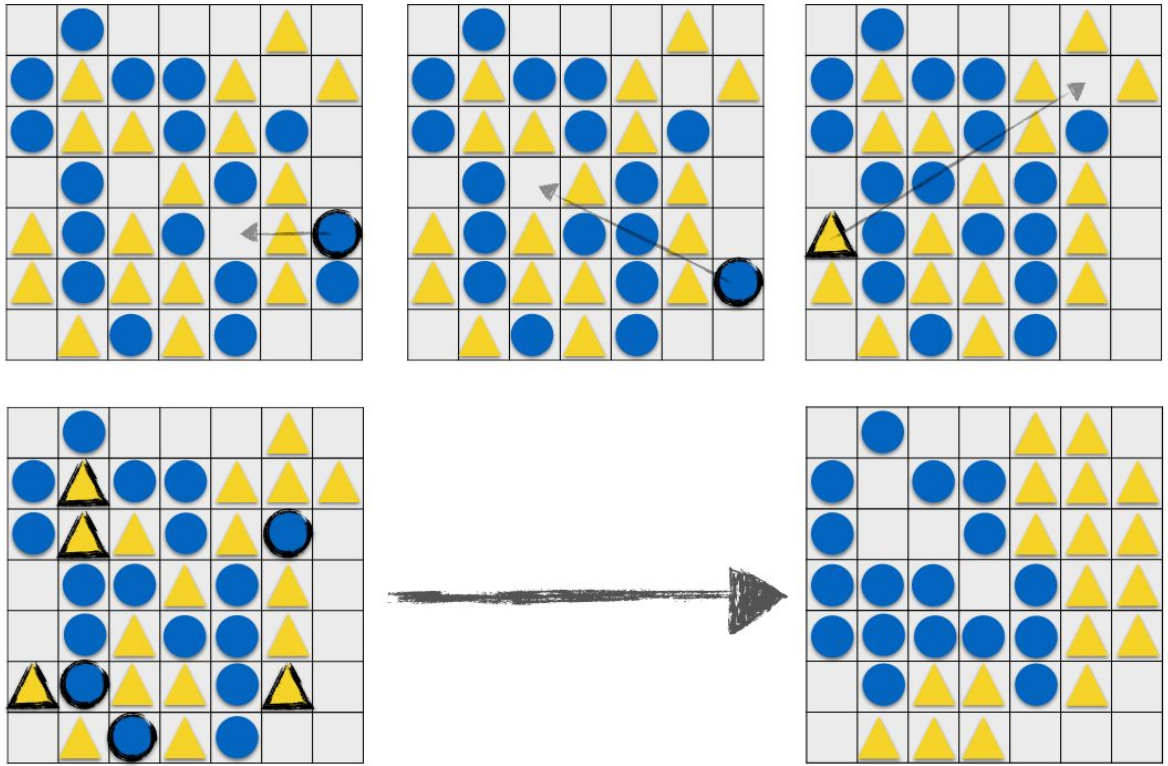
Each agent with type red/blue “lives” in a position on the map (x,y)

They observe the types of neighbours in a circle with radius r around them.

If the fraction of neighbours with the same type is high enough (happy), they stay.
If not (unhappy), they move to a new random location.

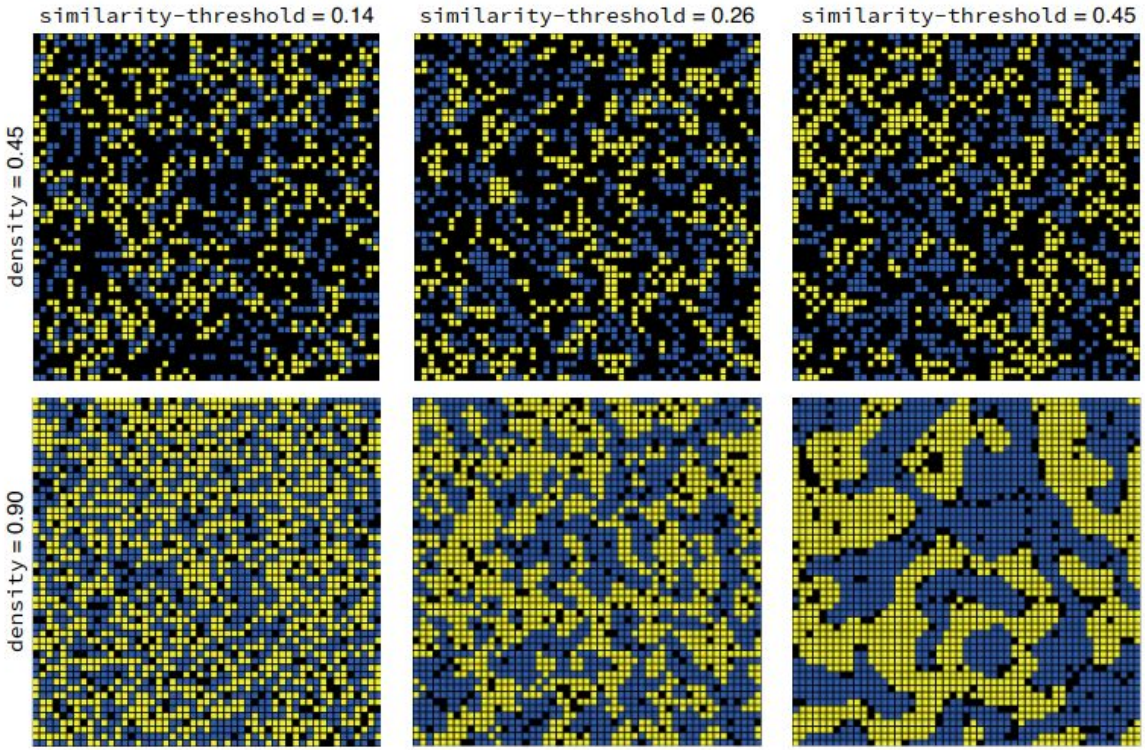
Code: 08_schelling.py

Schelling model on a grid



From Paul Smaldino

Schelling model on a grid



people with "mild" in-group preference towards their own group could still lead to a highly segregated society

From Paul Smaldino

Break and Remaining things

1. Python [1min] ✅
2. Mathematical Modelling [4min] ✅
3. System Dynamics Models with Python [30-45min] ✅
    a. Structure ✅
    b. SIR model (with playtime) ✅
4. ABM with Python [60++min]
    a. Structure ✅
    b. SIR model (with playtime) ✅
    c. Opinion Dynamics Model (with playtime) ✅
    d. Sensitivity Analysis ❓
    e. (Schelling Model) ❓
5. (optional) Visualisation – core parts of python figures [5min]
6. Au revoir and Summary

Visualisation – structure in python

Create canvas

⟶ argument: figsize=(16,9)

```
1  fig = plt.figure()
2  ax1 = fig.add_subplot(1,2,1)  # args = nrows, ncolumns, index
3  ax1.hist(initial_opinions, bins=np.linspace(0,1,21))
4  ax1.set_xlim(0,1)
5  ax1.set_ylim(0,)
6  ax1.set_xticks(np.linspace(0,1.0,5))
7  ax1.set_xlabel("opinion")
8  ax1.set_ylabel("frequency")
9  ax1.set_title("time 0")
10
11 ax2 = fig.add_subplot(1,2,2)
12 ax2.hist(final_opinions, bins=np.linspace(0,1,21))
13 ax2.set_xlim(0,1)
14 ax2.set_ylim(0,)
15 ax2.set_xticks(np.linspace(0,1.0,5))
16 ax2.set_xlabel("opinion")
17 #ax2.set_ylabel("frequency")
18 ax2.set_title(f"time {times[-1]}")
19
20 fig.tight_layout()
21 # plt.savefig("simple-opinion-dynamics.pdf")
```

Visualisation – structure in python

Create axes
(panels in a figure)

→ single axes:
ax = plt.axes()

```
1  fig = plt.figure()
2  ax1 = fig.add_subplot(1,2,1)  # args = nrows, ncolumns, index
3  ax1.hist(initial_opinions, bins=np.linspace(0,1,21))
4  ax1.set_xlim(0,1)
5  ax1.set_ylim(0,)
6  ax1.set_xticks(np.linspace(0,1.0,5))
7  ax1.set_xlabel("opinion")
8  ax1.set_ylabel("frequency")
9  ax1.set_title("time 0")
10
11 ax2 = fig.add_subplot(1,2,2)
12 ax2.hist(final_opinions, bins=np.linspace(0,1,21))
13 ax2.set_xlim(0,1)
14 ax2.set_ylim(0,)
15 ax2.set_xticks(np.linspace(0,1.0,5))
16 ax2.set_xlabel("opinion")
17 #ax2.set_ylabel("frequency")
18 ax2.set_title(f"time {times[-1]}")
19
20 fig.tight_layout()
21 # plt.savefig("simple-opinion-dynamics.pdf")
```

Visualisation – structure in python

Plotting:

ax.**plot**(x, y)
ax.**scatter**(xArr, yArr)
ax.**hist**(x, samples)

ax.pcolormesh(X,Y,Z)

Arguments in plot:
- color and alpha
- lw (linewidth)
- ls (linestyle)
- marker=”o” & ms (size)
- label

```
1  fig = plt.figure()
2  ax1 = fig.add_subplot(1,2,1)  # args = nrows, ncolumns, index
3  ax1.hist(initial_opinions, bins=np.linspace(0,1,21))
4  ax1.set_xlim(0,1)
5  ax1.set_ylim(0,)
6  ax1.set_xticks(np.linspace(0,1.0,5))
7  ax1.set_xlabel("opinion")
8  ax1.set_ylabel("frequency")
9  ax1.set_title("time 0")
10
11 ax2 = fig.add_subplot(1,2,2)
12 ax2.hist(final_opinions, bins=np.linspace(0,1,21))
13 ax2.set_xlim(0,1)
14 ax2.set_ylim(0,)
15 ax2.set_xticks(np.linspace(0,1.0,5))
16 ax2.set_xlabel("opinion")
17 #ax2.set_ylabel("frequency")
18 ax2.set_title(f"time {times[-1]}")
19
20 fig.tight_layout()
21 # plt.savefig("simple-opinion-dynamics.pdf")
```

Visualisation – structure in python

Aesthetics:

- Limits

- Ticks (and ticklabels)

- Axis labels

- Legend

- Title

- layout & save
(fig.subplots_adjust(left, right, bottom, top))

```
1  fig = plt.figure()
2  ax1 = fig.add_subplot(1,2,1)  # args = nrows, ncolumns, index
3  ax1.hist(initial_opinions, bins=np.linspace(0,1,21))
4  ax1.set_xlim(0,1)
5  ax1.set_ylim(0,)
6  ax1.set_xticks(np.linspace(0,1.0,5))
7  ax1.set_xlabel("opinion")
8  ax1.set_ylabel("frequency")
9  ax1.set_title("time 0")
10
11 ax2 = fig.add_subplot(1,2,2)
12 ax2.hist(final_opinions, bins=np.linspace(0,1,21))
13 ax2.set_xlim(0,1)
14 ax2.set_ylim(0,)
15 ax2.set_xticks(np.linspace(0,1.0,5))
16 ax2.set_xlabel("opinion")
17 #ax2.set_ylabel("frequency")
18 ax2.set_title(f"time {times[-1]}")
19
20 fig.tight_layout()
21 # plt.savefig("simple-opinion-dynamics.pdf")
```

Visualisation - Extras

- plt.subplot_mosaic → for funky layouts of subplots
- ax.text(x, y, “bla”, fontsize=10, va=”center”, ha=”left”, transform=ax.transAxes)

  E.g. ax.text(0.05, 0.95, “A”, fontsize=10, va=”top”, ha=”left”, transform=ax.transAxes) for figure panels

  E.g. ax.text(someX, opinions[ag0][someX]), “agent 0”, va=”bottom”, ha=”center”) for annotating lines

- ax2.sharey(ax1)
- For the rest: 

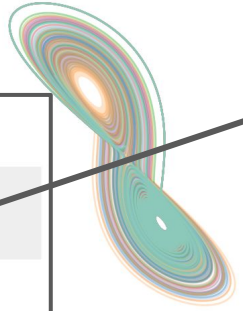
it.

- Inspiration from 

Break and Remaining things

1. Python [1min] ✅
2. Mathematical Modelling [4min] ✅
3. System Dynamics Models with Python [30-45min] ✅
    a. Structure ✅
    b. SIR model (with playtime) ✅
4. ABM with Python [60++min]
    a. Structure ✅
    b. SIR model (with playtime) ✅
    c. Opinion Dynamics Model (with playtime) ✅
    d. Sensitivity Analysis ✅
    e. (Schelling Model) ✅
5. (optional) Visualisation – core parts of python figures [5min] ✅
6. Au revoir and Summary

Take Homes: Mathematical Modelling with python



SYSTEM DYNAMICS MODELS

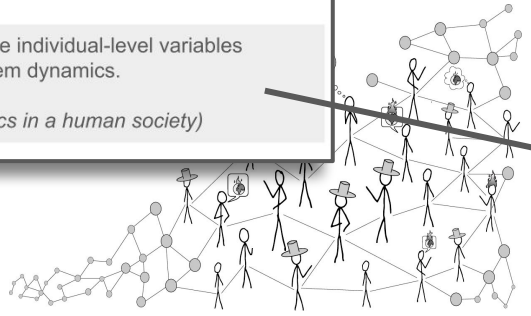
Differential equations describe system-level variable(s)

(think: modelling temperature dynamics in a heated room)

AGENT BASED MODELS

Heuristic rules/equations describe individual-level variables
Aggregate to get (complex) system dynamics.

(think: modelling opinion dynamics in a human society)



system: ***dy/dt = k * y***

```python
def derivative(y , t):
    dydt = k * y
    return dydt

from scipy.integrate import odeint
t = np.linspace(start=0 , stop=4.0, num=401)
y0 = 1.0
y = odeint(derivative, y0, t)
```

Class, Objects and ABM structure

```python
class agent:
    pass

def initialise(params) :
    agents = []
    # ...
    return agents

def update(t, agents, params) :
    # ...
    return agents

def observe(t, agents):
    # ... plot / store the results
    return snap
```

Main loop

```python
if __name__ == "__main__":
    t = 0
    T = 100
    params = dict(a=1, b=2, seed=2026)
    np.random.seed(params["seed"])

    agents = initialise(params)
    results = observe(t, agents)
    for t in range(1, T):
        agents = update(t, agents, params)
        results.extend(observe(t, agents))
```

Today's Applications:

> Infection Dynamics

> Opinion Dynamics

+ Sensitivity Analysis, "parallelisation"
+ Visualisation

APPENDIX

BASIC PYTHON EXERCISES

- Play with lists, dictionaries, strings, arrays … and the basic syntax.
- Tackle the following problems:
  1. Missing data:
     define an array with some missing data points (np.nan). Use the mean of the remaining data to fill these entries in the array
  2. Similarity:
     build a function that takes two binary arrays and returns the number of equivalent bits
  3. Numerical encoding:
     transform 5-point Likert-scale responses (“Strongly agree”, “Agree”, …”Strongly disagree”) into numeric values, -2, -1, …, e.g. using a lookup table (via a dictionary).
  4. Remove outliers:
     build a function that takes a numeric array and return all values that lie between a specified min and max value
  5. create a list containing several different data types. Then create a “cleaner” function that returns all the elements of the list that are integer/float (check "type(...)")

Workflow

- Editor:
  - Virtual Studio Code
  - Spyder
  - pycharm
- Jupyter Notebook vs. python-script
  - separate simulation/data curation vs. analysis/visualisation
- High-performance clusters
- Version control: github
- Folder structure
- Debugging + Error message

Data Science <3 python <3 Data Science

- Pandas is great!

- Many of the data science hacks can be done in R & often more out-of-the-box
  But:
  - Python is a general-purpose tool → you can more easily combine data analysis with other tasks
  - Python is extremely common in academia, business, … huge community.
  - Python is fast (for its simplicity).
- Any data science project is:
  90% data preparation (wrangling, modelling, …),
  10% running a statistical model (Machine Learning today is a one-liner).

Pandas dataframe

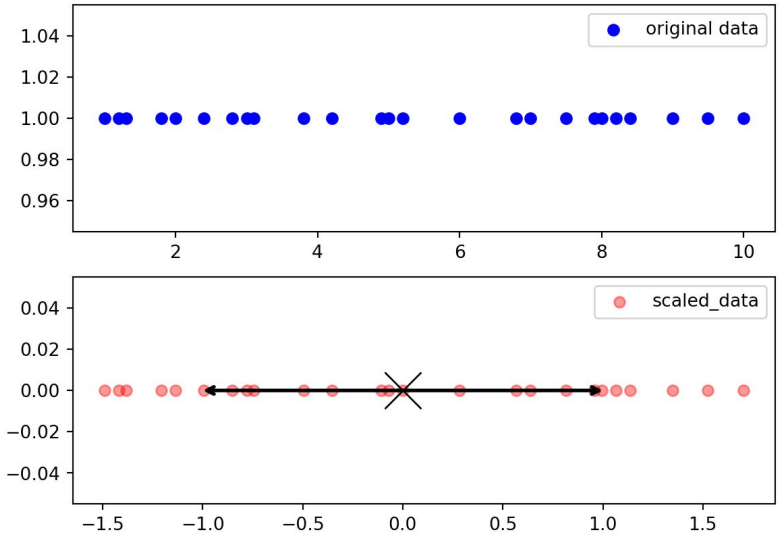
Similar to tables in excel (or R dataframe)

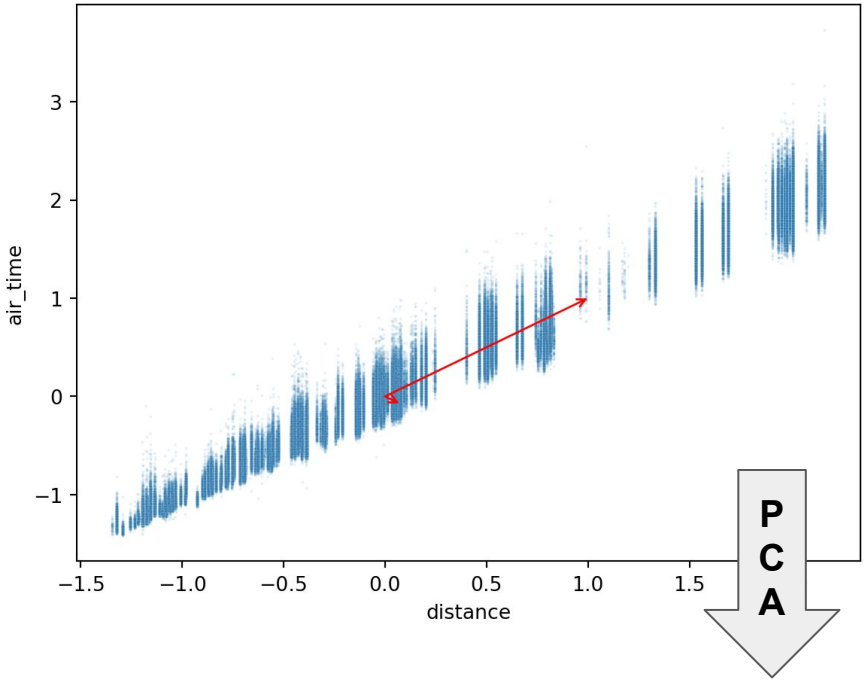
	age	party_feel_closest	SPD	CDU	Greens	FDP	AfD	CSU	Left Party
24849	25	SPD	8	8	7	7	0	7	3
24850	20	FDP	4	2	2	10	0	2	1
24851	30	Greens	6	2	9	4	0	1	8
24852	38	SPD	8	2	8	1	0	1	4
24853	57	NaN	7	8	6	0	0	8	0

- Data import
- Data transformation
  - Filtering, selecting, indexing, sorting
  - Missing data
  - Inspecting (value_counts, head, summary stats)
  - grouping

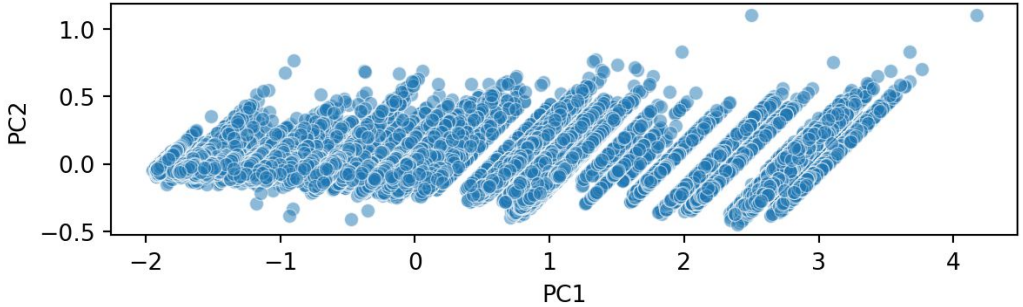
datascience-basics.ipynb

Preprocessing: scaling and PCA





PCA

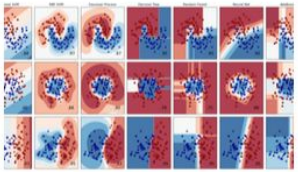


Statistics Models – the sklearn way

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: Gradient boosting, nearest neighbors, random forest, logistic regression, and more...

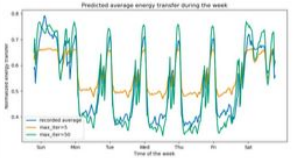


Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, stock prices.
Algorithms: Gradient boosting, nearest neighbors, random forest, ridge, and more...



Examples

Clustering

Automatic grouping of similar objects into sets.

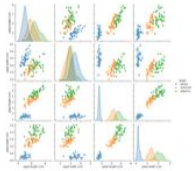
Applications: Customer segmentation, grouping experiment outcomes.
Algorithms: k-Means, HDBSCAN, hierarchical clustering, and more...

Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, increased efficiency.
Algorithms: PCA, feature selection, non-negative matrix factorization, and more...

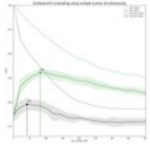


Examples

Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning.
Algorithms: Grid search, cross validation, metrics, and more...

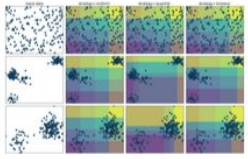


Examples

Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: Preprocessing, feature extraction, and more...



Examples

Statistics Models – the sklearn way

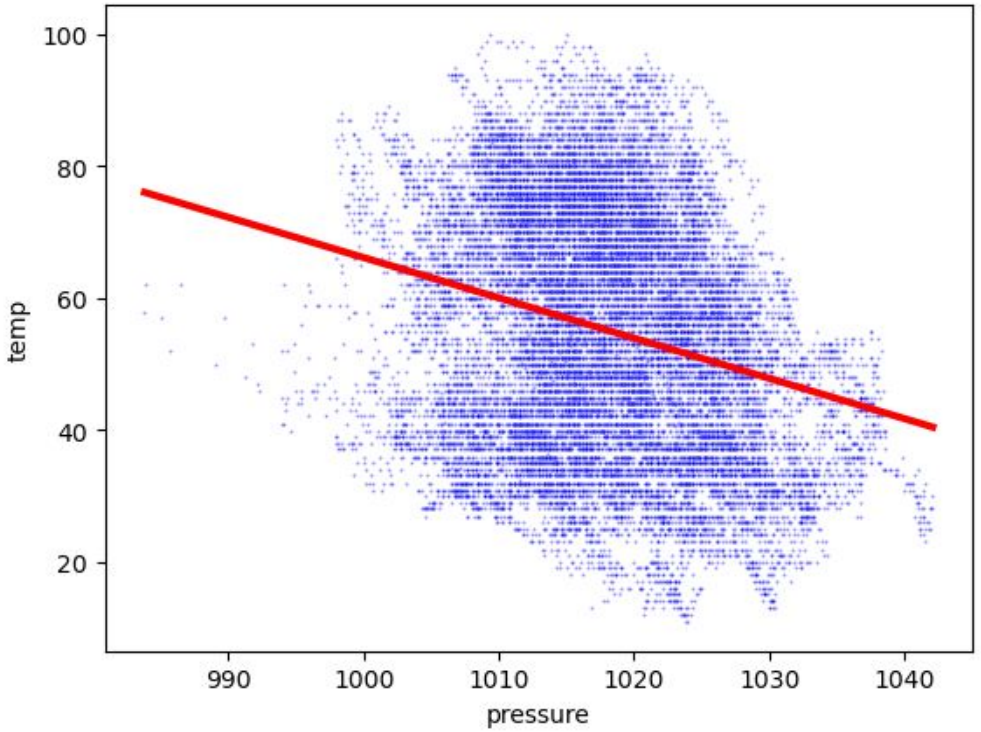
1. choose model class
2. (choose model hyperparameters)
3. arrange data into a features matrix, X, and target vector, y.
4. Split data into training and test data
5. fit the model to your data by calling the .fit() method of the model instance.
6. use .transform() or .predict() to apply the model to (new) data.

```
1  import pandas as pd
2  from sklearn.preprocessing import StandardScaler
3
4  X = pd.DataFrame({"a":[1,2,3,4,5,7,9,11,13]})
5  model = StandardScaler()
6  model.fit(X)
7  standardized_X = model.transform(X)
8
```

Linear Modelling

Y = intercept + coeff * X

Here: higher pressure is associated with decreased temperature (-0.6 °F per hPa)



Playtime: Possible Projects

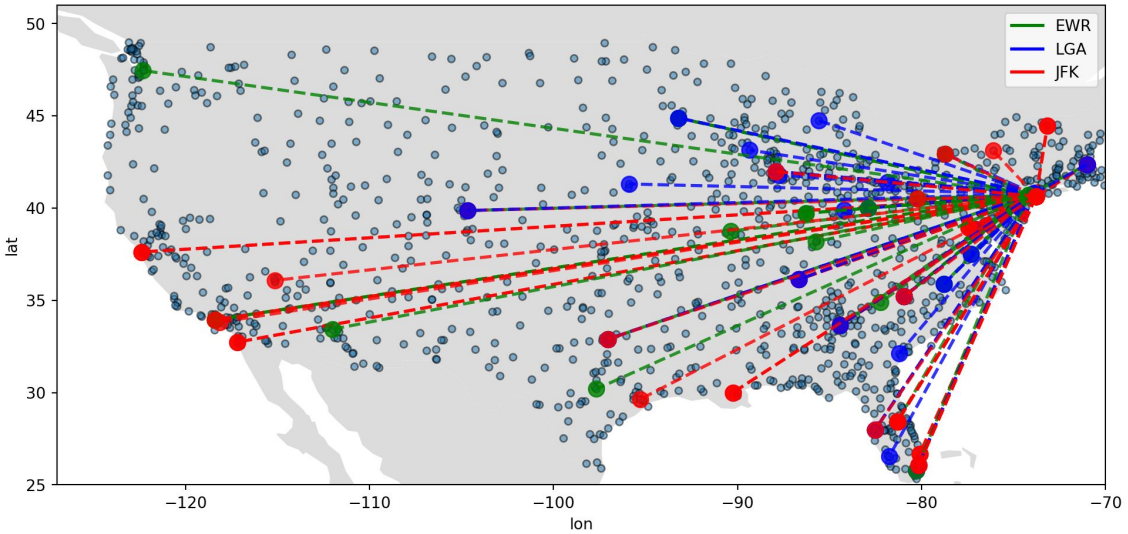
Datasets:

1. Flights from/to New York City airports in 2013 (example dataset in pandas)
2. European Social Survey (Demographics + Habits + Attitudes + Values?)
3. Comparative Survey of Electoral Systems (How much do you like party X?)
4. Your own dataset?

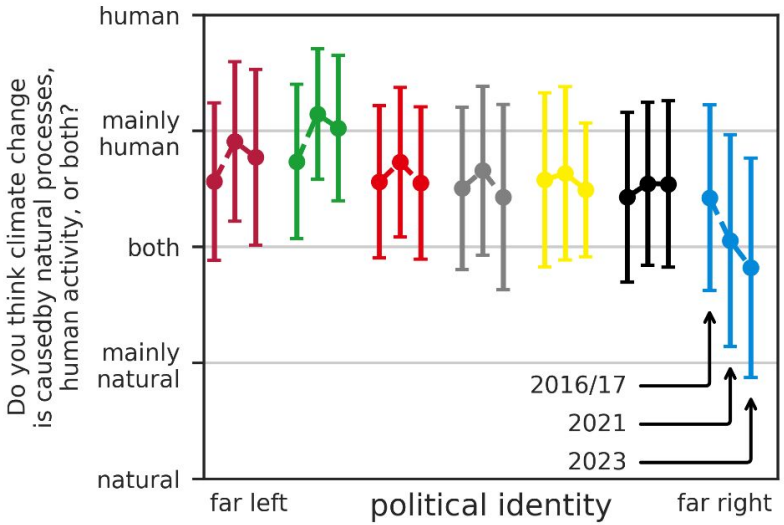
Flights

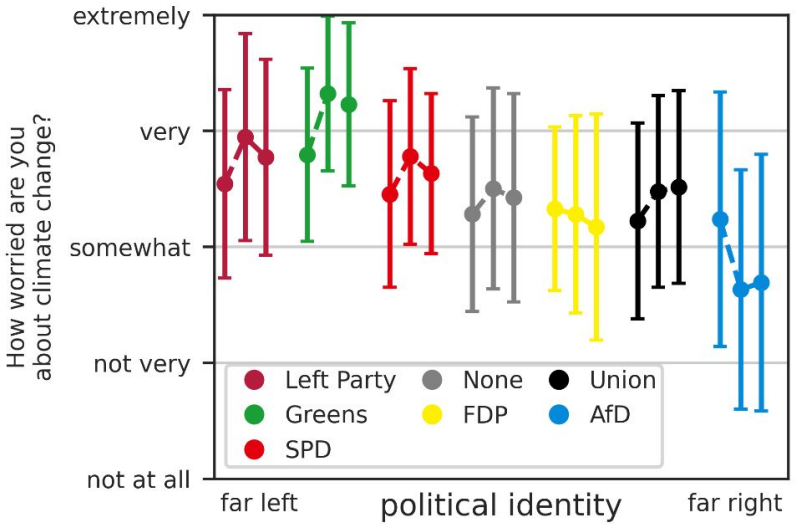
from nycflights13 import flights, weather, airports

Standard example dataset

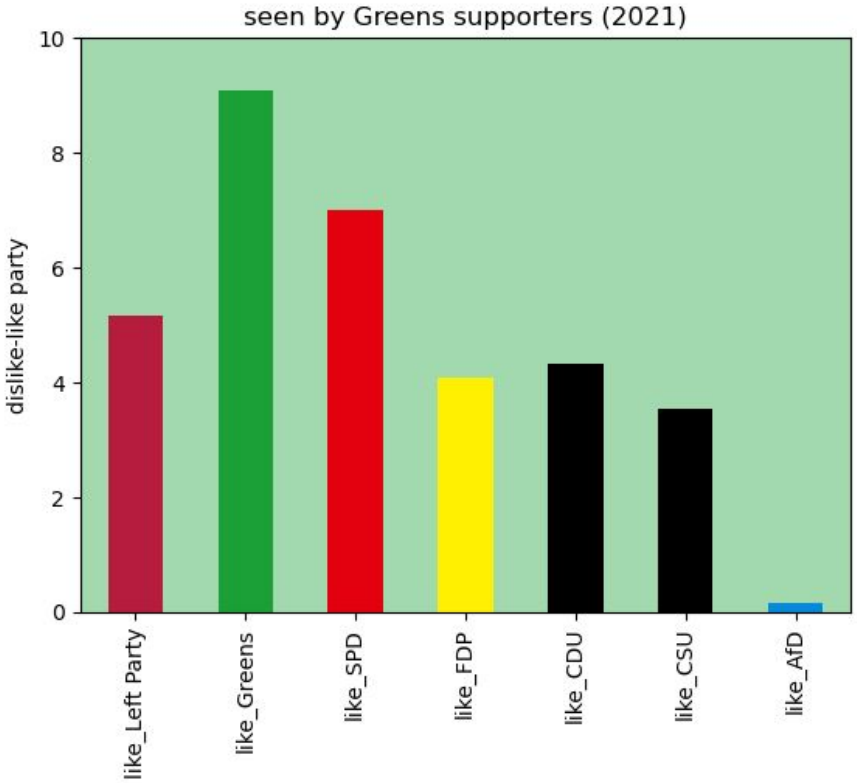


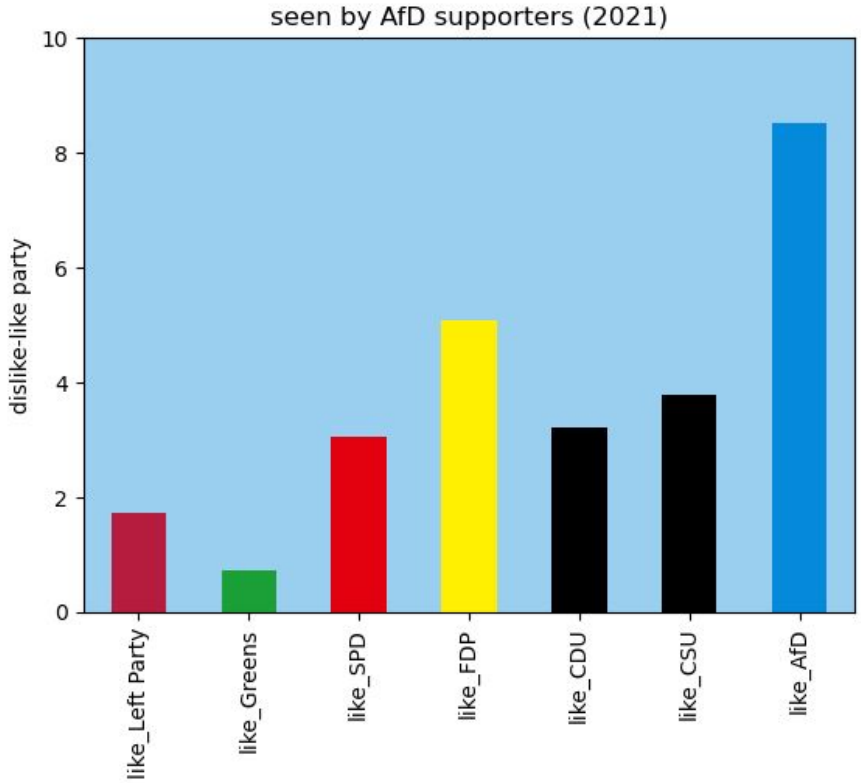
European Social Survey



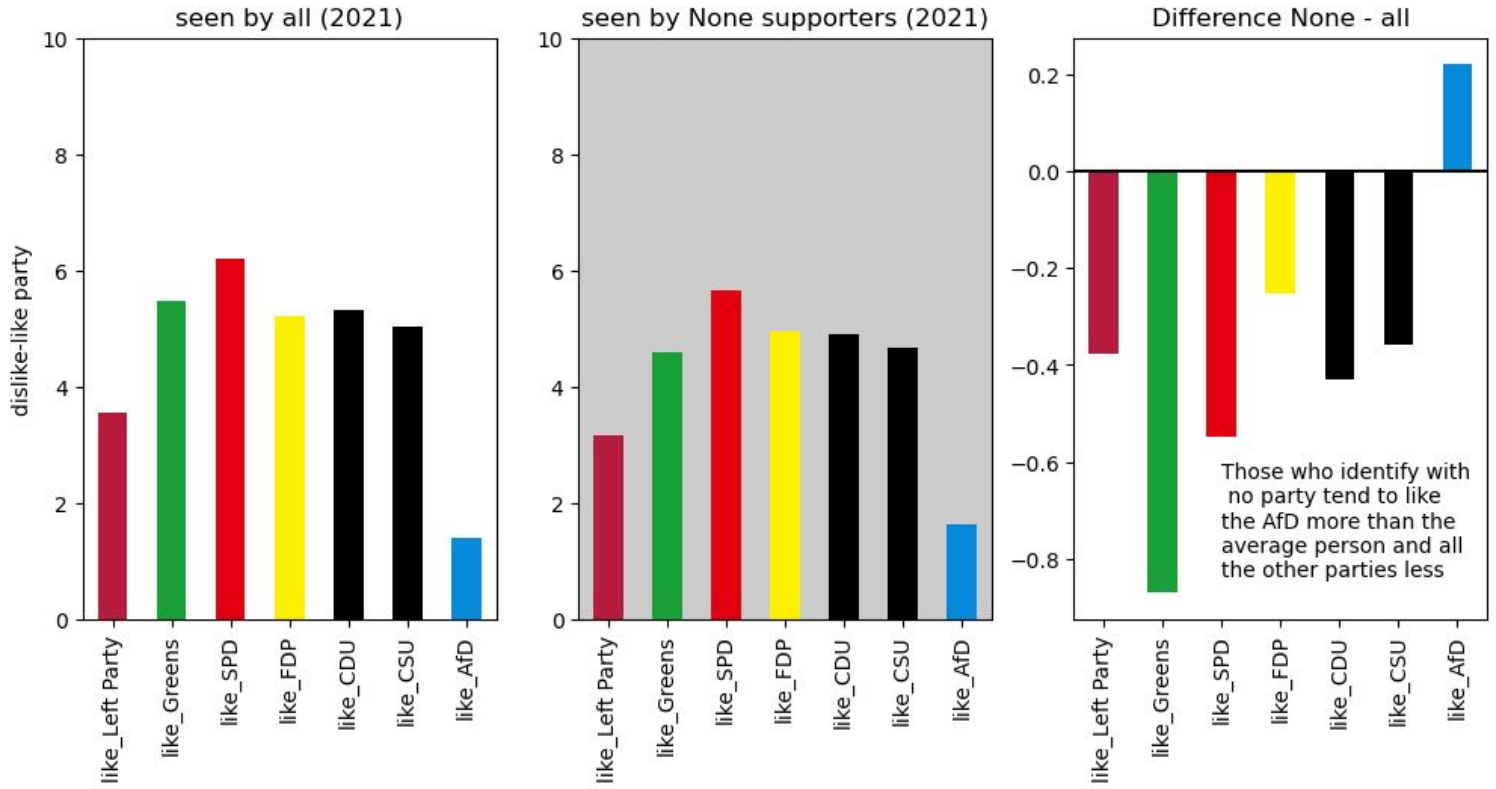


Comparative Study of Electoral Systems





Comparative Study of Electoral Systems



Computational Social Science (outlook)

The wake-up call: Reproducibility crisis in psychology and social sciences (2010s)

⟶ Transparent, open and FAIR data & code (Findable, Accessible, Interoperable, Retrievable).

Coding (at least pseudocoding) is essential to nearly any kind of science.
Mediocre code is absolutely OK (we are not software developers).
Bad code* (or, god forbid, no code**) is NOT OK in 2024 (we are not dinosaurs).

* code you do not understand yourself,
code that produces different things every time you run it,
code that involves any type of hard-coded results,
code that other people can not run or understand,
code that black-boxes things beyond good reason

** manipulated figures,
manually calculated statistics,
not publishing the code (on a publicly available platform)

Computational Social Science has transformed the social sciences and will continue to do so. Be a part of it!