Computational Sociology

Agent-based modeling and programming fundamentals

Dr. Thomas Davidson

Rutgers University

February 1, 2024

Plan

- Course updates
- Part I: Agent-based modeling
- Part II: Programming fundamentals

Course updates

Homework

- Homework 1 will be released by at the end of class
  - Due next Friday (2/9) at 5pm Eastern.

Introduction to agent-based modeling

Agent-based modeling and quantitative social science

- Most quantitative social science is variable-centered
  - e.g. We study the associations and interactions between variables in a linear regression

Introduction to agent-based modeling

Agent-based modeling and quantitative social science

- As a consequence, many sociologists think about the world in terms of what Andrew Abbott calls "general linear reality"
  - A social world composed of fixed entities with fixed attributes

Introduction to agent-based modeling

Agent-based modeling and quantitative social science

- Agent-based modeling is the study of "social life as interactions among adaptive agents who influence one another in response to the influence they receive." (Macy and Willer 2002)
  - Rather than interactions between *variables*, we consider interactions between *interdependent individuals*

Introduction to agent-based modeling

Agent-based modeling and quantitative social science

- Often we are interested in the *emergent* properties of local interactions between agents and how they aggregate into system-level processes such as diffusion, polarization, and segregation
  - These complex system-level patterns can emerge without any centralized coordination
- Like historical sociology and ethnography, agent-based modeling is a *relational* approach, focusing on the context-dependent and contingent nature of social interaction

Introduction to agent-based modeling

Key assumptions

- Macy and Willer 2002 outline four key assumptions that underpin many sociological agent-based models
  - Agents are *autonomous*
    - There is no system-wide coordination
  - Agents are *interdependent*
    - Agents respond to each other and to their environment
  - Agents follow *simple rules*
    - Simple local rules can generate global complexity
  - Agents are *adaptive* and *backwards looking*
    - Agents can alter their behavior through processes such as imitation and learning

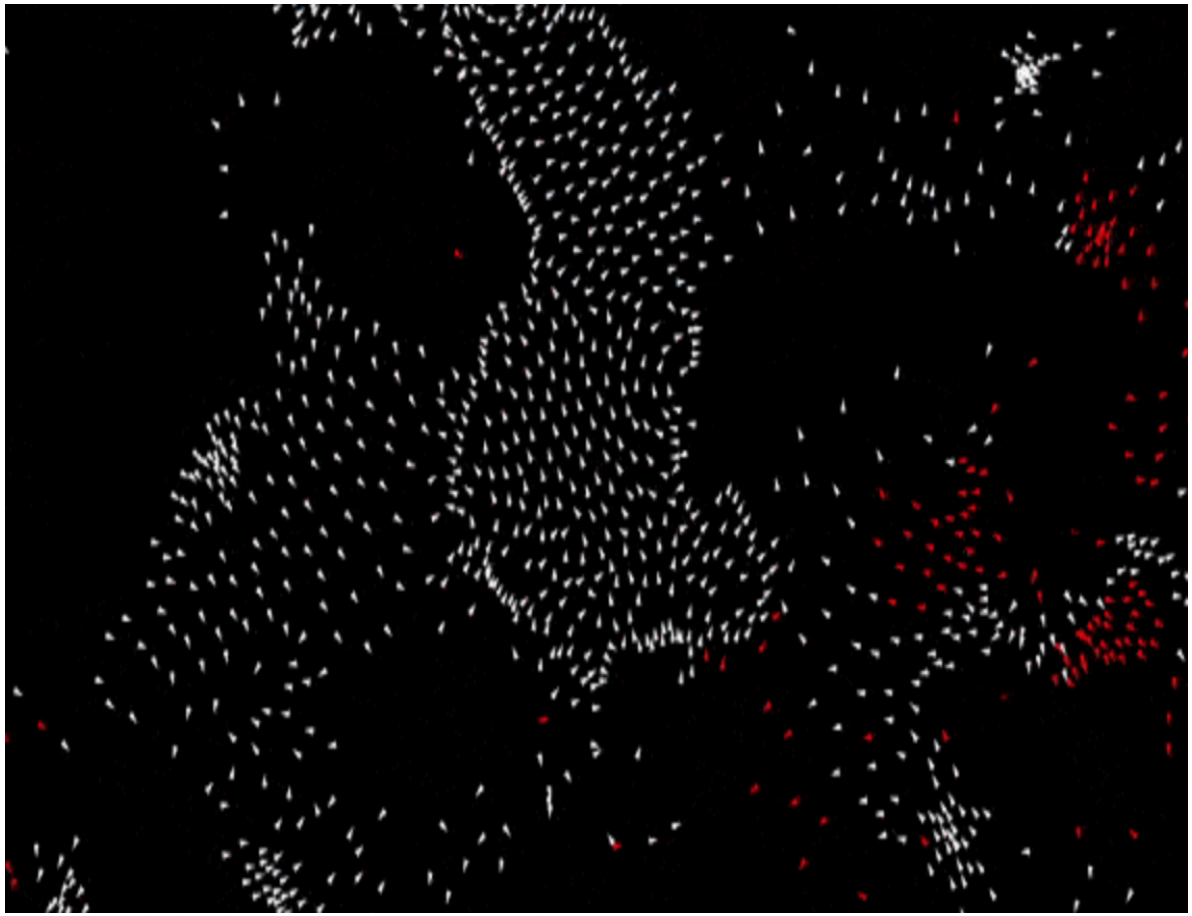
Introduction to agent-based modeling

Advantages of ABMs

- Virtual experiments to test causal mechanisms
  - Particularly useful where real-world experimentation is impractical
- Theory building and testing
  - Bridging between micro and macro levels of analysis
  - Varying the social structure *and* the agency of individuals

Introduction to agent-based modeling

Craig Reynolds *Flocking behavior* (1987)



Reynolds, Craig W. 1987. "Flocks, Herds and Schools: A Distributed Behavioral Model." In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, 25–34.

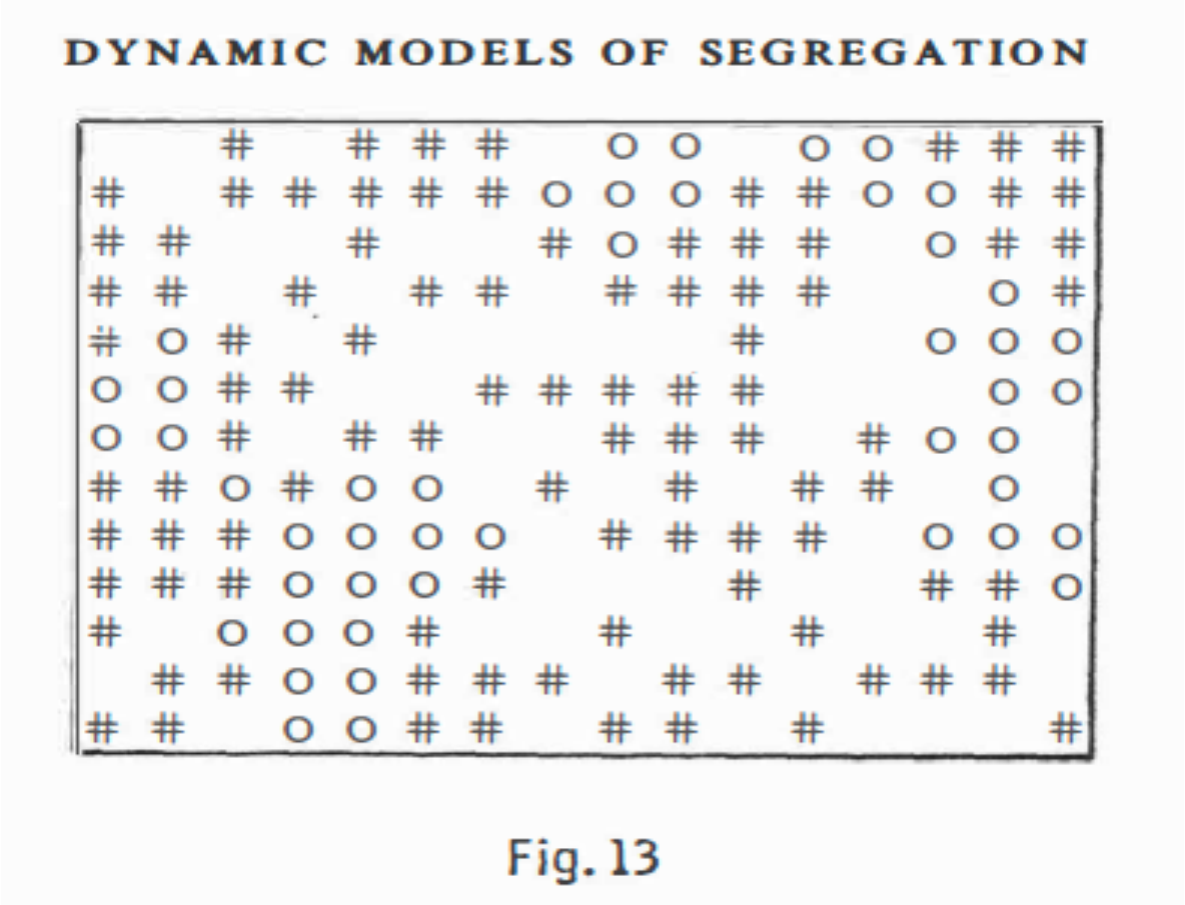
Introduction to agent-based modeling

Flocking behavior in NetLogo

http://www.netlogoweb.org/launch#http://ccl.northwestern.edu/netlogo/models/models/Sample%20Models/Biology/Flocking.nlogo

Introduction to agent-based modeling

Thomas Schelling *Homophily and segregation*



Schelling, Thomas C. 1971. "Dynamic Models of Segregation." *Journal of Mathematical Sociology* 1: 143–86.

NetLogo and NetLogoWeb

Schelling's segregation model in NetLogo

http://www.netlogoweb.org/launch#http://ccl.northwestern.edu/netlogo/models/models/IABM%20Textbook/chapter%203/Segregation%20Extensions/Segregation%20Simple.nlogo

Modeling diffusion

Simple diffusion and the S-curve

Modeling diffusion

Complex contagions

- A simple diffusion process, like catching the flu, requires a single exposure
- *Complex contagions* require exposures to multiple people
  - Joining a high-risk social movement
  - Avant garde fashions
  - New technologies

Modeling diffusion

Thresholds

- *Thresholds* denote the number of exposures
  - Variation across diffusion processes
  - Individual variation
    - Distributions of thresholds in a population*

*See: Granovetter, Mark. 1978. "Threshold Models of Collective Behavior." *American Journal of Sociology* 83(6):1420–43.

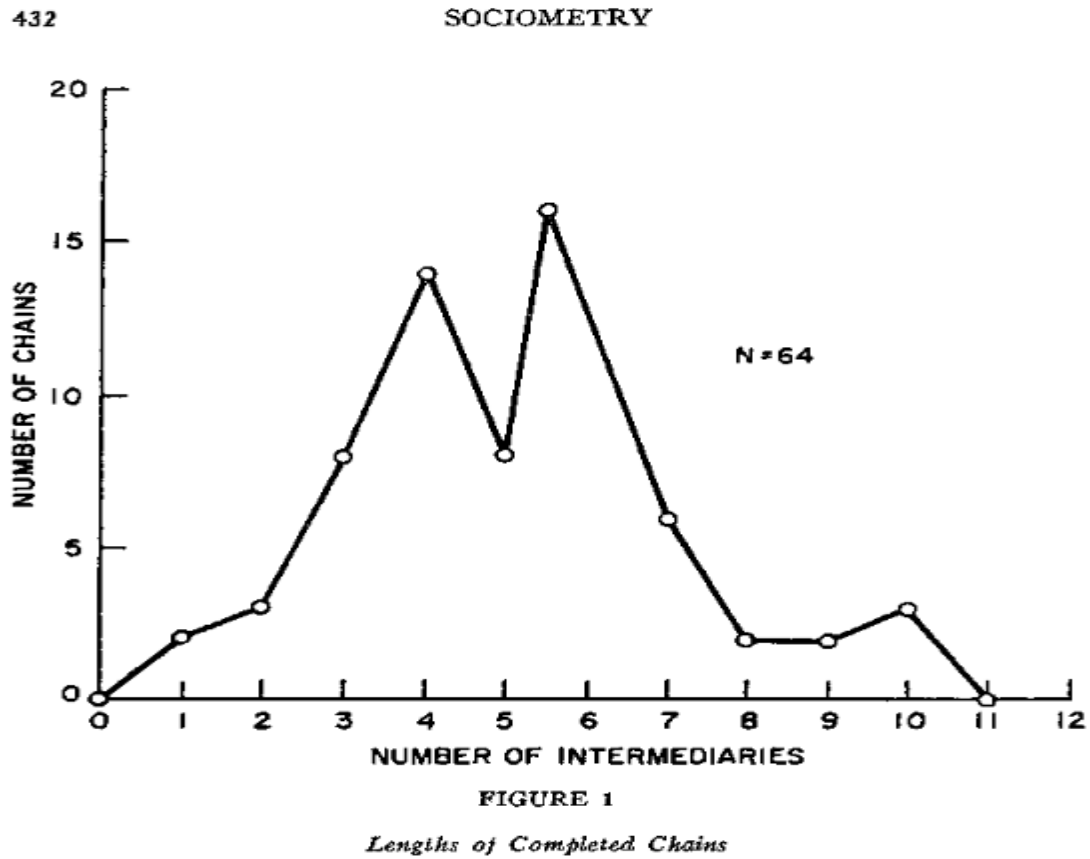
Modeling diffusion

Weak ties as long ties

- Granovetter (1971) emphasized the "*strength* of weak ties"
- Centola and Macy (2007) put emphasis on the "*length*" of weak ties, noting how the can help to connect different parts of a network

Modeling diffusion

Six degrees of separation and small-worlds

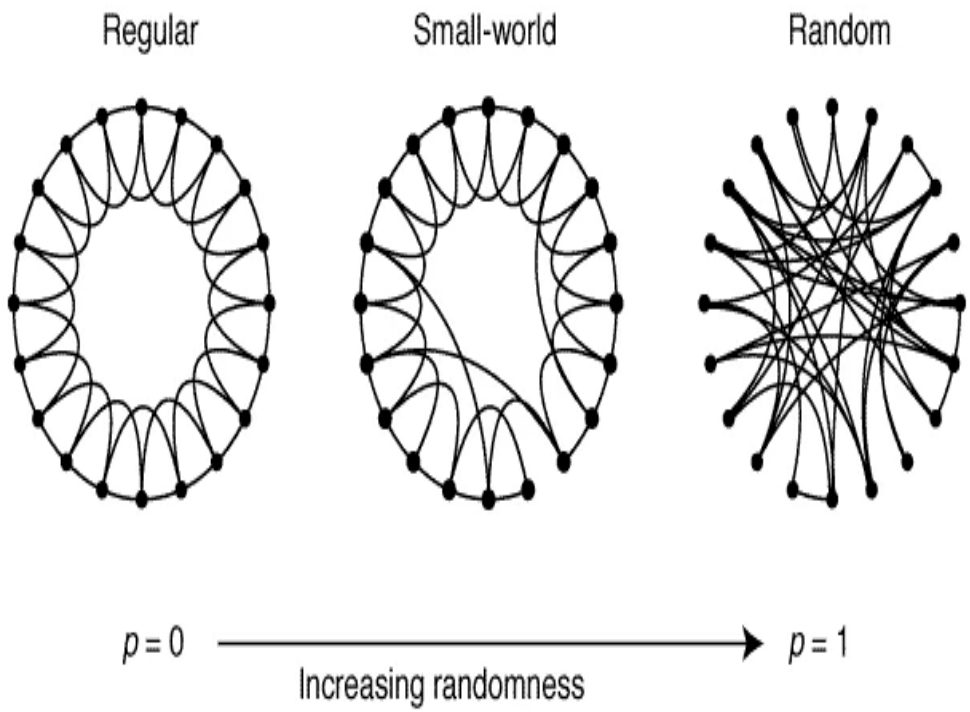


Travers, Jeffrey, and Stanley Milgram. 1969. "An Experimental Study of the Small World Problem." *Sociometry* 32(4):425.

Modeling diffusion

Six degrees of separation and small-worlds

Figure 1: Random rewiring procedure for interpolating between a regular ring lattice and a random network, without altering the number of vertices or edges in the graph.



Watts, Duncan J., and Steven H. Strogatz. 1998. "Collective Dynamics of 'Small-World' Networks." *Nature* 393(6684):440–42.

Modeling diffusion

Six degrees of separation and small-worlds

Table 1 Empirical examples of small-world networks

	L_{actual}	L_{random}	C_{actual}	C_{random}
Film actors	3.65	2.99	0.79	0.00027
Power grid	18.7	12.4	0.080	0.005
C. elegans	2.65	2.25	0.28	0.05

Characteristic path length L and clustering coefficient C for three real networks, compared to random graphs with the same number of vertices (n) and average number of edges per vertex (k). (Actors: $n = 225{,}226$, $k = 61$. Power grid: $n = 4{,}941$, $k = 2.67$. *C. elegans*: $n = 282$, $k = 14$.) The graphs are defined as follows. Two actors are joined by an edge if they have acted in a film together. We restrict attention to the giant connected component[16] of this graph, which includes ~90% of all actors listed in the Internet Movie Database (available at http://us.imdb.com), as of April 1997. For the power grid, vertices represent generators, transformers and substations, and edges represent high-voltage transmission lines between them. For *C. elegans*, an edge joins two neurons if they are connected by either a synapse or a gap junction. We treat all edges as undirected and unweighted, and all vertices as identical, recognizing that these are crude approximations. All three networks show the small-world phenomenon: $L \gtrsim L_{random}$ but $C \gg C_{random}$.

Watts, Duncan J., and Steven H. Strogatz. 1998. "Collective Dynamics of 'Small-World' Networks." *Nature* 393(6684):440–42.

Modeling diffusion

Complex contagions

- While much information flows through weak ties, Centola and Macy (2007) argue that weak ties are often insufficient for social contagions
- It is not just the presence of ties that "bridge" communities, but the *width* of the bridge, or the number of ties, that matters

Modeling diffusion

Modeling complex contagions

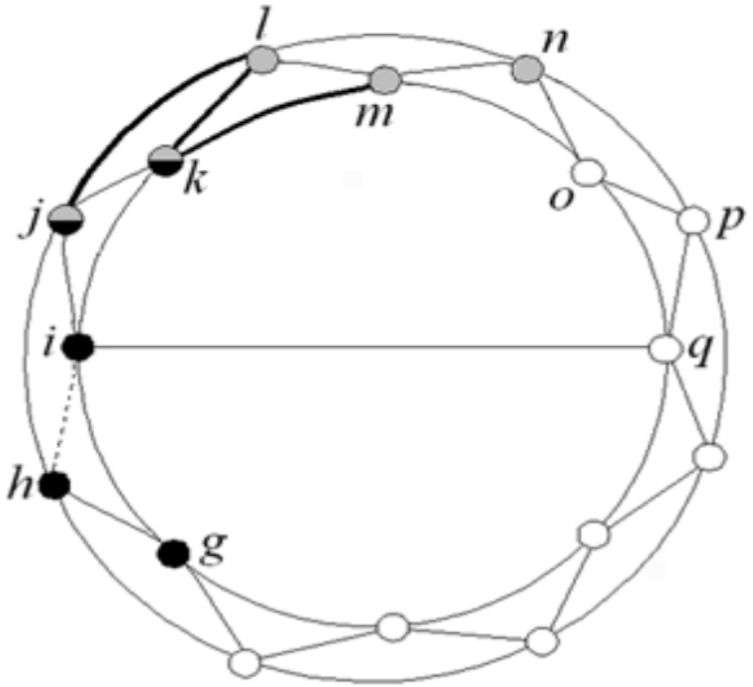


FIG. 1.—A ring lattice with $z = 4$ and one long tie. The figure illustrates the width of the bridge between the neighborhoods of i (black and gray/black nodes) and l (gray and gray/black nodes), showing the two common members (gray/black nodes). The bridge between these two neighborhoods consists of the three ties jl, kl, and km (shown as bold lines). The long tie from i to q provides a shortcut for a simple contagion but not for one that is complex.

Modeling diffusion

Modeling complex contagions

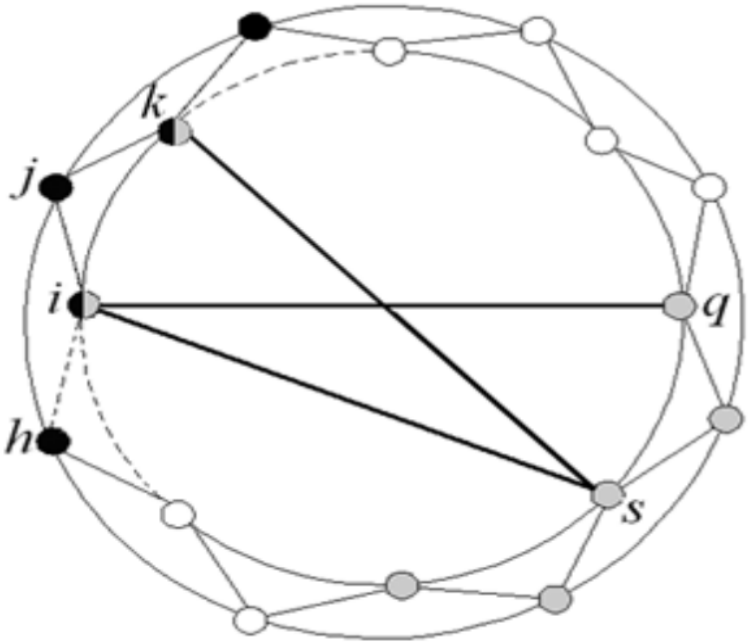
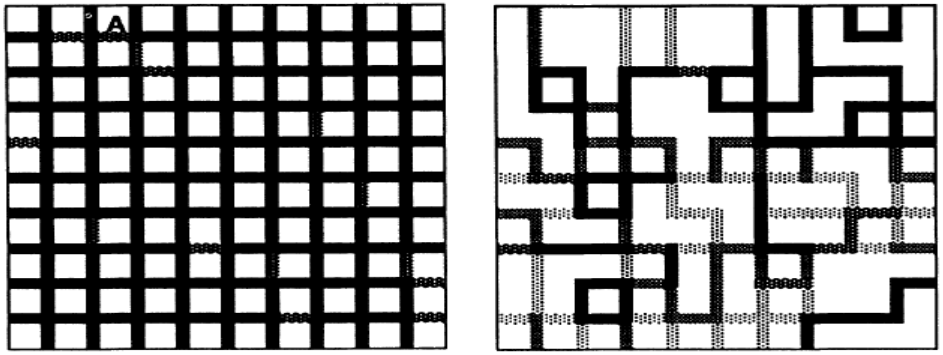


FIG. 2.—A ring lattice with $z = 4$ and three ties. The figure shows the width of the bridge between neighborhood J (black and gray/black nodes, with focal node j) and neighborhood S (gray and gray/black nodes, with focal node s), showing the two common members (gray/black nodes). An increment in the threshold from $\tau = 1/z$ to $\tau = 2/z$ triples the width of the bridge required to create a shortcut (bold lines) between J and S, from one tie to three. The two ties is and ks are sufficient to activate s, and the third tie from i to q is sufficient to activate q, given the tie from s to q.

Modeling polarization

Robert Axelrod *Local convergence and global polarization* (1987)



(a) At start

(b) After 20,000 events

(c) After 40,000 events

(d) After 80,000 events

Axelrod, Robert. 1997. "The Dissemination of Culture: A Model with Local Convergence and Global Polarization." *Journal of Conflict Resolution* 41 (2): 203–26.

Modeling polarization

Delia Baldassarri and Peter Bearman (2007)

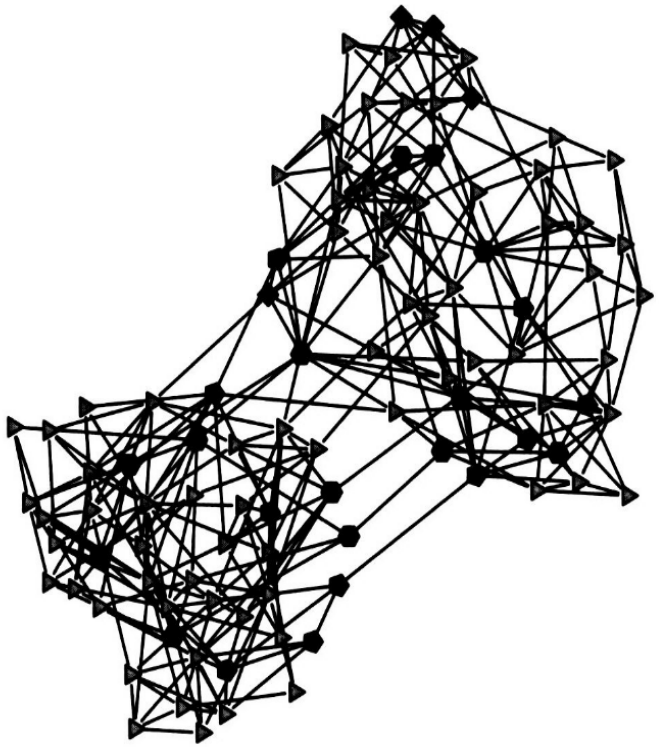


Figure 7. Discussion Network at Time 500 in a Takeoff Case (#963)

Notes: Nodes represent the issue that has been discussed most frequently by each actor. The color distinguishes between the most popular issue (grey) and all the other issues (black). Simulation #963, takeoff.

Baldassarri, Delia, and Peter Bearman. 2007. "Dynamics of Political Polarization." *American Sociological Review* 72(5):784–811.

Modeling polarization

Politics and lifestyle choices

DellaPosta, Shi, and Macy (2015) suggest a mechanism to explain observed correlations between political attitudes and lifestyle choices

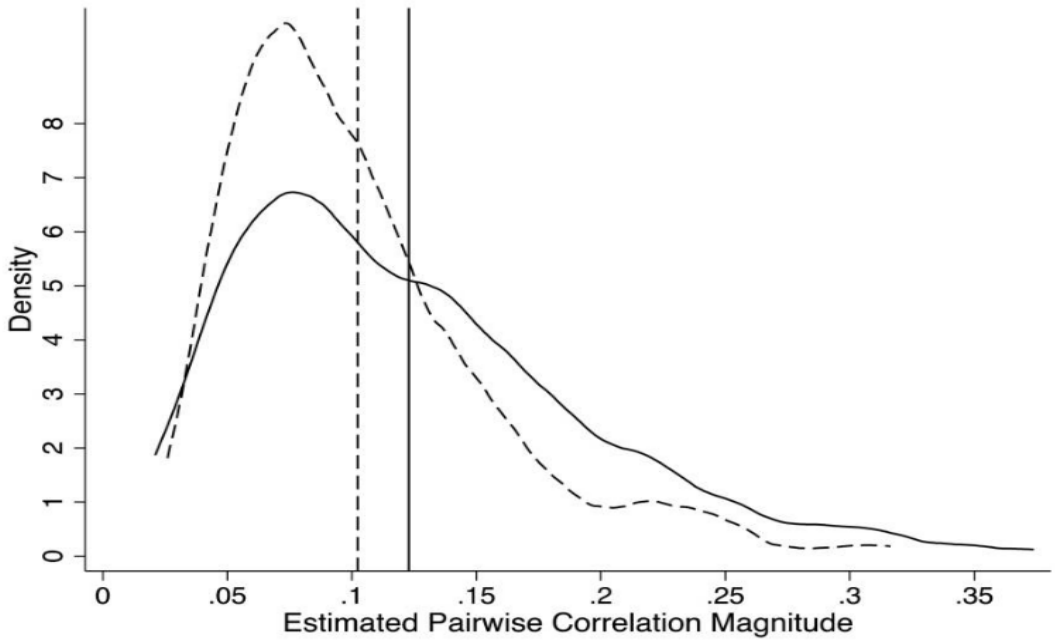


FIG. 2.—Magnitude of zero-order and partial correlation between GSS lifestyle items and ideological identity. Graphs plot Epanechnikov kernel density functions for both zero-order (solid lines) and partial (dashed lines) correlation magnitudes estimated from the mixed-effects model (see table A2 in the appendix). One value is plotted for each of the 216 item pairs. Time is set to 2010 to facilitate comparison across item pairs. The solid vertical reference line gives the mean predicted zero-order correlation magnitude across all item pairs in 2010 and the dashed vertical reference line gives the same value for partial correlation magnitude.

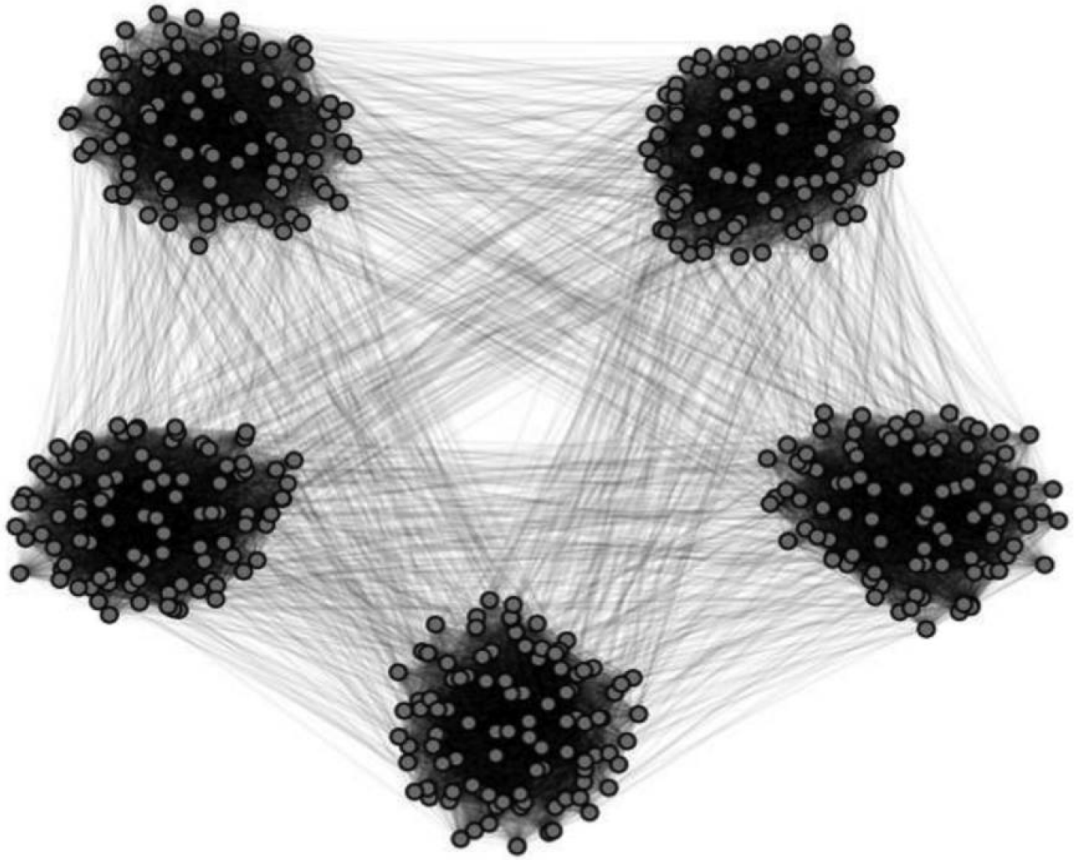
Modeling polarization

Politics and lifestyle choices

- Dellaposta and colleagues argue that correlation between politics and lifestyle choices explained by *network autocorrelation*, "the tendency for people to resemble their network neighbors" (p.1488)
- Feedback loops between spatial settings, relations, and lifestyle

Modeling polarization

Politics and lifestyle choices



Modeling polarization

Competing explanations: Cognitive mechanisms

Goldberg and Stein (2018) propose an alternative mechanism, arguing that culture does not spread like a virus, but depends on belief structures

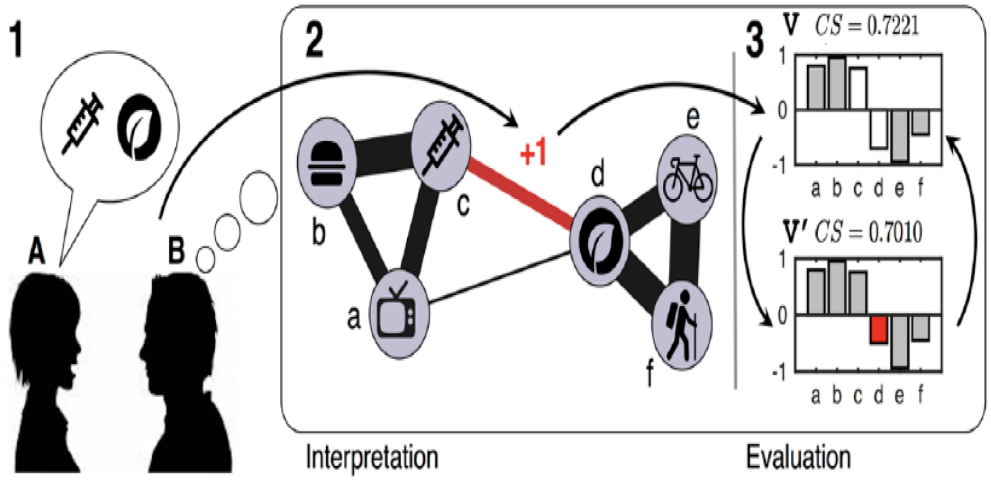


Figure 3. An Illustration of the Agent-Based Model Sequence

Note: (1) Agent *B* observes *A* express support for vaccinations and organic food (practices *c* and *d*); (2) *B* updates the corresponding element in his associative matrix, *R* (the edge connecting nodes *c* and *d* in the network representation of *R*); and (3) randomly updates his preference for organic food (practice *d*, resulting in preference vector *V*′), which is the weaker preference of the pair {c,d} in his preference vector *V*. Because constraint satisfaction is reduced from .7221 to .7010, this preference update is rejected, and *B*'s preference vector *V* remains unchanged.

Introduction to agent-based modeling

Integrating real-world data

DiMaggio and Garip (2011) construct agent with attributes based on the General Social Survey

TABLE 2
LINEAR REGRESSION OF ADOPTION LEVELS ON EXPERIMENTAL CONDITIONS

		RACE		INCOME		EDUCATION	
	ALL	Whites	Blacks	High	Low	BA	Less than High School
No network externalities	−.516**	−.536**	−.399**	−.685**	−.238**	−.611**	−.351**
General network externalities	.030**	.028**	.043**	.032**	.017**	.023**	.030**
Homophily = .25	−.003**	−.001	−.012**	.009**	−.014**	.005**	−.011**
Homophily = .5	−.005**	−.002**	−.024**	.017**	−.028**	.010**	−.024**
Homophily = .75	−.011**	−.006**	−.040**	.024**	−.046**	.012**	−.043**
Homophily = 1	−.019**	−.012**	−.061**	.029**	−.067**	.015**	−.068**
Intercept	.618**	.647**	.454**	.925**	.249**	.788**	.392**
R^2	.99	.99	.97	.99	.96	.99	.96

NOTE.—All independent variables are binary. Both dependent and independent variables are measured on the final period of simulations ($t = 100$). Reference: homophily = 0; $N = 7{,}000$.

* $P < .05$.

** $P < .01$.

Introduction to agent-based modeling

Realism and external validity

- Bruch and Atwell (2015) distinguish between two types of realism in ABMs
  - *Low-dimensional realism*: simple, parsimonious models
  - *High-dimensional realism*: complex, complicated models
- Trade-offs:
  - The latter might be more realistic, but involve more parameters and may be less intelligible

Introduction to agent-based modeling

Parameters and sensitivity

- Use theory to guide decisions regarding which parameters vary and should be fixed
  - Formalizations can be hard to operationalize
- Models can be extremely sensitive to small variations in parameters
  - Be careful to check for coding errors!*
- Timing matters
  - Constant time vs. discrete-time
  - Asynchronous vs. synchronous updating

* Read these papers and associated exchanges for some cautionary tales: van de Rijt, Arnout, David Siegel, and Michael Macy. 2009. "Neighborhood Chance and Neighborhood Change: A Comment on Bruch and Mare." *American Journal of Sociology* 114(4):1166–80 & Goldberg, Amir. 2021. "Reply to DellaPosta and Davoodi: Associative Diffusion and the Pitfalls of Structural Reductionism." *American Sociological Review* 1–6.

Recap

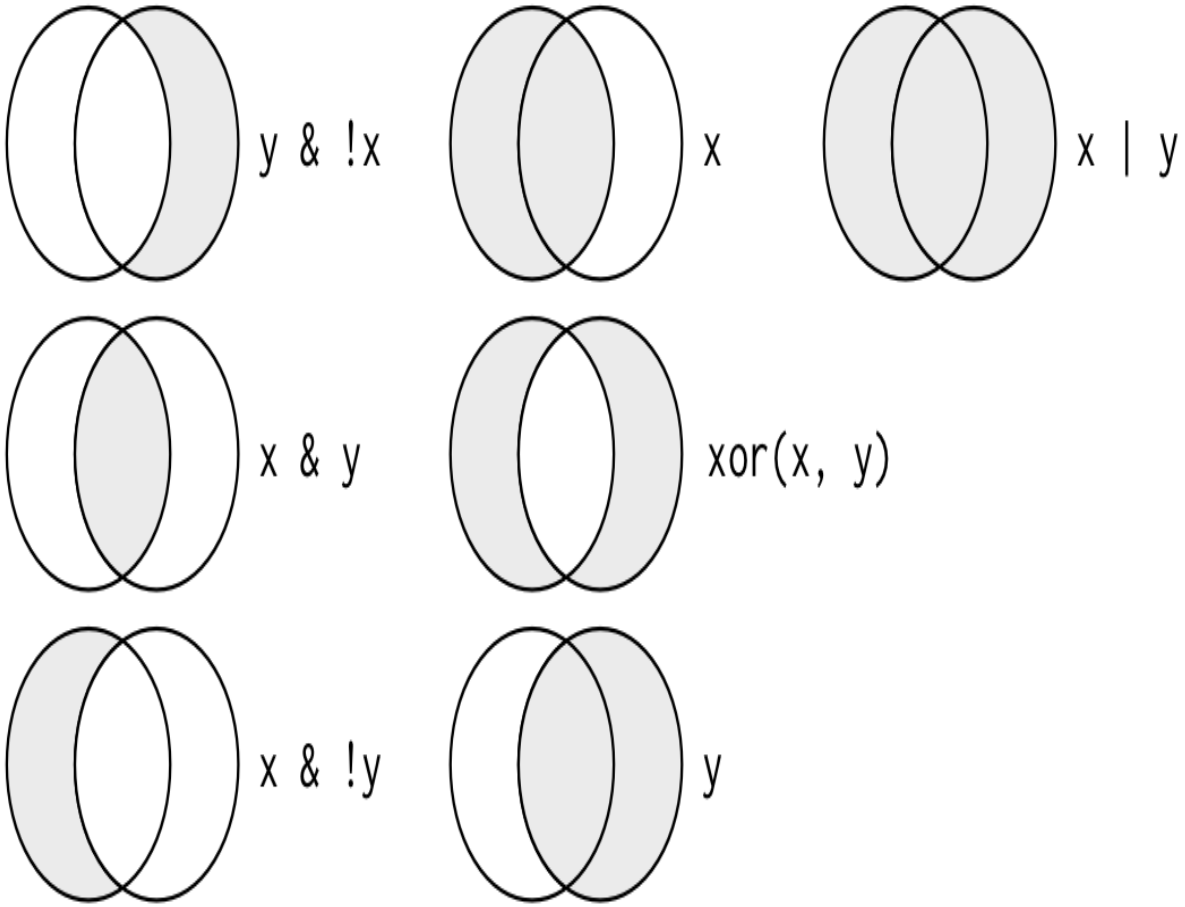
Data structures in R

- Basic data types
- Vectors
- Lists
- Matrices

Programming fundamentals

- Boolean logic
- If-else statements
- Loops
- Functions
- Pipes

Boolean logic in R



Boolean logic

```
TRUE == TRUE  # equals
```

```
## [1] TRUE
```

```
TRUE != FALSE  # not equals
```

```
## [1] TRUE
```

```
TRUE == !FALSE
```

```
## [1] TRUE
```

```
!TRUE != FALSE
```

```
## [1] FALSE
```

Boolean logic

```
TRUE | FALSE  # or
```

```
## [1] TRUE
```

```
TRUE & FALSE  # and
```

```
## [1] FALSE
```

```
TRUE & FALSE == FALSE
```

```
## [1] TRUE
```

Boolean logic

```
TRUE | FALSE & FALSE
```

```
## [1] TRUE
```

```
FALSE | TRUE & FALSE
```

```
## [1] FALSE
```

See the documentation for more on logic in R.

If-else statements

- We often encounter situations where we want to make a choice contingent upon the value of some information received.
- If-else statements allow us to chain together one or more conditional actions.
  - e.g. If time is between 10:20-11:40am AND day is Monday or Thursday, attend Computational Data Science lecture. Else, do something else.

If-else statements

The basic syntax. The `if` is followed by a conditional statement in parentheses. If the condition is met, then the code in the braces is executed.

```
x <- TRUE

if (x == TRUE) {
    print("x is true")
}
```

```
## [1] "x is true"
```

If-else statements

In this case we have a vector containing five fruits. We use `sample` to randomly pick one. We can use an if-statement to determine whether we have selected an apple. Complete the conditional.

```
fruits <- c("apple", "apple",
            "orange", "orange",
            "apple")

f <- sample(fruits, 1)

if () {print("We selected an apple")}
```

If-else statements

In the previous example we only have an if-statement. If the condition is not met then nothing happens. Here we add an else statement.

```
fruits <- c("apple", "apple", "orange", "orange", "apple")

f <- sample(fruits, 1)

if (f == "apple") {
    print("We selected an apple")
} else {
    print("We selected an orange")
}
```

```
## [1] "We selected an orange"
```

Note that R can be quite fussy about the syntax. If `else` is on the line below then the function throws an error.

If-else statements

What about this case where we have another fruit? If we only care about apples we could modify the output of our else condition.

```
fruits <- c("apple", "apple", "orange", "orange", "apple", "pineapple")

f <- sample(fruits, 1)

if (f == "apple") {
    print("We selected an apple")
} else {
    print("We selected another fruit.")
}
```

```
## [1] "We selected another fruit."
```

If-else statements

We could also use else-if statements to have a separate consideration of all three.

```
f <- sample(fruits, 1)

if (f == "apple") {
    print("We selected an apple")
} else if (f == "orange") {
    print("We selected an orange")
} else {
    print("We selected a pineapple")
}
```

```
## [1] "We selected an apple"
```

Loops

- Often when we program we need to complete the same operation many times. One of the common approaches is to use a loop.
- There are two kinds of loops you will encounter
  - For-loops
    - Iterative over an entire sequence
  - While-loops
    - Iterate over a sequence while a condition is met

For-loops

Here is a simple example where we use a loop to calculate the sum of a sequence of values.

```
s <- 0  # value to store our sum

for (i in 1:100) {
    # for i from 1 to 100
    s <- s + i  # add i to sum
}

print(s)
```

```
## [1] 5050
```

For-loops

```
s = 0

for i in range(1,101):
  s += i

print(s)
```

```
## 5050
```

The syntax varies slightly across programming languages but the basic structure is very similar, as this Python example shows. Note that for-loops and other functions in R tend to use braces around the operations. We will see this again when we look at functions.

For-loops

Write a loop to print each number.

```
nums <- 1:10
```

For-loops

```
is_orange <- logical(length(fruits))  # a vector of logical objects

i = 1  # In this case we need to maintain a counter
for (f in fruits) {
    if (f == "orange") {
        is_orange[i] <- TRUE
    }
    i <- i + 1  # increment counter by 1
}

print(fruits)
```

```
## [1] "apple"     "apple"     "orange"    "orange"    "apple"     "pin
```

```
print(is_orange)
```

```
## [1] FALSE FALSE  TRUE  TRUE FALSE FALSE
```

For-loops

Loops can also easily be *nested*. Here we start a second loop within the first one and use it to populate a matrix.

```
M <- matrix(nrow = 5, ncol = 5)

for (i in 1:5) {
    for (j in 1:5) {
        M[i, j] <- i * j
    }
}
print(M)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    2    4    6    8   10
## [3,]    3    6    9   12   15
## [4,]    4    8   12   16   20
## [5,]    5   10   15   20   25
```

While-loops

A while loop runs when a condition is true and ends when it becomes false. *Make sure the condition will eventually be false to avoid an infinite loop.*

```
i <- 1  # iterator
while (i < 5) {
    print(i)
    i <- i + 1
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
```

While-loops

In this example we iterate over fruits until we get a pineapple.

```
i <- 1  # iterator
f <- fruits[i]  # define initial value
while (f != "pineapple") {
    print(f)
    i <- i + 1  # increment index
    f <- fruits[i]  # get next f
}
```

```
## [1] "apple"
## [1] "apple"
## [1] "orange"
## [1] "orange"
## [1] "apple"
```

Functions

- A function is a customized sequence of operations
- We use functions to make our code modular and extendable
- There are thousands of functions built into R and available for packages, but sometimes it is useful to create our own
  - *R4DS* contains the following heuristic:
    - "You should consider writing a function whenever you've copied and pasted a block of code more than twice"

Functions

Here is an example of a simple function that returns the mean of a vector of values x.

```
avg <- function(x) {
    return(sum(x)/length(x))
}

avg(c(5, 6, 6, 4, 3))
```

```
## [1] 4.8
```

We define the function by using the `function` command and assigning it to the name `avg`. The content in the parentheses is called the `argument` of the function. The `return` statement tells the function what output to produce.

Functions

Here is the same function in Python.

```
def avg(x):
  return(sum(x)/len(x))

avg([5,6,6,4,3])
```

```
## 4.8
```

Again you can see that the syntax is slightly different, for example the `def` command is used to define a function on the left hand side, followed by the name.

Functions

Testing

- It is important to test functions to ensure they work as expected
  - Ensure the function will only process valid inputs
  - It is good practice to handle incorrect inputs
    - There are many ways a function could behave that would not raise errors in R but could still be problematic
  - Write unit tests to ensure the function works as expected
    - Make sure to handle edge cases, inputs that require special handling

Functions

Testing

```
avg(c("a", "b", "c"))
```

```
## Error in sum(x): invalid 'type' (character) of argument
```

Functions

Testing

```
avg(c())
```

```
## [1] NaN
```

Functions

Testing

The function can be modified to return a message if input is incorrect. Note the use of two `return` statements within the function.

```
avg <- function(values) {
    if (!is.numeric(values)) {
        return("Input must be numeric.")
    } else {
        return(sum(values)/length(values))
    }
}
```

Functions

Testing

```
# Unit tests
avg(c("a", "b", "c"))
```

```
## [1] "Input must be numeric."
```

```
avg(c())
```

```
## [1] "Input must be numeric."
```

```
avg(c(2.6, 2.4))
```

```
## [1] 2.5
```

Pipes

- Pipes are a tool designed to allow you to chain together a sequence of operations
- The pipe is designed to improve the readability of complex chains of function
- Implemented in the `magrittr` package but loaded in `tidyverse`

Pipes

Pipes can be used to chain together sequences of operations. There are two different versions of the syntax:

```
library(tidyverse)
x <- 10

x %>%
    print()  # Old style
```

```
## [1] 10
```

```
x |>
    print()  # New style
```

```
## [1] 10
```

Pipes

Note how pipes allow us to chain operations from left to right, rather than nesting them from inner to outer. In this case we take a sequence from 1 to 10, get the square root of each value, sum the roots, then print the sum.

```
print(sum(sqrt(seq(1:10))))  # nested functions
```

```
## [1] 22.46828
```

```
seq(1:10) %>%
    sqrt() %>%
    sum() %>%
    print()  # using pipes
```

```
## [1] 22.46828
```

Pipes

We can also use pipes to do basic arithmetic using pipes. Note again the difference between the nested operations and the pipe operator.

```
library(magrittr)
((1 + 2) - 10) * 10
```

```
## [1] -70
```

```
1 %>%
    add(2) %>%
    subtract(10) %>%
    multiply_by(10)
```

```
## [1] -70
```

Note how `magrittr` provides aliases for certain mathematical operations as shown in the second line. This StackOverflow post has some further discussion.

Pipes

Pipes are particularly useful we're working with tabular data. Here's an example without pipes or nesting. Each line produces an object that is then passed as input to the following line.

```
library(nycflights13)



not_delayed <- filter(flights, !is.na(dep_delay), !is.na(arr_delay))
grouped <- group_by(not_delayed, year, month, day)
summary <- summarize(grouped, mean = mean(dep_delay))
print(summary)
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [12]
##     year month   day  mean
##    <int> <int> <int> <dbl>
##  1  2013     1     1 11.4
##  2  2013     1     2 13.7
##  3  2013     1     3 10.9
##  4  2013     1     4  8.97
```

Pipes

In this case the expressions have been nested. This is better as we are not unnecessarily storing intermediate objects.

```
summarize(group_by(
  filter(flights, !is.na(dep_delay), !is.na(arr_delay)),
  year, month, day), mean = mean(dep_delay))
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [12]
##     year month   day  mean
##    <int> <int> <int> <dbl>
##  1  2013     1     1 11.4
##  2  2013     1     2 13.7
##  3  2013     1     3 10.9
##  4  2013     1     4  8.97
##  5  2013     1     5  5.73
##  6  2013     1     6  7.15
##  7  2013     1     7  5.42
##  8  2013     1     8  2.56
##  9  2013     1     9  2.30
```

Pipes

Write out this expression using a pipe

Putting it all together: Schelling's segregation model in R

- Open the file `schelling.R`, located inside the `code` directory

Homework

- Visit the link on Slack for link to Github Classroom
- Clone to your computer (see instructions on website)
- Homework due 2/9 (next Friday) at 5pm
- Submit via Github (see instructions on website)

Next week

- Application Programming Interfaces for data collection
  - *Sign up for a Spotify account*
- Tabular data and visualization