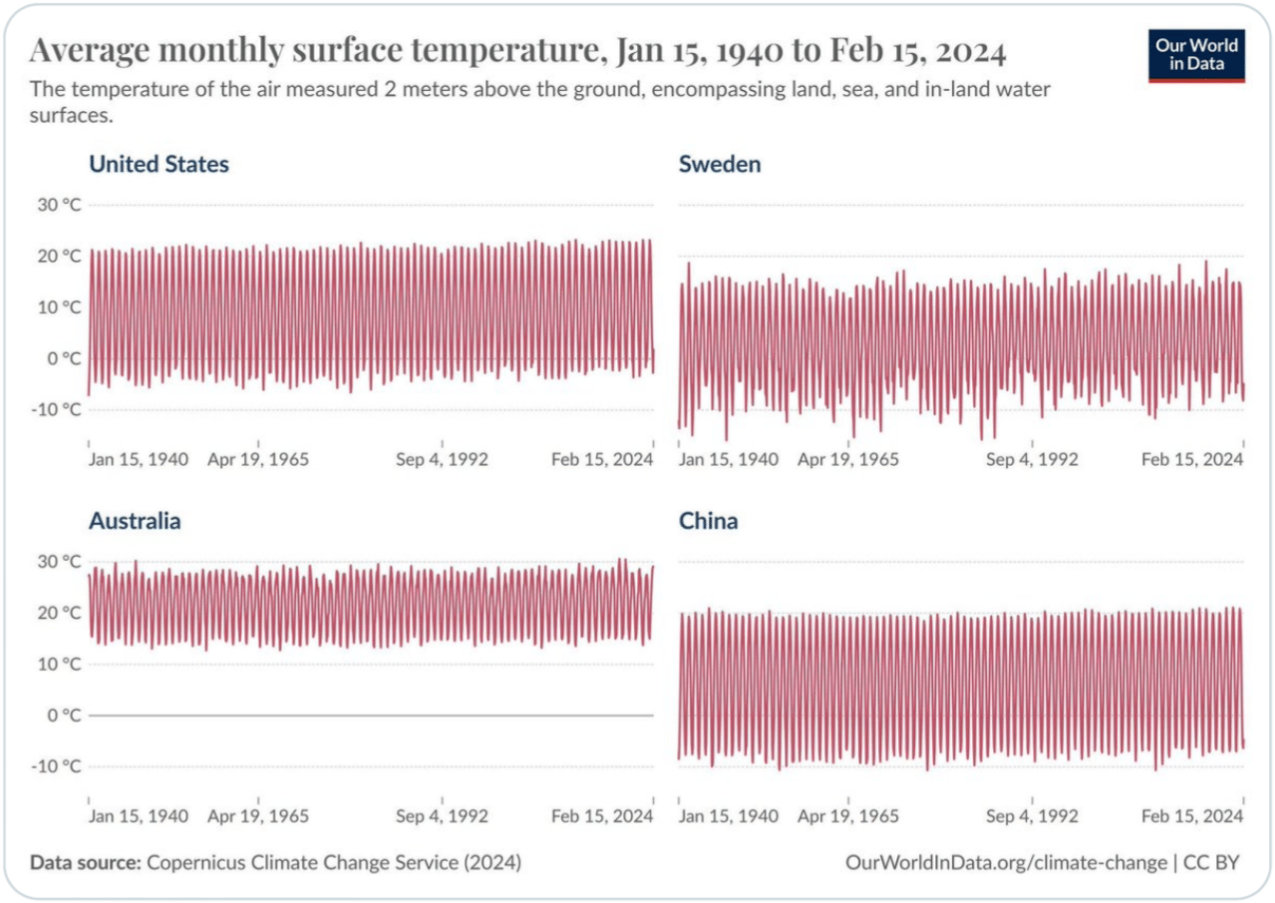
11 — Text as Data

Kieran Healy

March 29, 2024

Pedro Domingos ✓
@pmddomingos

Average temperatures by month instead of year, and suddenly the climate crisis looks like a joke.

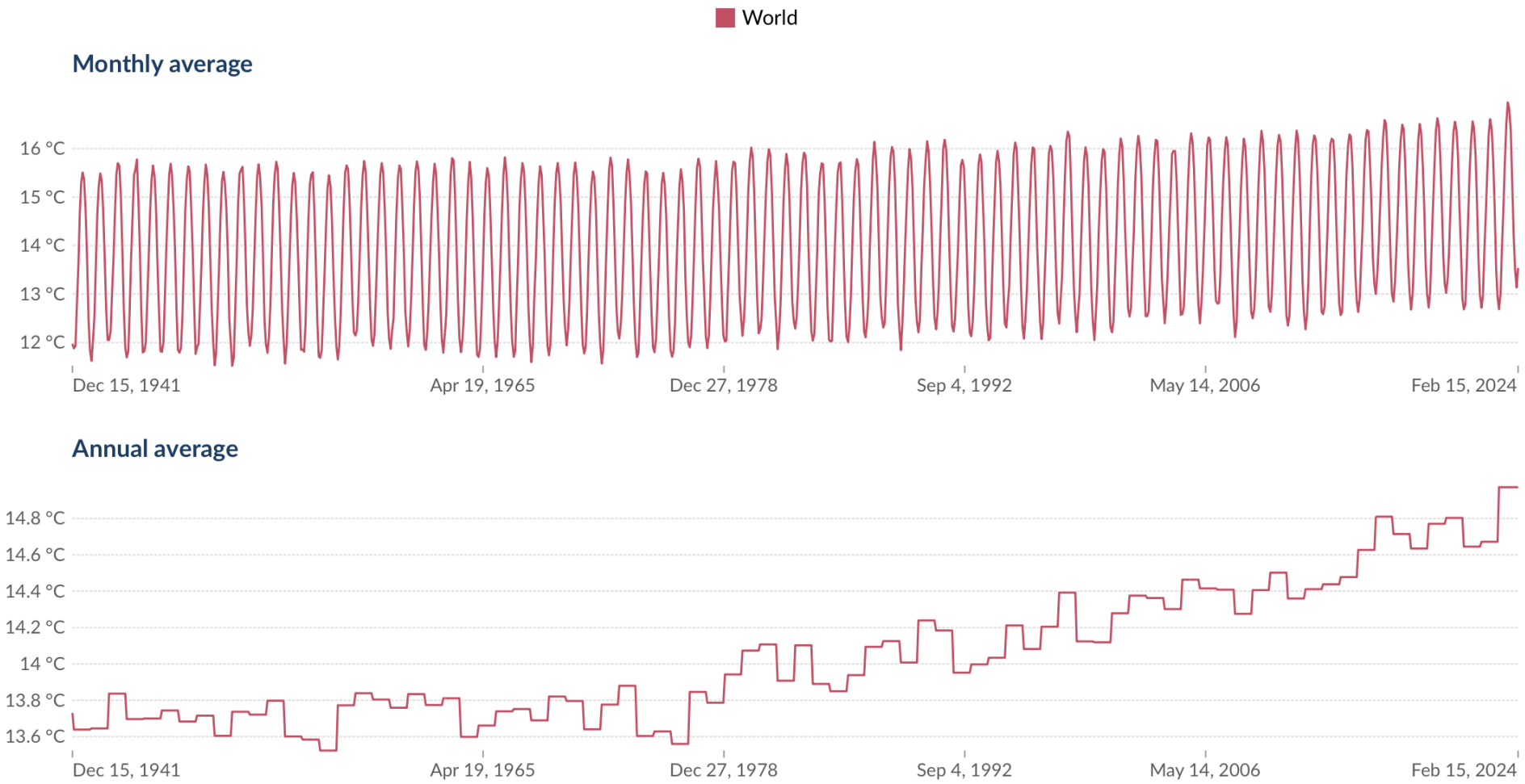


Average monthly surface temperature, World, Dec 15, 1941 to Feb 15, 2024

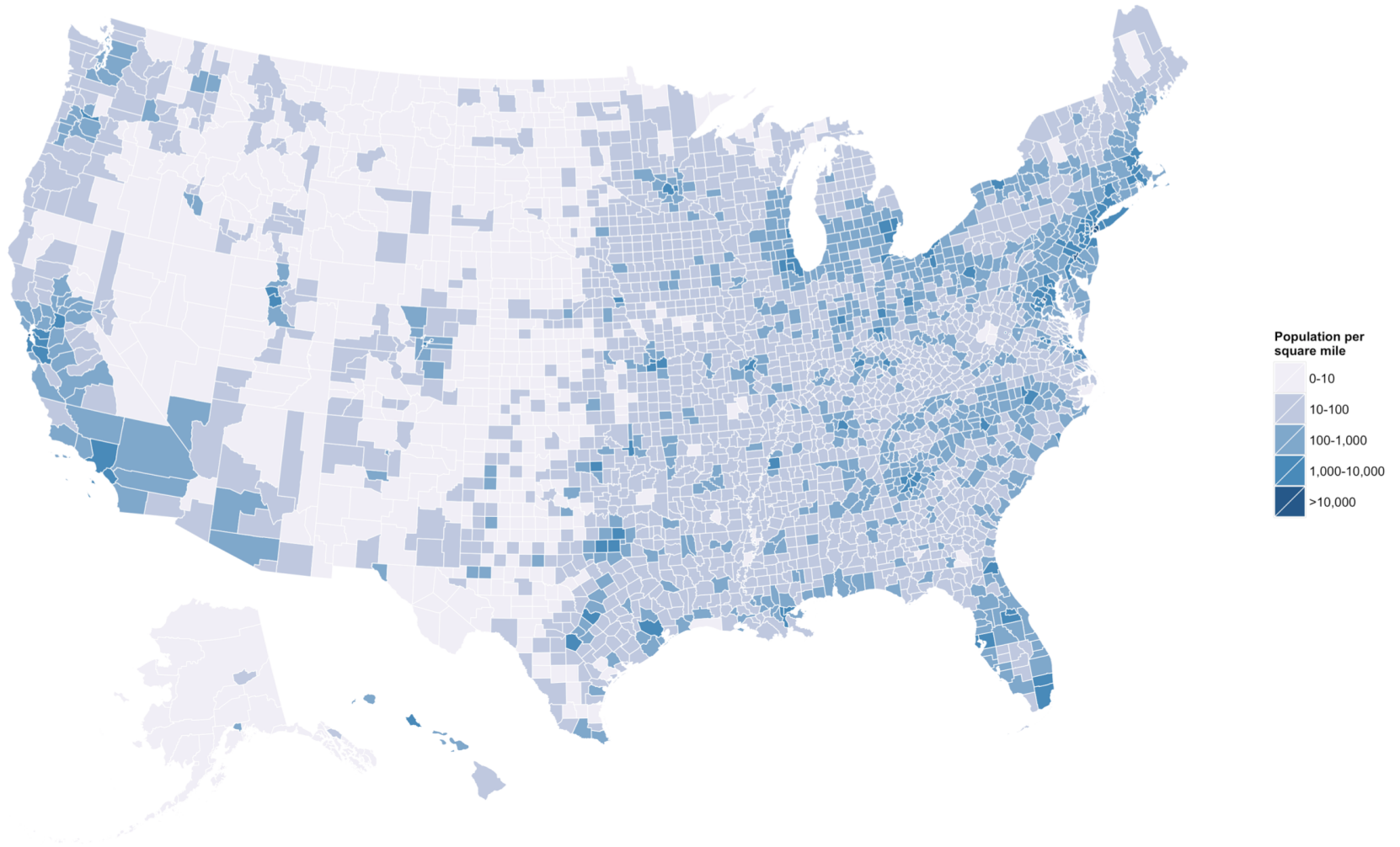
Our World in Data

Table | Map | Chart

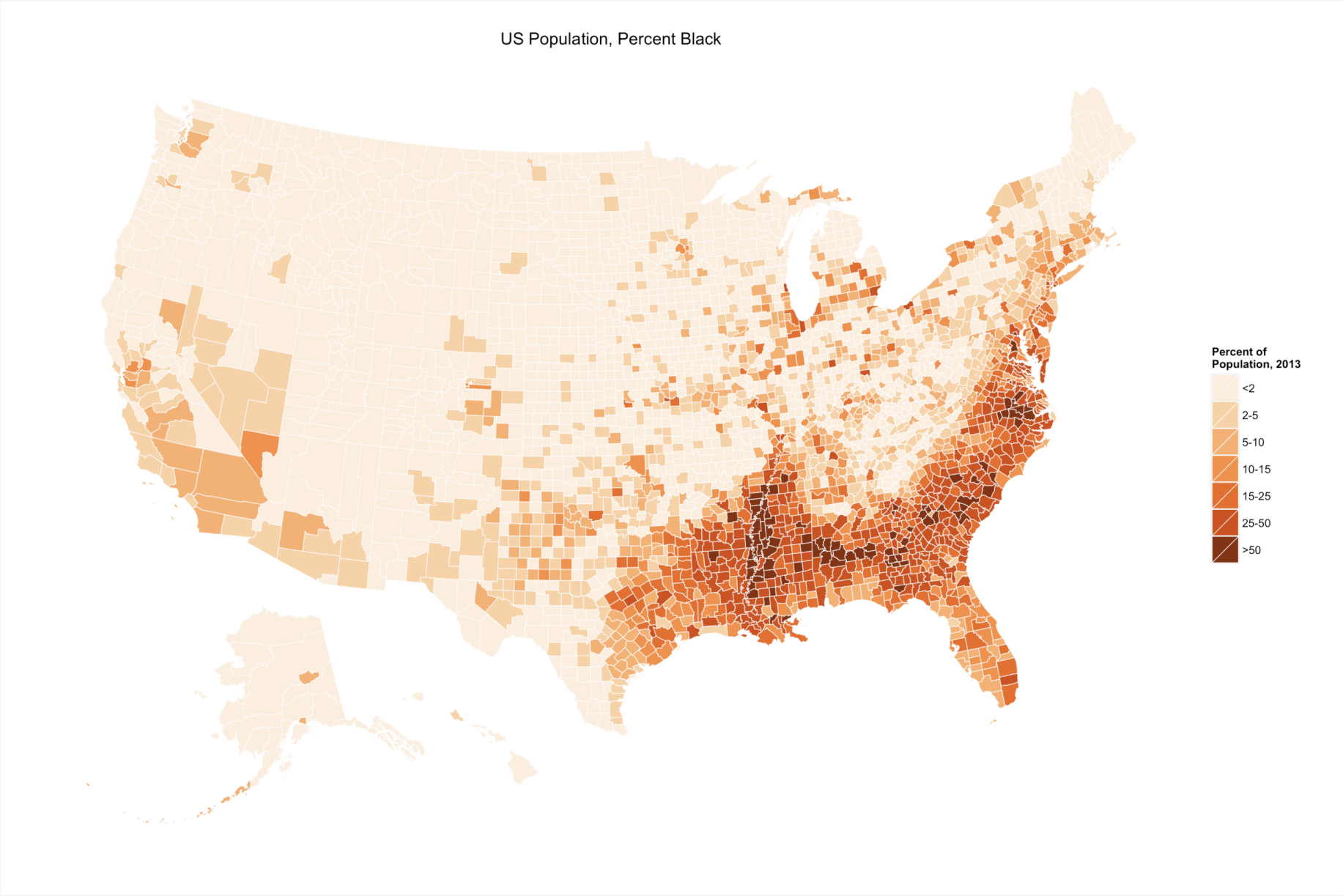
Edit countries and regions | Settings

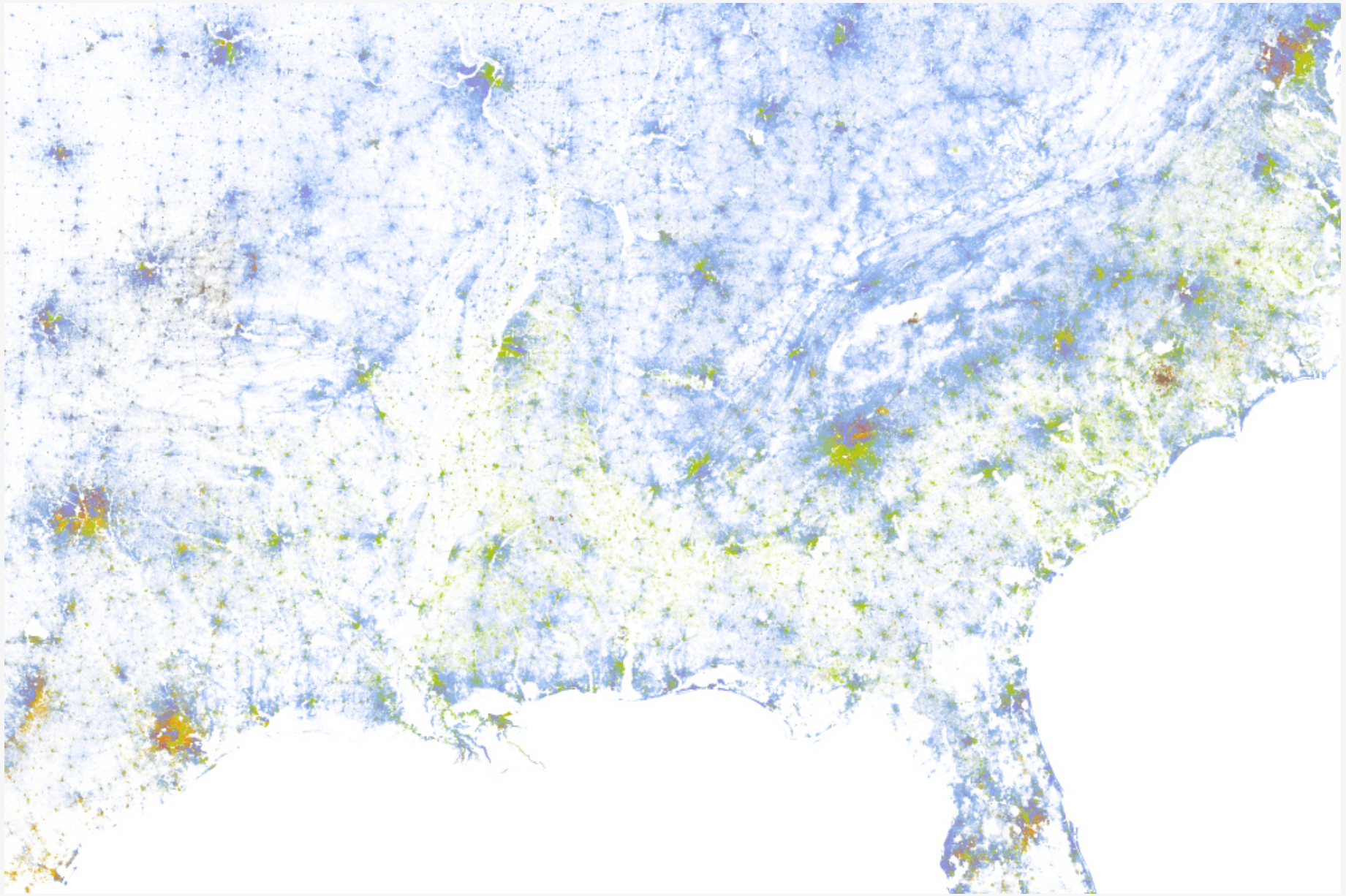


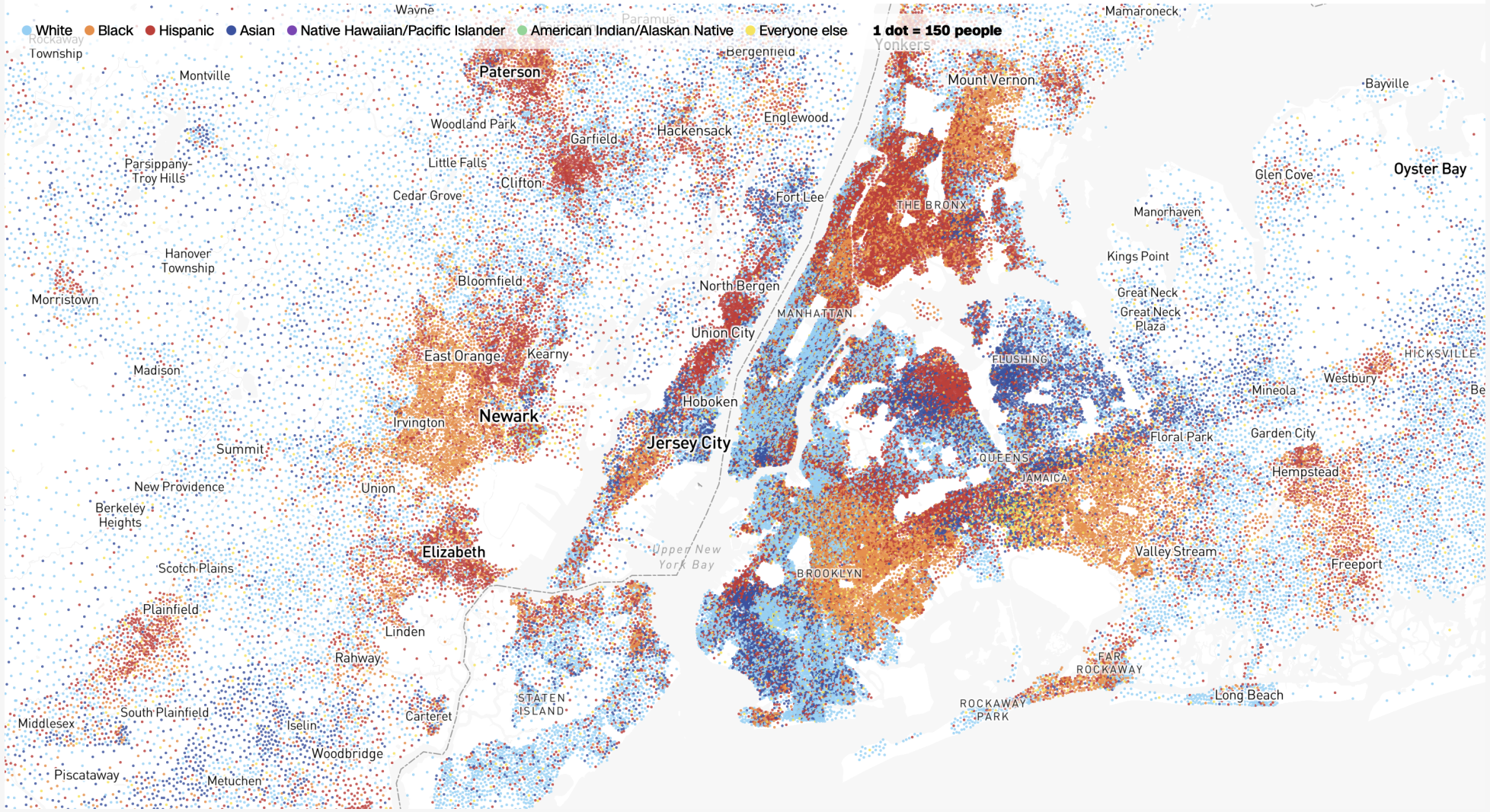
US Population Density, 2014

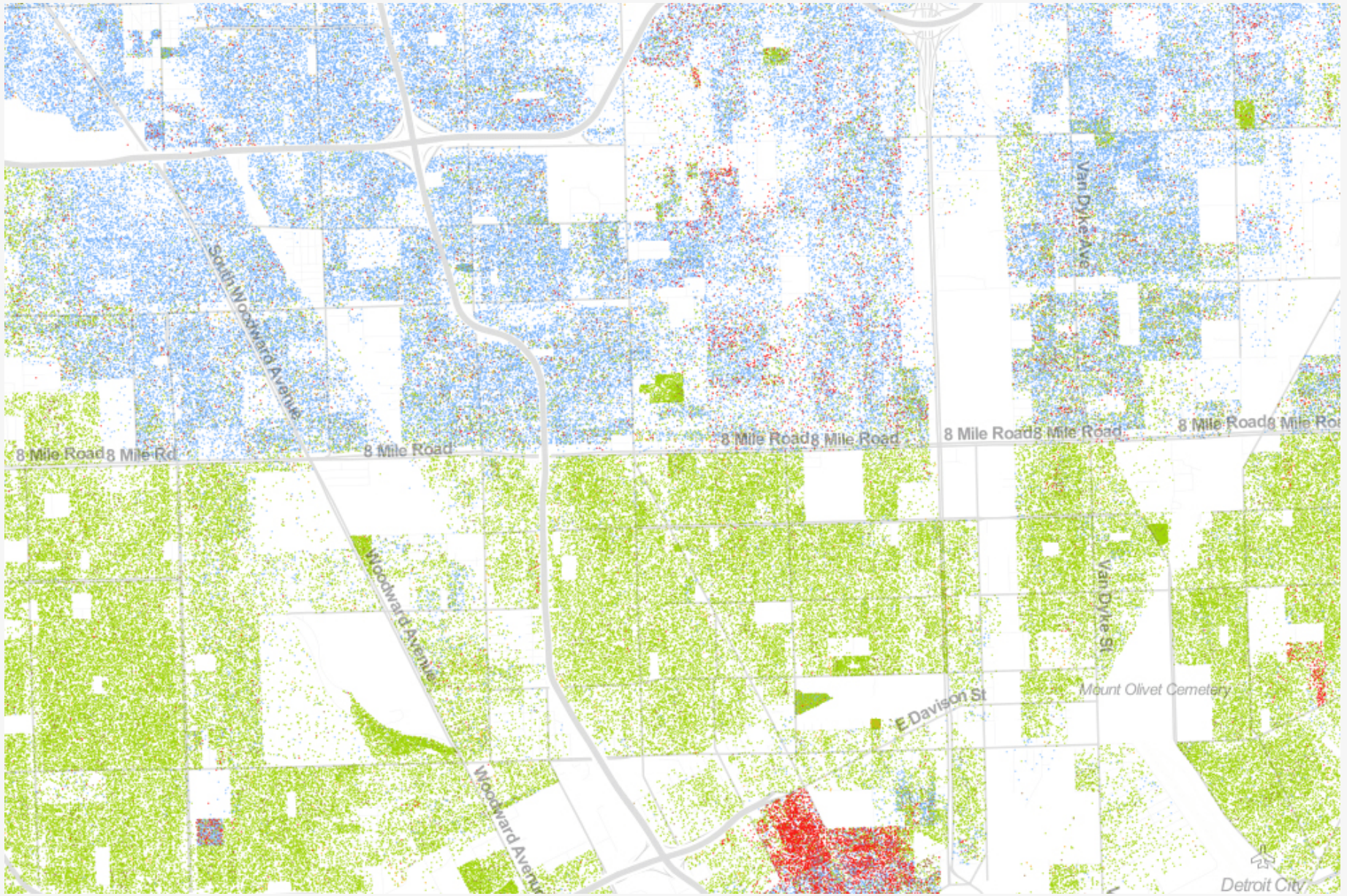


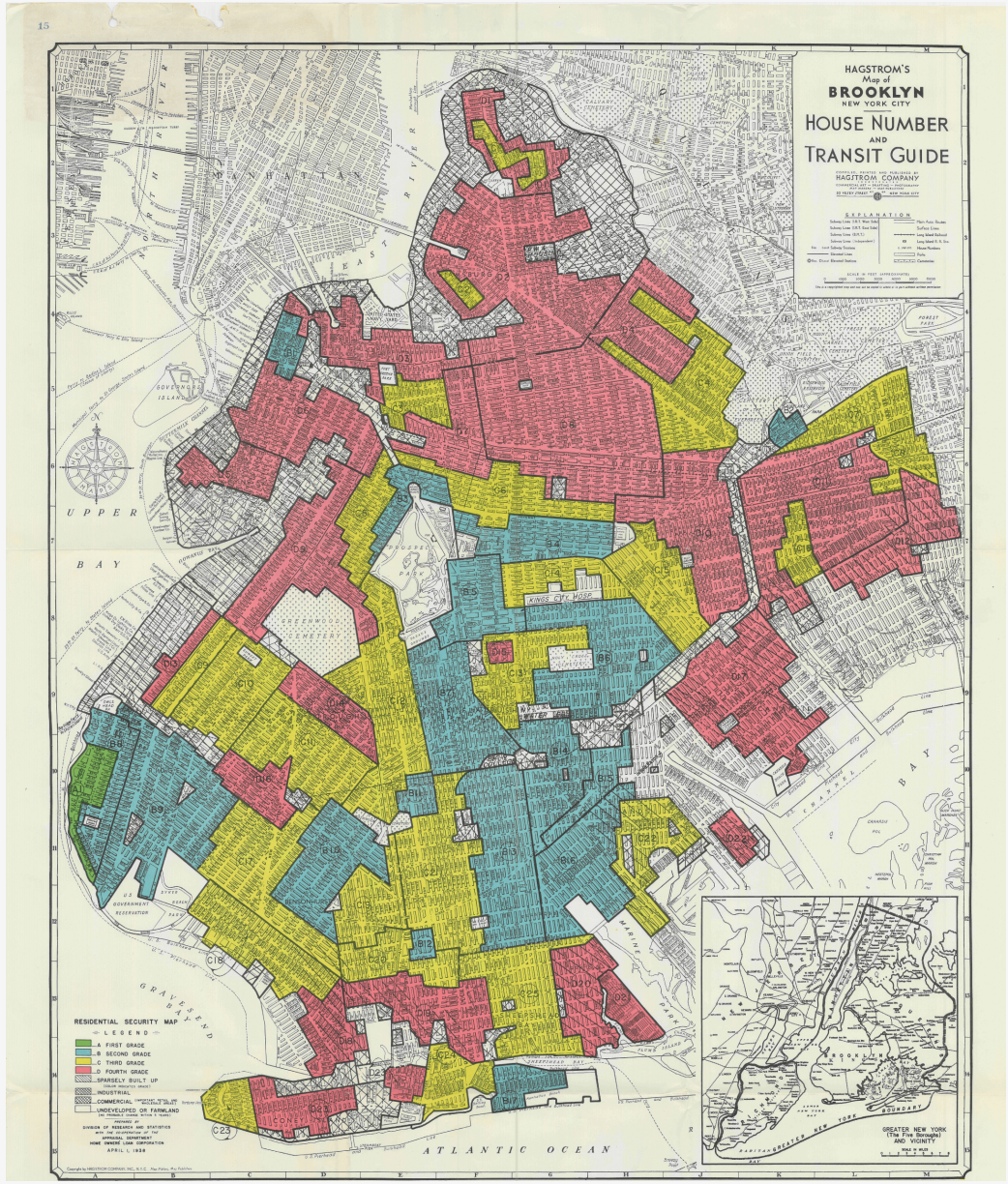
US Population, Percent Black

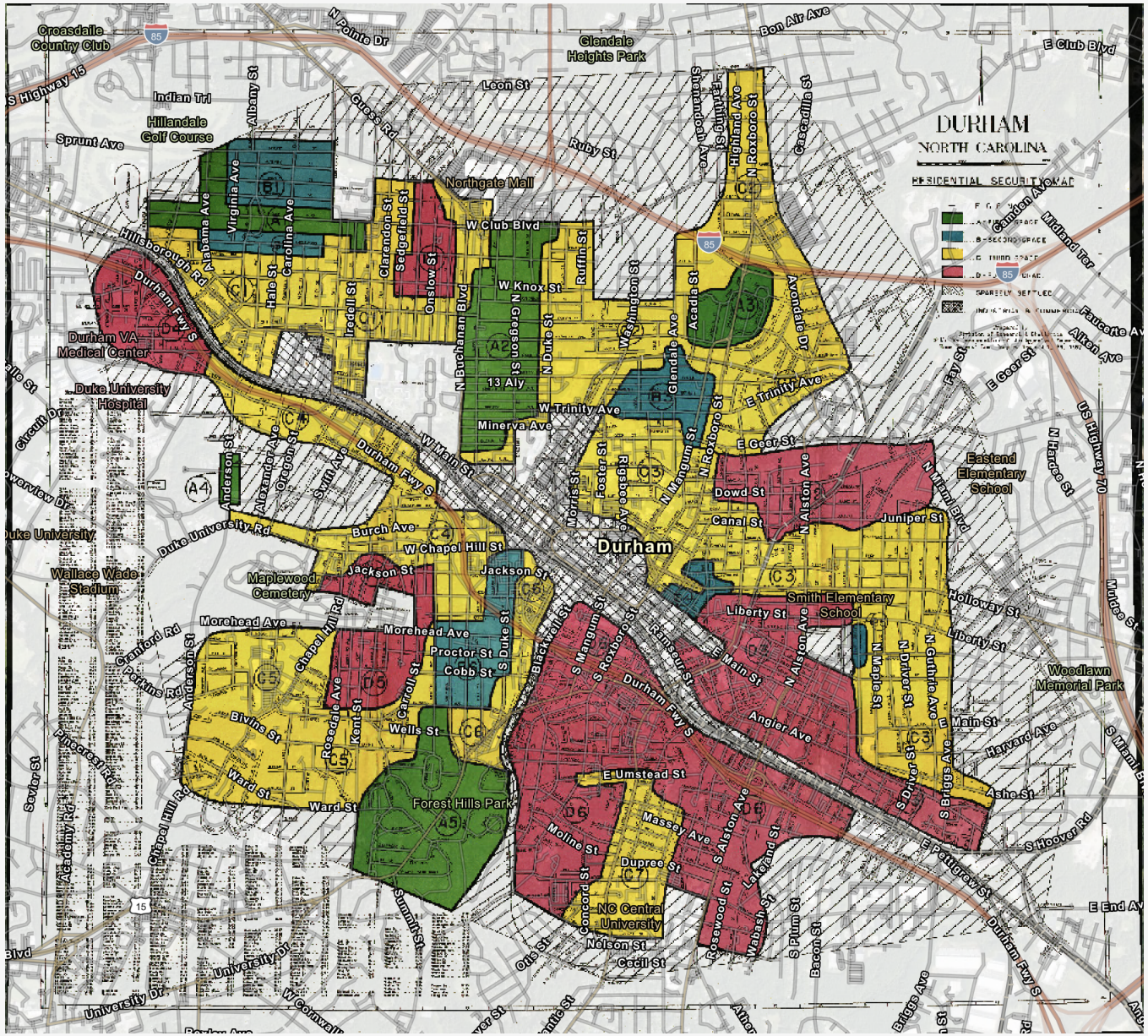


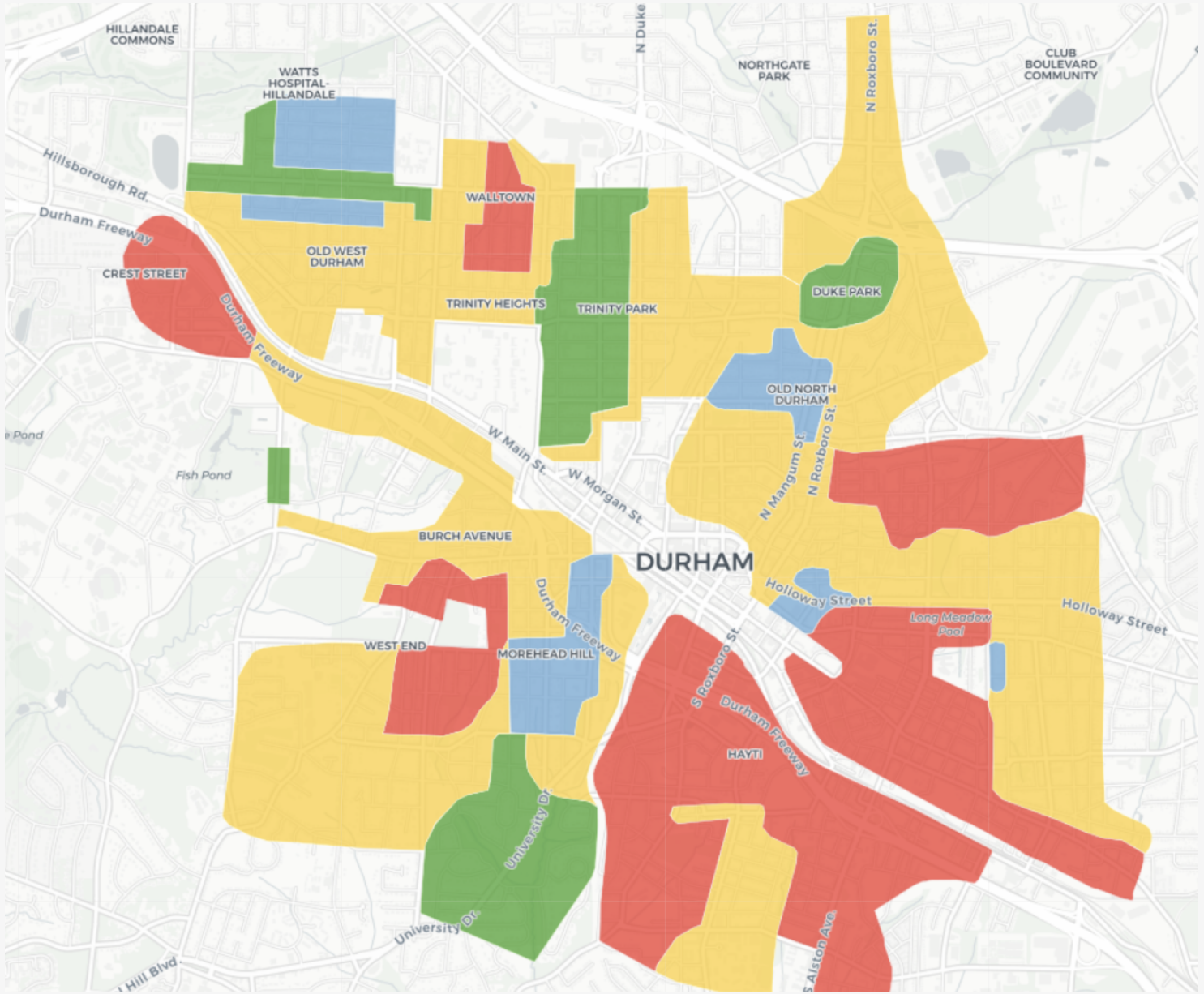



White
Black
Hispanic
Asian
Native Hawaiian/Pacific Islander
American Indian/Alaskan Native
Everyone else
1 dot = 150 people
Wayne
Paramus
Mamaroneck
Rockaway Township
Montville
Bergenfield
Paterson
Yonkers
Mount Vernon
Bayville
Englewood
Woodland Park
Garfield
Hackensack
Parsippany-Troy Hills
Little Falls
Clifton
Glen Cove
Oyster Bay
Cedar Grove
Fort Lee
THE BRONX
Manorhaven
Hanover Township
Kings Point
Bloomfield
North Bergen
Morristown
Great Neck
Great Neck Plaza
MANHATTAN
Union City
HICKSVILLE
East Orange
Kearny
FLUSHING
Madison
Westbury
Mineola
Hoboken
Irvington
Newark
Floral Park
Garden City
Jersey City
QUEENS
JAMAICA
Summit
Hempstead
New Providence
Union
Berkeley Heights
Upper New York Bay
Valley Stream
Elizabeth
Freeport
Scotch Plains
BROOKLYN
Plainfield
Linden
Rahway
FAR ROCKAWAY
STATEN ISLAND
Long Beach
South Plainfield
Middlesex
Carteret
ROCKAWAY PARK
Iselin
Woodbridge
Piscataway
Metuchen

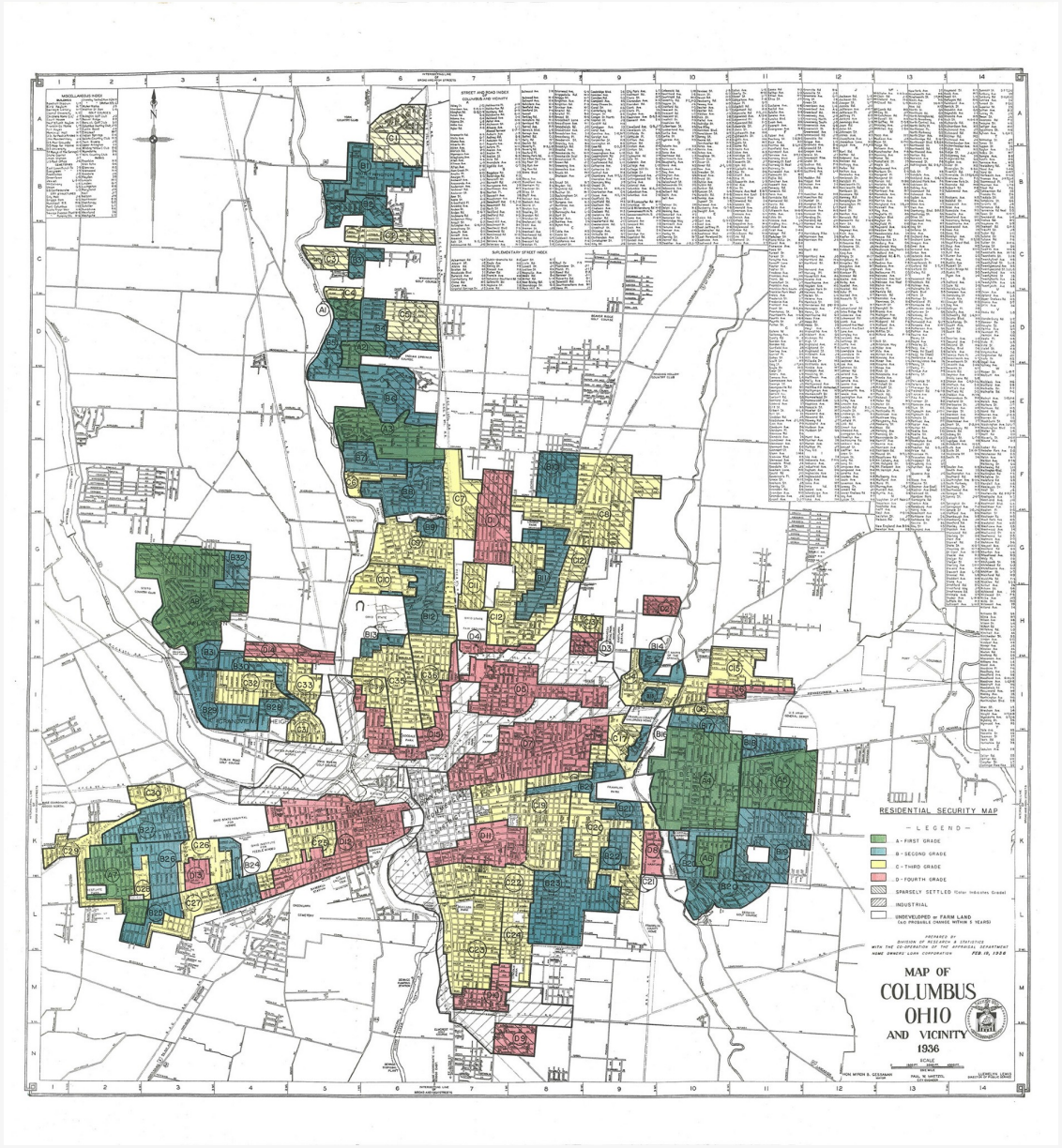

8 Mile Road
8 Mile Rd
8 Mile Road
8 Mile Road
8 Mile Road
8 Mile Road
8 Mile Road
South Woodward Avenue
Woodward Avenue
Woodward Avenue
Van Dyke Ave
Van Dyke St
E Davison St
Mount Olivet Cemetery
Detroit City

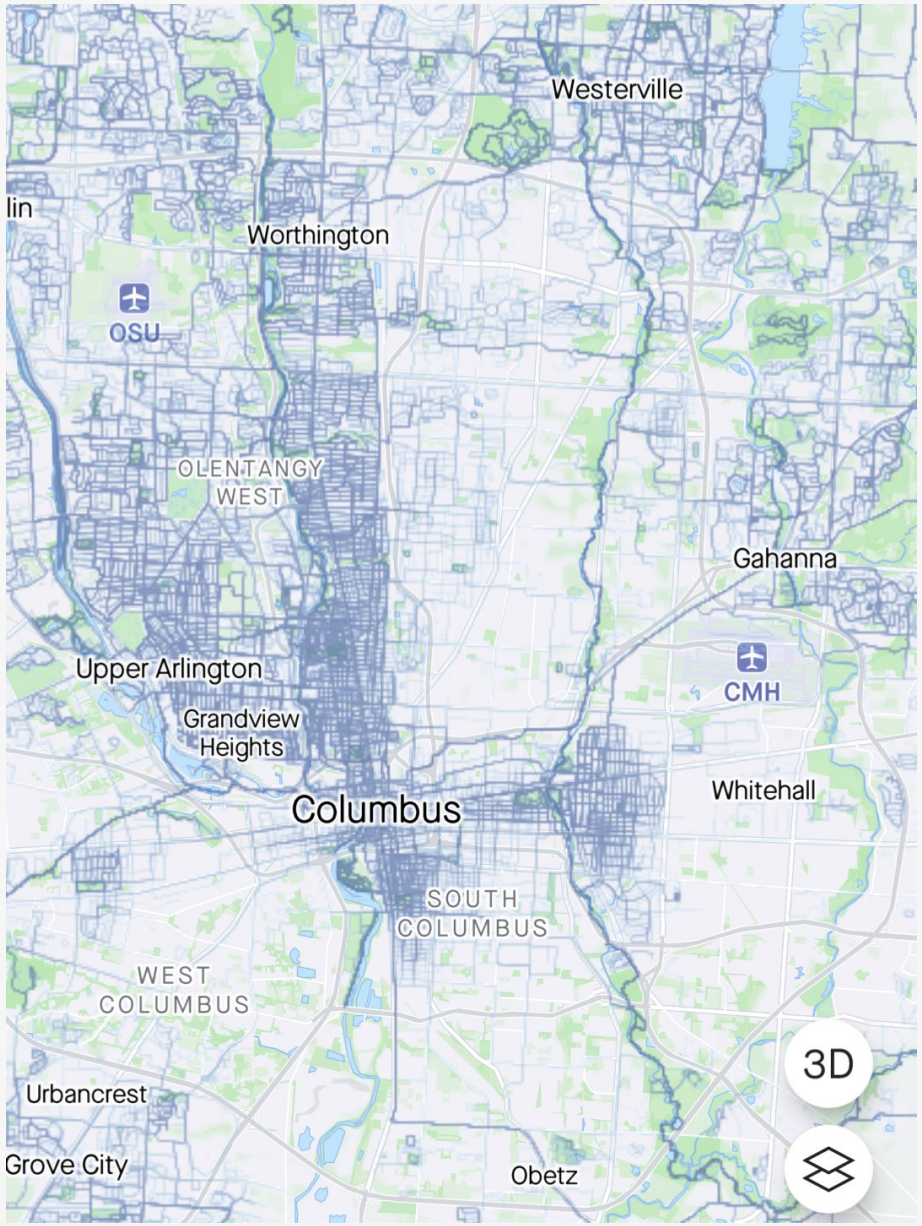

HAGSTROM'S
Map of
BROOKLYN
NEW YORK CITY
HOUSE NUMBER
AND
TRANSIT GUIDE
HAGSTROM COMPANY
EXPLANATION
RESIDENTIAL SECURITY MAP
LEGEND
A FIRST GRADE
B SECOND GRADE
C THIRD GRADE
D FOURTH GRADE
SPARSELY BUILT UP
INDUSTRIAL
COMMERCIAL
UNDEVELOPED OR FARMLAND
PREPARED BY
DIVISION OF RESEARCH AND STATISTICS
HOME OWNERS' LOAN CORPORATION
APRIL 1, 1938
MANHATTAN
UPPER BAY
ATLANTIC OCEAN
GRAVESEND BAY
GREATER NEW YORK
(The Five Boroughs)
AND VICINITY


DURHAM
NORTH CAROLINA
RESIDENTIAL SECURITY MAP
Durham
Croasdaile Country Club
Hillandale Golf Course
Glendale Heights Park
Northgate Mall
Durham VA Medical Center
Duke University Hospital
Wallace Wade Stadium
Maplewood Cemetery
Forest Hills Park
NC Central University
Eastend Elementary School
Smith Elementary School
Woodlawn Memorial Park
Durham Fwy S
W Main St
Duke University Rd
W Chapel Hill St
Morehead Ave
E Main St
Angler Ave
E Umstead St
Ward St
Wells St
Burch Ave
W Trinity Ave
E Trinity Ave
E Geer St
Dowd St
Canal St
Juniper St
Liberty St
Holloway St
Massey Ave
Dupree St
Nelson St
Cecil St
Ashe St
Harvard Ave
E Pettigrew St
S Hoover Rd
Jackson St
Morehead Ave
Proctor St
Cobb St
Blvins St
Minerva Ave
W Knox St
W Club Blvd
Leon St
Ruby St
Bon Air Ave
E Club Blvd
Hillsborough Rd
Sprunt Ave
Indian Trl
Cross Rd
N Pointe Dr
S Highway 15
Alabama Ave
Virginia Ave
Carolina Ave
Albany St
Hale St
Iredell St
Clarendon St
Sedgefield St
Onslow St
Ruffin St
N Buchanan Blvd
N Duke St
Gregson St
13 Aly
Washington St
Glendale Ave
Acadia St
Avondale Dr
Cascadilla St
N Roxboro St
Highland Ave
Shenandoah Ave
Faircloth St
Camden Ave
Midland Ter
Faucette Ave
Alton Ave
Fay St
US Highway 70
N Hardee St
Murdoc St
N Miami Blvd
N Alston Ave
N Mangum St
Foster St
Rigsbee Ave
Morris St
Anderson St
Alexander Ave
Oregon St
Swift Ave
Duke University
Overview Dr
Circuit Dr
Cranford Rd
Perkins Rd
Chapel Hill Rd
Rosedale Ave
Kent St
Carroll St
S Duke St
Blackwell St
S Mangum St
S Roxboro St
Ransom St
E Main St
N Alston Ave
N Maple St
N Driver St
N Gregson St
S Driver St
S Briggs Ave
Lakeland St
Rosewood St
Wabash St
S Plum St
Bacon St
Concord St
Melane St
Sovlar St
Pinecrest Rd
Academy Rd
Crispel Hill Rd
Summit St
University Dr
W Cornwallis Rd
Otis St
Briggs Ave
Durham Fwy S
E End Ave
Harden Ave
C1
C2
C3
C4
C5
C6
C7
A1
A2
A3
A4
A5
B1
B2
B3
B4
B5
D1
D2
D3
D4
D5
D6
D7


HILLANDALE COMMONS
WATTS HOSPITAL-HILLANDALE
N Duke
NORTHGATE PARK
N Roxboro St.
CLUB BOULEVARD COMMUNITY
Hillsborough Rd.
Durham Freeway
WALLTOWN
OLD WEST DURHAM
CREST STREET
TRINITY HEIGHTS
TRINITY PARK
DUKE PARK
OLD NORTH DURHAM
N Mangum St.
N Roxboro St.
W Main St.
W Morgan St.
Fish Pond
BURCH AVENUE
DURHAM
Holloway Street
Long Meadow Pool
WEST END
MOREHEAD HILL
S Roxboro St.
HAYTI
University Dr.
Alston Ave.


HILLANDALE COMMONS
WATTS HOSPITAL-HILLANDALE
Hillandale Rd
Hillsborough Rd
Ruby St
NORTHGATE PARK
N Roxboro St
Hugo St
CLUB BOULEVARD COMMUNITY
Midland Ter
Camden Ave
WALLTOWN
Engtewood Ave
Ruffin St
W Knox St
Morreene Rd
CREST STREET
OLD WEST DURHAM
Iredell St
TRINITY HEIGHTS
TRINITY HISTORIC DISTRICT
Fay St
E Geer St
Drew St
Dowd St
Juniper St
Durham
N Alston Ave
N Hyde Park Ave
Spruce St
N Miami Blvd
Muldee St
N Hoover Rd
Liberty St
WEST END
MOREHEAD HILL
Kent St
Taylor St
HAYTI
Bivins St
Hart St
Harvard Ave
Ashe St
W Cornwallis Rd
Academy Rd
Chapel Hill Rd
Fayetteville Rd
E Pettigrew St
S Plum St
Auto Dr
Ote St
Hayes
E End Connector


RESIDENTIAL SECURITY MAP
— LEGEND —
A - FIRST GRADE
B - SECOND GRADE
C - THIRD GRADE
D - FOURTH GRADE
SPARSELY SETTLED (Color Indicates Grade)
INDUSTRIAL
UNDEVELOPED or FARM LAND (NO PROBABLE CHANGE WITHIN 5 YEARS)
PREPARED BY
DIVISION OF RESEARCH & STATISTICS
WITH THE CO-OPERATION OF THE APPRAISAL DEPARTMENT
HOME OWNERS' LOAN CORPORATION
FEB. 15, 1936
MAP OF
COLUMBUS
OHIO
AND VICINITY
1936
SCALE
JOHN HOWE S. GESSAMAN
PAUL W. WATTEL
LUMBERTH LEWIS


Westerville
lin
Worthington
OSU
OLENTANGY
WEST
Gahanna
CMH
Upper Arlington
Grandview
Heights
Whitehall
Columbus
SOUTH
COLUMBUS
WEST
COLUMBUS
3D
Urbancrest
Grove City
Obetz

Text as Data

Load the packages, as always

```
library(here)      # manage file paths
library(socviz)    # data and some useful functions
library(tidyverse) # your friend and mine
```

Specialty packages

```
#install.packages("tidytext")
#install.packages("gutenbergr")
#install.packages("janeaustenr")

library(tidytext)    # Tools for analyzing text
library(gutenbergr)  # Get books from Project Gutenberg
library(janeaustenr) # Pre-organized dataset of Jane Austen's novels
```

This week's examples are mostly taken from Silge and Robinson (2017), *Text Mining with R*.

Tidy text

```
original_books ← austen_books() ▷
  group_by(book) ▷
  mutate(linenumber = row_number(),
         chapter = cumsum(str_detect(text,
                                     regex("^chapter [\\divxlc]",
                                           ignore_case = TRUE)))) ▷
  ungroup()

tidy_books ← original_books ▷
  unnest_tokens(word, text)

tidy_books
```

```
# A tibble: 725,055 × 4
   book                linenumber chapter word
   <fct>                    <int>   <int> <chr>
 1 Sense & Sensibility          1       0 sense
 2 Sense & Sensibility          1       0 and
 3 Sense & Sensibility          1       0 sensibility
 4 Sense & Sensibility          3       0 by
 5 Sense & Sensibility          3       0 jane
 6 Sense & Sensibility          3       0 austen
 7 Sense & Sensibility          5       0 1811
 8 Sense & Sensibility         10       1 chapter
 9 Sense & Sensibility         10       1 1
10 Sense & Sensibility         13       1 the
# ℹ 725,045 more rows
```

“Stopwords”

For many purposes (not always!) very common words like prepositions and articles are not interesting.

```
data(stop_words)

stop_words
```

```
# A tibble: 1,149 × 2
   word        lexicon
   <chr>       <chr>
 1 a           SMART
 2 a's         SMART
 3 able        SMART
 4 about       SMART
 5 above       SMART
 6 according   SMART
 7 accordingly SMART
 8 across      SMART
 9 actually    SMART
10 after       SMART
# ℹ 1,139 more rows
```

```
tidy_books ← tidy_books ▷
  anti_join(stop_words)
```

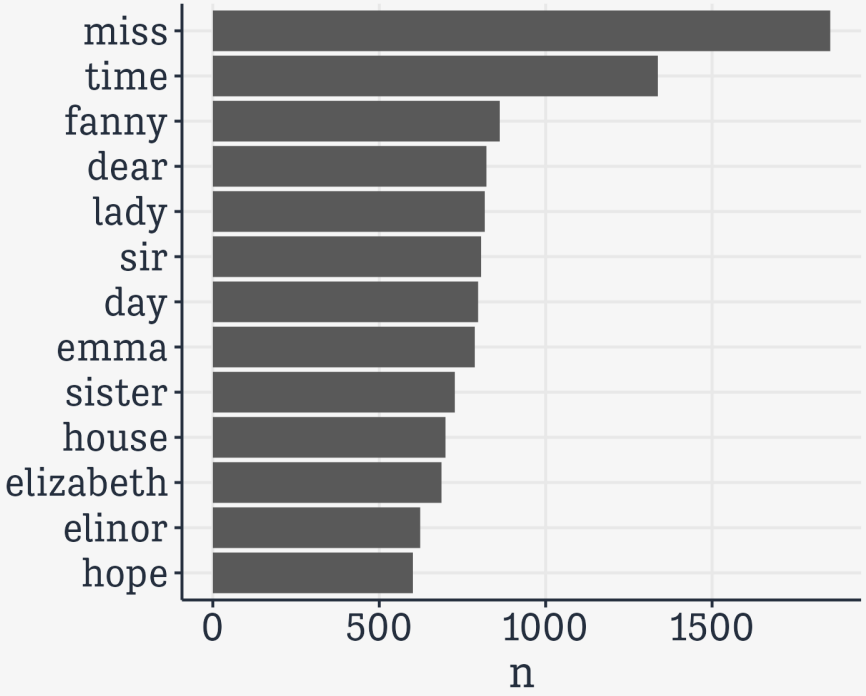
Stopwords removed

```
tidy_books ▷
  count(word, sort = TRUE)
```

```
# A tibble: 13,914 × 2
   word       n
   <chr>  <int>
 1 miss    1855
 2 time    1337
 3 fanny    862
 4 dear     822
 5 lady     817
 6 sir      806
 7 day      797
 8 emma     787
 9 sister   727
10 house    699
# ℹ 13,904 more rows
```

Word frequency

```
tidy_books ▷
  count(word, sort = TRUE) ▷
  filter(n > 600) ▷
  mutate(word = reorder(word, n)) ▷
  ggplot(aes(n, word)) + geom_col() +
  labs(y = NULL)
```



tf-idf

Stands for “Term Frequency–Inverse Document Frequency”

The idea is to count the frequency of terms in a document, but decrease the weight of commonly used words and increase the weight for words that are not used very much in a corpus.

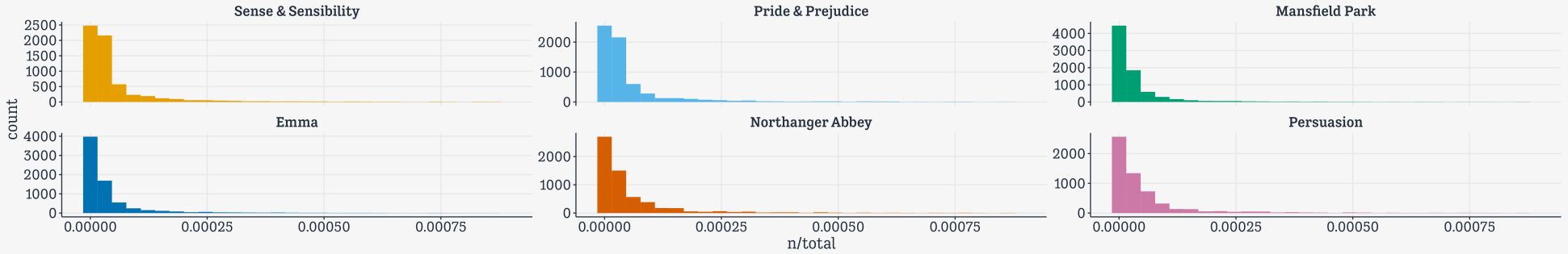
For example ...

```
book_words ← austen_books() ▷
  unnest_tokens(word, text) ▷
  count(book, word, sort = TRUE)

total_words ← book_words ▷
  group_by(book) ▷
  summarize(total = sum(n))

book_words ← left_join(book_words, total_words)

book_words
```

```
# A tibble: 40,379 × 4
   book              word      n  total
   <fct>             <chr> <int>  <int>
 1 Mansfield Park    the    6206 160460
 2 Mansfield Park    to     5475 160460
 3 Mansfield Park    and    5438 160460
 4 Emma              to     5239 160996
 5 Emma              the    5201 160996
 6 Emma              and    4896 160996
 7 Mansfield Park    of     4778 160460
 8 Pride & Prejudice the    4331 122204
 9 Emma              of     4291 160996
10 Pride & Prejudice to     4162 122204
# ℹ 40,369 more rows
```

For example ...

```
ggplot(book_words, aes(n/total, fill = book)) +
  geom_histogram(show.legend = FALSE) +
  xlim(NA, 0.0009) +
  facet_wrap(~book, nrow = 2, scales = "free_y")
```



Zipf's Law

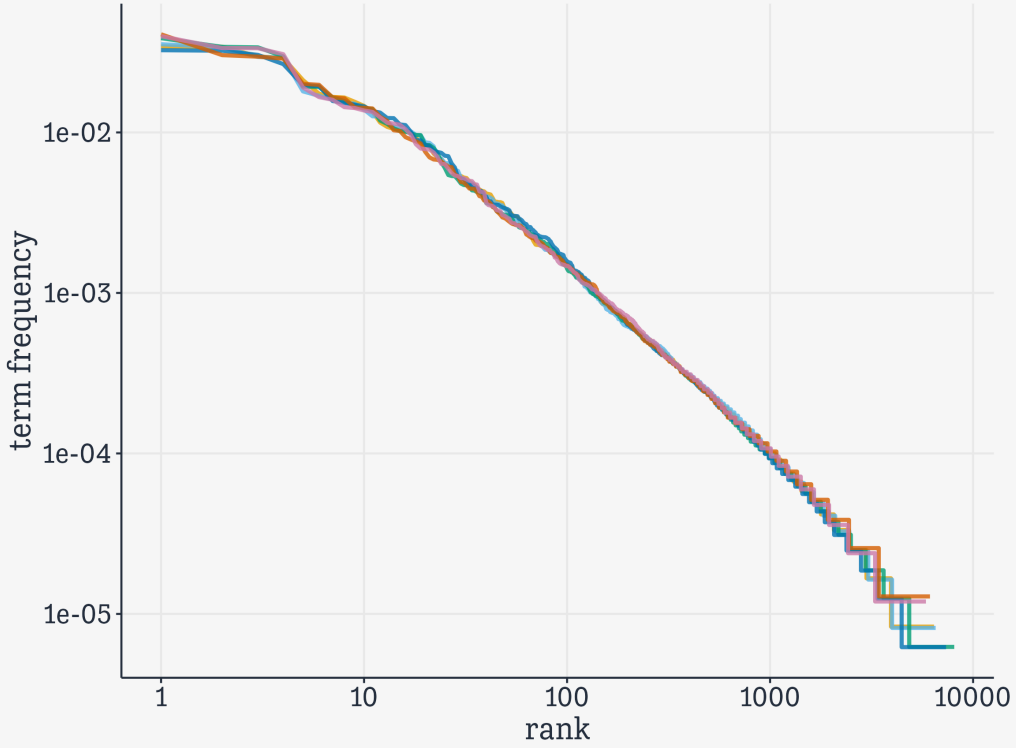
"The frequency that a word appears is inversely proportional to its rank."

```
freq_by_rank ← book_words ▷
  group_by(book) ▷
  mutate(rank = row_number(),
         `term frequency` = n/total) ▷
  ungroup()

freq_by_rank
```

```
# A tibble: 40,379 × 6
   book              word      n  total  rank `term frequency`
   <fct>             <chr> <int>  <int> <int>            <dbl>
 1 Mansfield Park    the    6206 160460     1           0.0387
 2 Mansfield Park    to     5475 160460     2           0.0341
 3 Mansfield Park    and    5438 160460     3           0.0339
 4 Emma              to     5239 160996     1           0.0325
 5 Emma              the    5201 160996     2           0.0323
 6 Emma              and    4896 160996     3           0.0304
 7 Mansfield Park    of     4778 160460     4           0.0298
 8 Pride & Prejudice the    4331 122204     1           0.0354
 9 Emma              of     4291 160996     4           0.0267
10 Pride & Prejudice to     4162 122204     2           0.0341
# ℹ 40,369 more rows
```

Zipf's Law

```
freq_by_rank ▷
  ggplot(aes(rank, `term frequency`, color = book)) +
  geom_line(linewidth = 1.1, alpha = 0.8, show.legend = FALSE) +
  scale_x_log10() +
  scale_y_log10()
```



n-grams

```
austen_bigrams ← austen_books() ▷
  unnest_tokens(bigram, text, token = "ngrams", n = 2) ▷
  filter(!is.na(bigram))

austen_bigrams
```

```
# A tibble: 662,783 × 2
   book                bigram
   <fct>               <chr>
 1 Sense & Sensibility sense and
 2 Sense & Sensibility and sensibility
 3 Sense & Sensibility by jane
 4 Sense & Sensibility jane austen
 5 Sense & Sensibility chapter 1
 6 Sense & Sensibility the family
 7 Sense & Sensibility family of
 8 Sense & Sensibility of dashwood
 9 Sense & Sensibility dashwood had
10 Sense & Sensibility had long
# ℹ 662,773 more rows
```

n-grams

```
austen_bigrams ▷
  count(bigram, sort = TRUE)
```

```
# A tibble: 193,209 × 2
   bigram       n
   <chr>    <int>
 1 of the    2853
 2 to be     2670
 3 in the    2221
 4 it was    1691
 5 i am      1485
 6 she had   1405
 7 of her    1363
 8 to the    1315
 9 she was   1309
10 had been  1206
# ℹ 193,199 more rows
```

Stopwords again.

n-grams

Split the columns

```
bigrams_separated ← austen_bigrams ▷
  separate(bigram, c("word1", "word2"), sep = " ")

bigrams_filtered ← bigrams_separated ▷
  filter(!word1 %in% stop_words$word) ▷
  filter(!word2 %in% stop_words$word)

# new bigram counts:
bigram_counts ← bigrams_filtered ▷
  count(word1, word2, sort = TRUE)

bigram_counts
```

```
# A tibble: 28,974 × 3
   word1   word2         n
   <chr>   <chr>     <int>
 1 sir     thomas      266
 2 miss    crawford    196
 3 captain wentworth   143
 4 miss    woodhouse   143
 5 frank   churchill   114
 6 lady    russell     110
 7 sir     walter      108
 8 lady    bertram     101
 9 miss    fairfax      98
```

n-grams

```
bigrams_united ← bigrams_filtered ▷
  unite(bigram, word1, word2, sep = " ")

bigrams_united
```

```
# A tibble: 38,913 × 2
   book                bigram
   <fct>               <chr>
 1 Sense & Sensibility jane austen
 2 Sense & Sensibility chapter 1
 3 Sense & Sensibility norland park
 4 Sense & Sensibility surrounding acquaintance
 5 Sense & Sensibility late owner
 6 Sense & Sensibility advanced age
 7 Sense & Sensibility constant companion
 8 Sense & Sensibility happened ten
 9 Sense & Sensibility henry dashwood
10 Sense & Sensibility norland estate
# ℹ 38,903 more rows
```

Now we have common bigrams without stopwords.

n-gram tf-idf

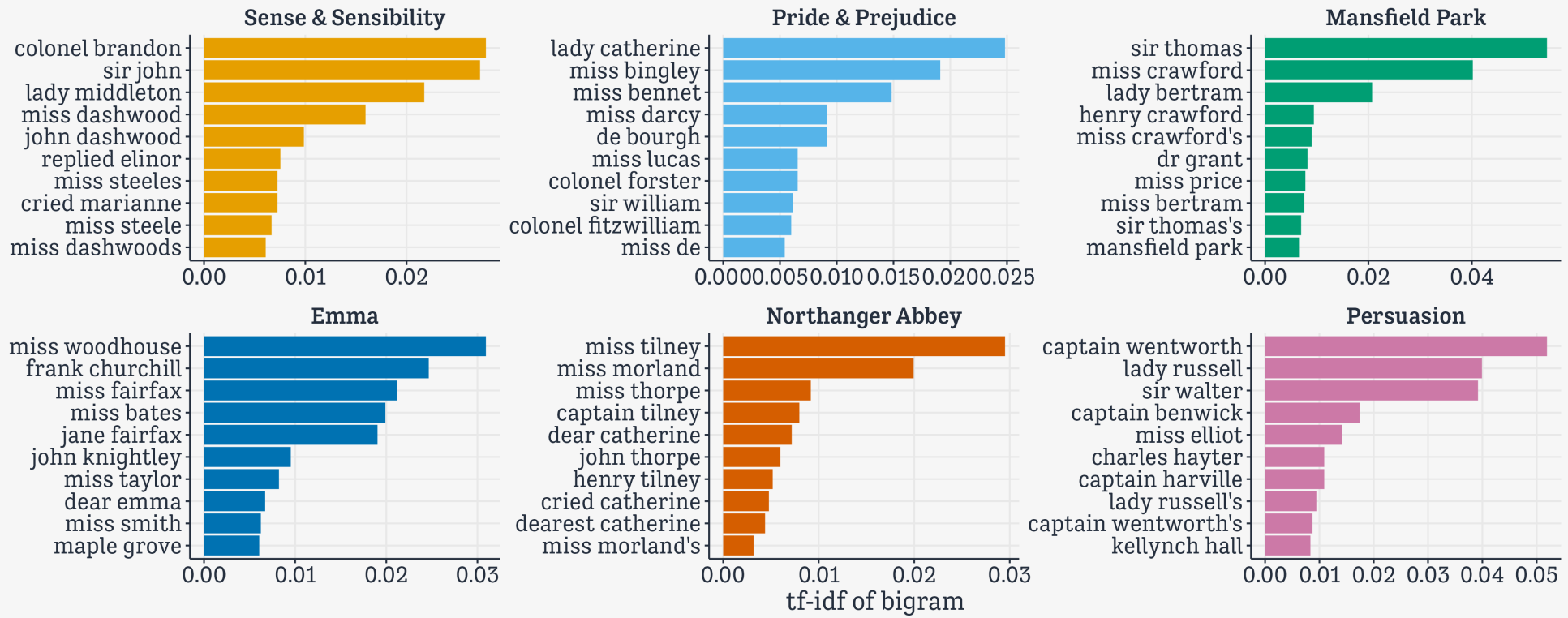
```
bigram_tf_idf ← bigrams_united ▷
  count(book, bigram) ▷
  bind_tf_idf(bigram, book, n) ▷
  arrange(desc(tf_idf))

bigram_tf_idf
```

```
# A tibble: 31,391 × 6
   book                bigram               n     tf   idf tf_idf
   <fct>               <chr>            <int>  <dbl> <dbl>  <dbl>
 1 Mansfield Park      sir thomas         266 0.0304  1.79 0.0545
 2 Persuasion          captain wentworth  143 0.0290  1.79 0.0519
 3 Mansfield Park      miss crawford      196 0.0224  1.79 0.0402
 4 Persuasion          lady russell       110 0.0223  1.79 0.0399
 5 Persuasion          sir walter         108 0.0219  1.79 0.0392
 6 Emma                miss woodhouse     143 0.0173  1.79 0.0309
 7 Northanger Abbey    miss tilney         74 0.0165  1.79 0.0295
 8 Sense & Sensibility colonel brandon     96 0.0155  1.79 0.0278
 9 Sense & Sensibility sir john            94 0.0152  1.79 0.0273
10 Pride & Prejudice   lady catherine      87 0.0139  1.79 0.0248
# ℹ 31,381 more rows
```

Plot them

```
out ← bigram_tf_idf ▷
  arrange(desc(tf_idf)) ▷
  group_by(book) ▷
  slice_max(tf_idf, n = 10) ▷
  ungroup() ▷
  mutate(bigram = reorder(bigram, tf_idf)) ▷
  ggplot(aes(tf_idf, bigram, fill = book)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ book, nrow = 2, scales = "free") +
  labs(x = "tf-idf of bigram", y = NULL)
```

```
print(out)
```


Sense & Sensibility
colonel brandon
sir john
lady middleton
miss dashwood
john dashwood
replied elinor
miss steeles
cried marianne
miss steele
miss dashwoods
0.00
0.01
0.02
Pride & Prejudice
lady catherine
miss bingley
miss bennet
miss darcy
de bourgh
miss lucas
colonel forster
sir william
colonel fitzwilliam
miss de
0.000
0.005
0.010
0.015
0.020
0.025
Mansfield Park
sir thomas
miss crawford
lady bertram
henry crawford
miss crawford's
dr grant
miss price
miss bertram
sir thomas's
mansfield park
0.00
0.02
0.04
Emma
miss woodhouse
frank churchill
miss fairfax
miss bates
jane fairfax
john knightley
miss taylor
dear emma
miss smith
maple grove
0.00
0.01
0.02
0.03
Northanger Abbey
miss tilney
miss morland
miss thorpe
captain tilney
dear catherine
john thorpe
henry tilney
cried catherine
dearest catherine
miss morland's
0.00
0.01
0.02
0.03
tf-idf of bigram
Persuasion
captain wentworth
lady russell
sir walter
captain benwick
miss elliot
charles hayter
captain harville
lady russell's
captain wentworth's
kellynch hall
0.00
0.01
0.02
0.03
0.04
0.05

Sentiment Analysis

```
ulysses ← "http://aleph.gutenberg.org/4/3/0/4300/4300-0.txt"
raw_text ← tibble(text = readr::read_lines(ulysses))

raw_text
```

```
# A tibble: 33,216 × 1
   text
   <chr>
 1 "The Project Gutenberg eBook of Ulysses, by James Joyce"
 2 ""
 3 "This eBook is for the use of anyone anywhere in the United States and"
 4 "most other parts of the world at no cost and with almost no restrictions"
 5 "whatsoever. You may copy it, give it away or re-use it under the terms"
 6 "of the Project Gutenberg License included with this eBook or online at"
 7 "www.gutenberg.org. If you are not located in the United States, you"
 8 "will have to check the laws of the country where you are located before"
 9 "using this eBook."
10 ""
# ℹ 33,206 more rows
```

Sentiment Analysis

```
raw_text[74,]
```

```
# A tibble: 1 × 1
  text
  <chr>
1 Stately, plump Buck Mulligan came from the stairhead, bearing a bowl of
```

```
raw_text[nrow(raw_text) - 360,]
```

```
# A tibble: 1 × 1
  text
  <chr>
1 yes I said yes I will Yes.
```

Sentiment Analysis

```
full_text ← raw_text ▷
  mutate(line=row_number()) ▷
  slice(-seq(n(), n() - 359)) ▷ # end
  slice(-seq(1:73)) ▷ # top
  unnest_tokens(word,text)
```

Sentiment Analysis

```
full_text[1:31,] ▷
  print(n = Inf)
```

```
# A tibble: 31 × 2
    line word
   <int> <chr>
 1    74 stately
 2    74 plump
 3    74 buck
 4    74 mulligan
 5    74 came
 6    74 from
 7    74 the
 8    74 stairhead
 9    74 bearing
10    74 a
11    74 bowl
12    74 of
13    75 lather
14    75 on
15    75 which
16    75 a
```

Sentiment Analysis

```
tail(full_text, n = 15)
```

```
# A tibble: 15 × 2
    line word
   <int> <chr>
 1 32855 and
 2 32855 his
 3 32855 heart
 4 32855 was
 5 32855 going
 6 32855 like
 7 32855 mad
 8 32855 and
 9 32856 yes
10 32856 i
11 32856 said
12 32856 yes
13 32856 i
14 32856 will
15 32856 yes
```

Sentiment Analysis

```
full_text ▷
  anti_join(stop_words) ▷
  filter(! str_detect(word, "'")) ▷
  filter(! str_detect(word, "’")) ▷
  count(word, sort = TRUE) ▷
  top_n(20) ▷
  mutate(word=reorder(word, n))
```

```
# A tibble: 20 × 2
   word        n
   <fct>   <int>
 1 bloom     933
 2 stephen   503
 3 time      376
 4 eyes      329
 5 hand      304
 6 street    293
 7 father    277
 8 day       250
 9 round     239
10 night     232
11 head      222
12 sir       217
13 god       215
14 john      195
15 life      192
16 woman     186
```

Sentiment Analysis

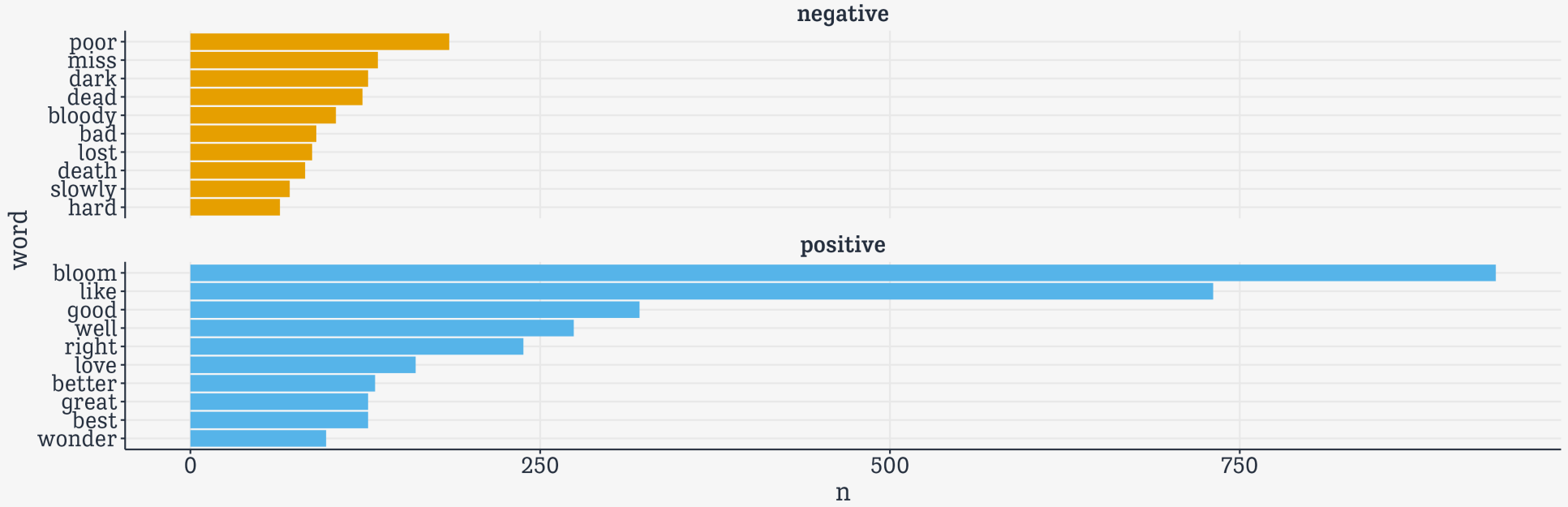
```
sent ← full_text ▷
  inner_join(get_sentiments("bing"), relationship = "many-to-many") ▷
  count(sentiment, word, sort = TRUE) ▷
  group_by(sentiment) ▷
  top_n(10) ▷
  ungroup() ▷
  mutate(word=reorder(word,n))


sent
```

```
# A tibble: 20 × 3
   sentiment word       n
   <chr>     <fct>  <int>
 1 positive  bloom    933
 2 positive  like     731
 3 positive  good     321
 4 positive  well     274
 5 positive  right    238
 6 negative  poor     185
 7 positive  love     161
 8 negative  miss     134
 9 positive  better   132
10 negative  dark     127
11 positive  best     127
12 positive  great    127
13 negative  dead     123
14 negative  bloody   104
```

Sentiment Analysis

```
out ← sent  ▷
  ggplot(mapping = aes(x = n,
                       y = word,
                       fill=sentiment)) +
  geom_col() +
  guides(fill = "none") +
  facet_wrap(~ sentiment,
             ncol = 1,
             scales = "free_y")
```

```
print(out)
```


negative
poor
miss
dark
dead
bloody
bad
lost
death
slowly
hard
positive
bloom
like
good
well
right
love
better
great
best
wonder
word
0
250
500
750
n

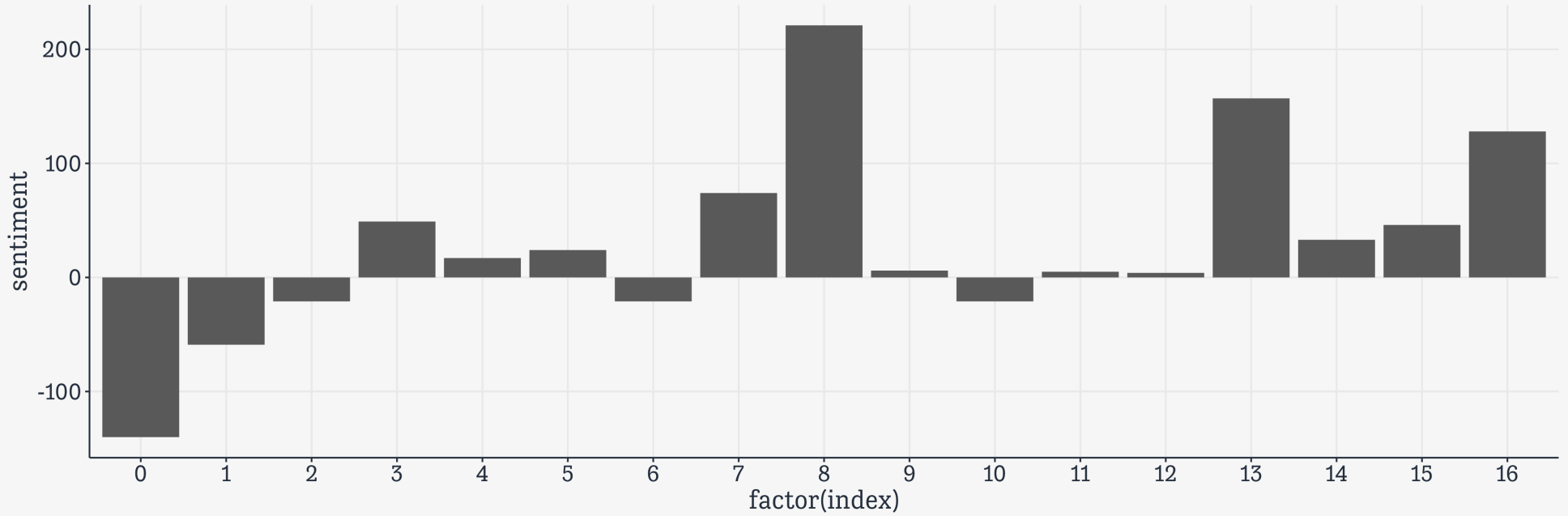
Sentiment Analysis

```
out ← full_text ▷
  inner_join(get_sentiments("bing"), relationship = "many-to-many") ▷
  count(index = line %/% 2000, sentiment) ▷
  pivot_wider(names_from = sentiment,
              values_from = n, values_fill = 0) ▷
  mutate(sentiment = positive - negative)

out
```

```
# A tibble: 17 × 4
   index negative positive sentiment
   <dbl>    <int>    <int>     <int>
 1     0      454      314      -140
 2     1      605      546       -59
 3     2      496      475       -21
 4     3      373      422        49
 5     4      480      497        17
 6     5      357      381        24
 7     6      490      469       -21
 8     7      457      531        74
 9     8      524      745       221
10     9      772      778         6
11    10      543      522       -21
12    11      572      577         5
13    12      464      468         4
14    13      553      710       157
15    14      340      373        33
16    15      607      653        46
```

Sentiment Analysis

```
out |>
  ggplot(mapping = aes(factor(index), sentiment)) +
  geom_col(show.legend = FALSE)
```



Pronouns

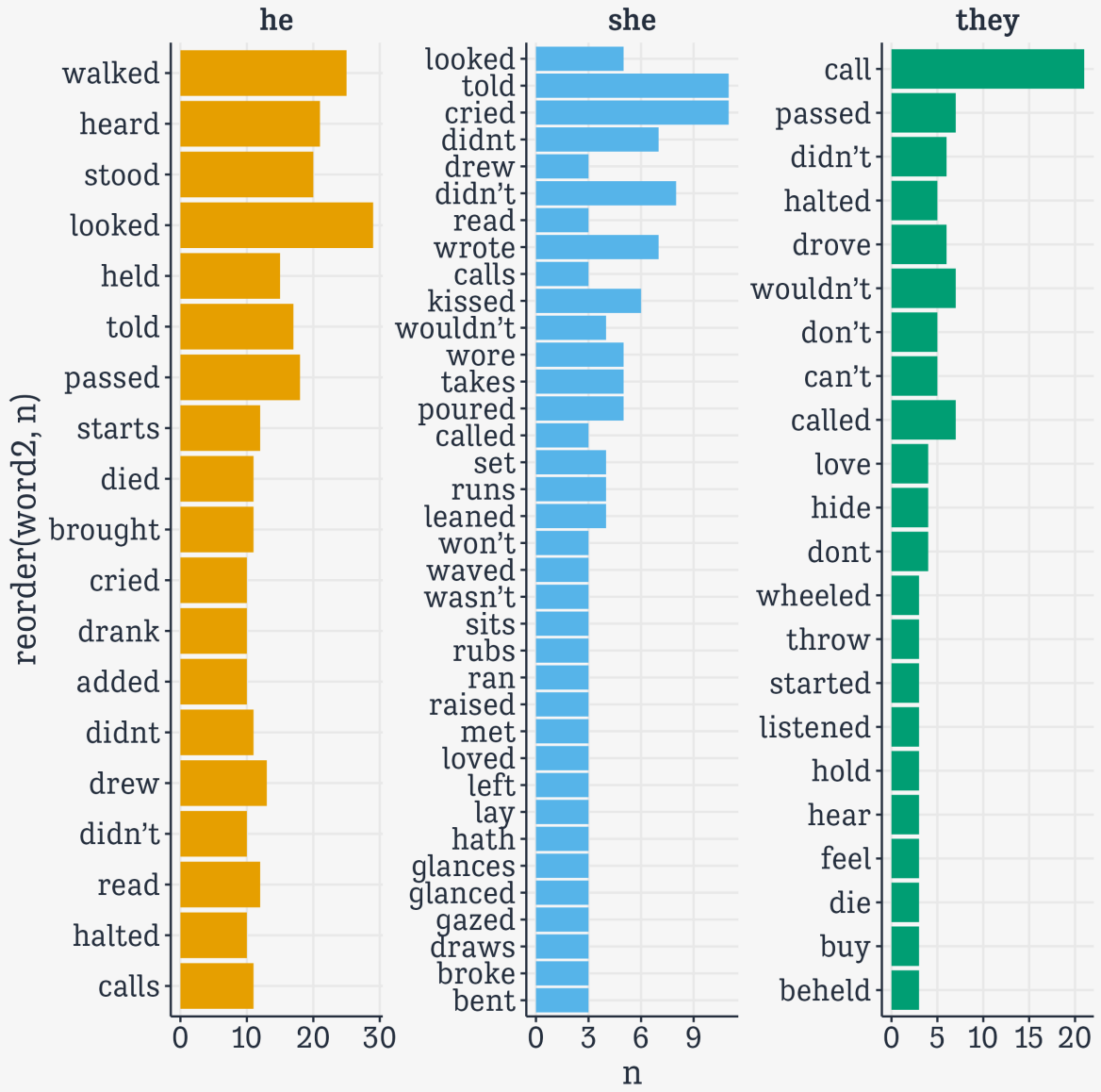
```
pronouns ← raw_text ▷
  unnest_tokens(bigram, text, token = "ngrams", n = 2) ▷
  separate(bigram, c("word1", "word2"), sep=" ") ▷
  filter(word1 %in% c("he", "she", "they")) ▷
  filter(!word2 %in% stop_words$word, !str_detect(word2, "'")) ▷
  count(word1, word2, sort=TRUE)

pronouns
```

```
# A tibble: 1,411 × 3
   word1 word2      n
   <chr> <chr>  <int>
 1 he    looked    29
 2 he    walked    25
 3 he    heard     21
 4 they  call      21
 5 he    stood     20
 6 he    passed    18
 7 he    told      17
 8 he    held      15
 9 he    drew      13
10 he    read      12
# ℹ 1,401 more rows
```

Pronouns

```
out ← pronouns ▷
  group_by(word1) ▷
  top_n(15) ▷
  ggplot(mapping = aes(x=n,
                       y=reorder(word2, n),
                     fill=word1)) +
  geom_col() +
  guides(fill = "none") +
  facet_wrap(~ word1, scales="free")
```

```
print(out)
```


he
she
they
reorder(word2, n)
walked
heard
stood
looked
held
told
passed
starts
died
brought
cried
drank
added
didnt
drew
didn't
read
halted
calls
looked
told
cried
didnt
drew
didn't
read
wrote
calls
kissed
wouldn't
wore
takes
poured
called
set
runs
leaned
won't
waved
wasn't
sits
rubs
ran
raised
met
loved
left
lay
hath
glances
glanced
gazed
draws
broke
bent
call
passed
didn't
halted
drove
wouldn't
don't
can't
called
love
hide
dont
wheeled
throw
started
listened
hold
hear
feel
die
buy
beheld
0
10
20
30
0
3
6
9
0
5
10
15
20
n

Word Embeddings

What more can we do?

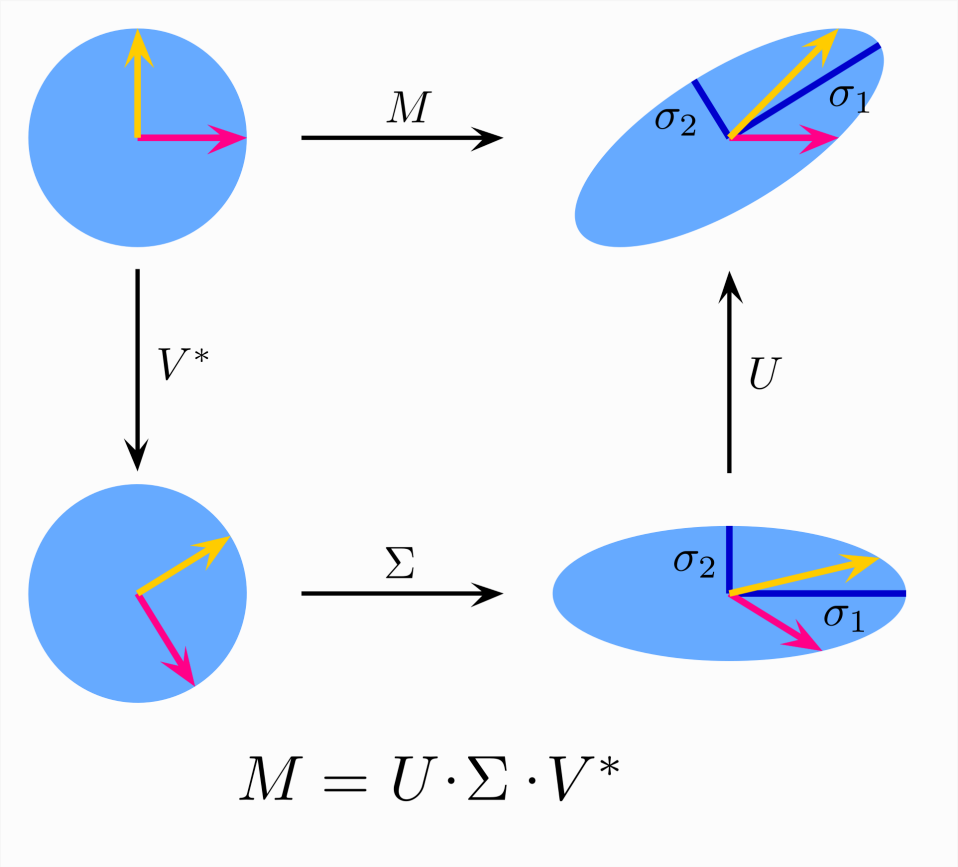
We might like to get a sense of what words mean

One way to think about meaning is by trying to capture context. That is, by trying to quantify the *other* words a particular *focal* word appears with (net of “stopwords”)

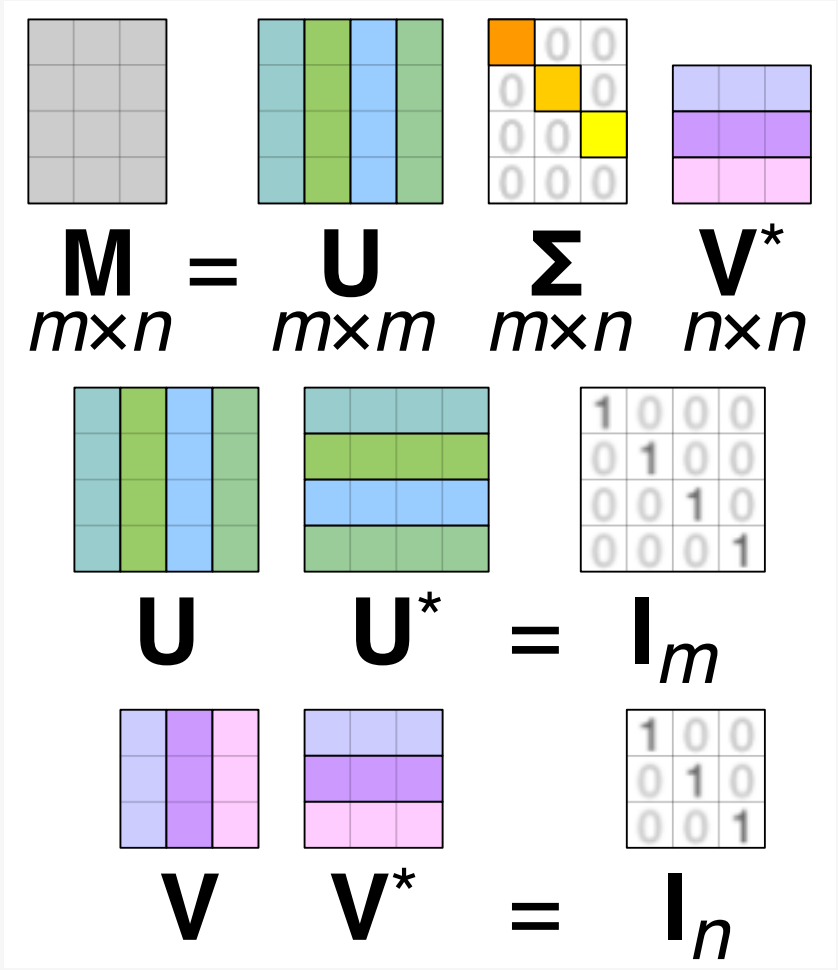
It turns out you can do a lot with this idea, with some linear algebra to help you.

Some Intuition

First, let me back up and give a tiny bit of intuition about the algebra.



Some Intuition



An iconic image

Long Tall Sally

```
# install.packages("imager")
img ← imager::load.image(here("files", "data", "elvis-nixon.jpeg"))
str(img)
```

```
 'cimg' num [1:800, 1:633, 1, 1] 0.914 0.929 0.91 0.906 0.898 ...
```

```
dim(img)
```

```
[1] 800 633   1   1
```

```
## Long
img_df_long ← as.data.frame(img)

head(img_df_long)
```

```
  x y value
1 1 1 0.914
2 2 1 0.929
3 3 1 0.910
4 4 1 0.906
5 5 1 0.898
6 6 1 0.886
```

Return to Sender

```
img_df ← pivot_wider(img_df_long,
                             names_from = y,
                             values_from = value)

dim(img_df)
```

```
[1] 800 634
```

```
img_df[1:5, 1:5]
```

```
# A tibble: 5 × 5
      x   `1`   `2`   `3`   `4`
  <int> <dbl> <dbl> <dbl> <dbl>
1     1 0.914 0.914 0.914 0.910
2     2 0.929 0.929 0.925 0.918
3     3 0.910 0.910 0.902 0.894
4     4 0.906 0.902 0.898 0.894
5     5 0.898 0.894 0.890 0.886
```

Don’t be Cruel

```
tmp ← img_df ▷ select(-x)
dim(tmp)
```

```
[1] 800 633
```

```
tmp[1:5, 1:5]
```

```
# A tibble: 5 × 5
    `1`   `2`   `3`   `4`   `5`
  <dbl> <dbl> <dbl> <dbl> <dbl>
1 0.914 0.914 0.914 0.910 0.902
2 0.929 0.929 0.925 0.918 0.910
3 0.910 0.910 0.902 0.894 0.886
4 0.906 0.902 0.898 0.894 0.890
5 0.898 0.894 0.890 0.886 0.882
```

```
# Scaled and centered
tmp_norm ← scale(tmp, center = TRUE, scale = TRUE)
tmp_norm[1:5, 1:5]
```

```
          1     2     3     4     5
[1,] 0.306 0.327 0.335 0.314 0.266
[2,] 0.417 0.433 0.412 0.365 0.317
[3,] 0.278 0.301 0.258 0.212 0.166
[4,] 0.251 0.248 0.232 0.212 0.191
[5,] 0.195 0.195 0.181 0.161 0.141
```

Don’t be Cruel

Doing the decomposition manually

```
# Decomposition/Factorization into eigenvalues and eigenvectors
cov_eig ← eigen(cov_mat)
names(cov_eig)
```

```
[1] "values"  "vectors"
```

```
# Eigenvalues (variances)
cov_evals ← cov_eig$values
cov_evals[1:5]
```

```
[1] 231.4 120.6  56.9  31.1  22.8
```

```
# Eigenvectors (principal components)
cov_evecs ← cov_eig$vectors
cov_evecs[1:5, 1:5]
```

```
         [,1]    [,2]   [,3]    [,4]   [,5]
[1,] -0.00616 -0.0657 0.0288 -0.0393 0.0601
[2,] -0.00673 -0.0661 0.0286 -0.0385 0.0590
[3,] -0.00715 -0.0659 0.0274 -0.0389 0.0585
[4,] -0.00747 -0.0660 0.0267 -0.0383 0.0594
[5,] -0.00648 -0.0661 0.0279 -0.0399 0.0606
```

Clean up your own Backyard

```
img_pca ← img_df ▷
  select(-x) ▷
  prcomp(scale = TRUE, center = TRUE)

pca_tidy ← broom::tidy(img_pca, matrix = "pcs")

pca_tidy
```

```
# A tibble: 633 × 4
      PC std.dev percent cumulative
   <dbl>   <dbl>   <dbl>      <dbl>
 1     1   15.2   0.366       0.366
 2     2   11.0   0.191       0.556
 3     3    7.54  0.0899      0.646
 4     4    5.57  0.0491      0.695
 5     5    4.78  0.0361      0.731
 6     6    4.56  0.0328      0.764
 7     7    4.06  0.0261      0.790
 8     8    3.66  0.0212      0.811
 9     9    3.37  0.0179      0.829
10    10    3.28  0.0170      0.846
# ℹ 623 more rows
```

Return to Sender

```
names(img_pca)
```

```
[1] "sdev"     "rotation" "center"   "scale"    "x"
```

I Gotta Know

```
reverse_pca ← function(n_comp = 20, pca_object = img_pca){
  ## The pca_object is an object created by base R's prcomp() function.

  ## Multiply the matrix of rotated data by the transpose of the matrix
  ## of eigenvalues (i.e. the component loadings) to get back to a
  ## matrix of original data values
  recon ← pca_object$x[, 1:n_comp] %*% t(pca_object$rotation[, 1:n_comp])

  ## Reverse any scaling and centering that was done by prcomp()

  if(all(pca_object$scale ≠ FALSE)){
    ## Rescale by the reciprocal of the scaling factor, i.e. back to
    ## original range.
    recon ← scale(recon, center = FALSE, scale = 1/pca_object$scale)
  }
  if(all(pca_object$center ≠ FALSE)){
    ## Remove any mean centering by adding the subtracted mean back in
    recon ← scale(recon, scale = FALSE, center = -1 * pca_object$center)
  }

  ## Make it a data frame that we can easily pivot to long format
  ## (because that's the format that the excellent imager library wants
  ## when drawing image plots with ggplot)
  recon_df ← data.frame(cbind(1:nrow(recon), recon))
  colnames(recon_df) ← c("x", 1:(ncol(recon_df)-1))

  ## Return the data to long form
  recon_df_long ← recon_df ▷
    tidyr::pivot_longer(cols = -x,
                        names_to = "y",
                        values_to = "value") ▷
```

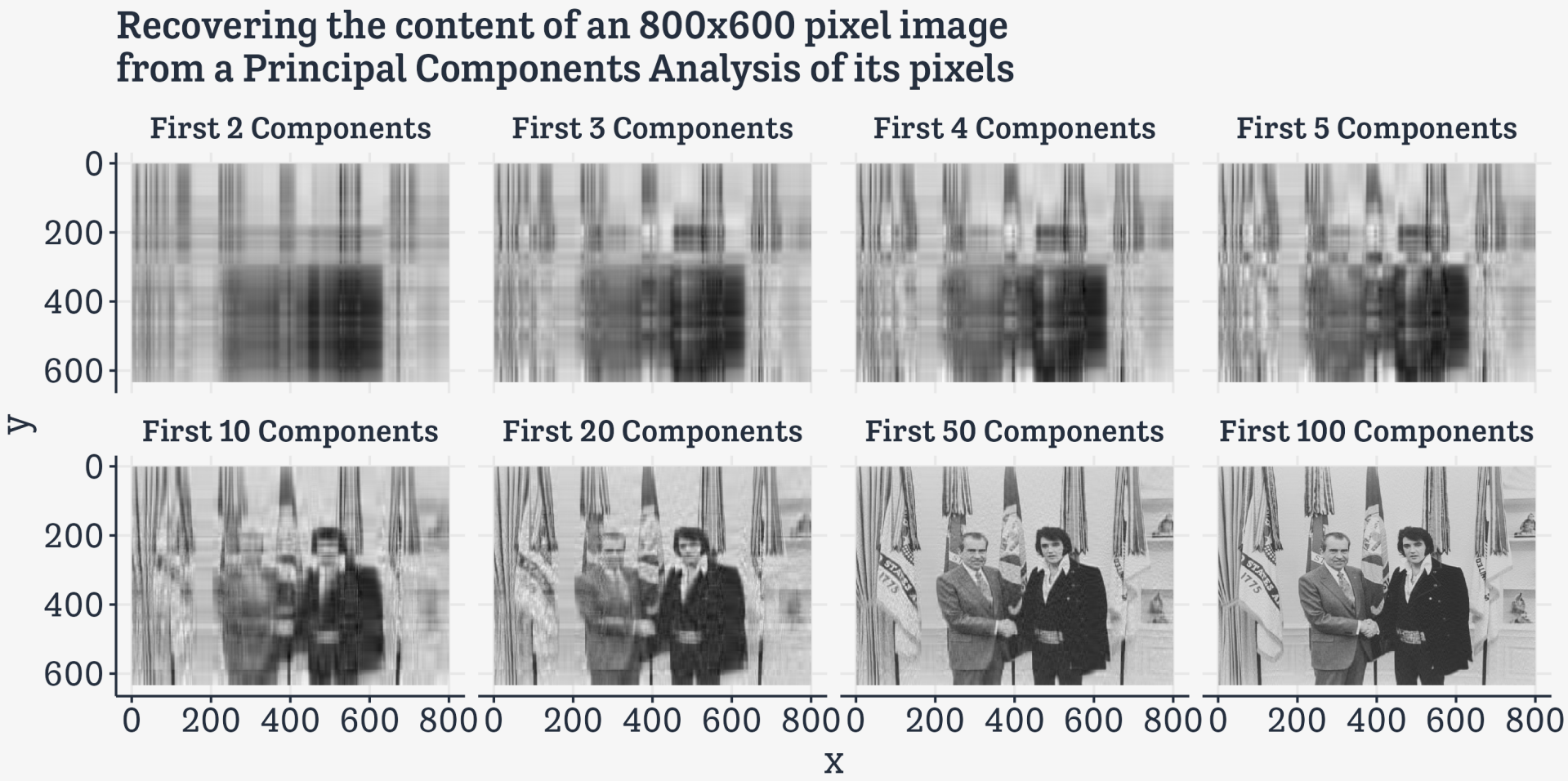
It's Now or Never

```
## The sequence of PCA components we want
n_pcs ← c(2:5, 10, 20, 50, 100)
names(n_pcs) ← paste("First", n_pcs, "Components", sep = "_")

## Map reverse_pca()
recovered_imgs ← map(n_pcs, reverse_pca) ▷
  list_rbind(names_to = "pcs") ▷
  mutate(pcs = str_replace_all(pcs, "_", " "),
         pcs = factor(pcs, levels = unique(pcs), ordered = TRUE))

recovered_imgs
```

```
# A tibble: 4,051,200 × 4
   pcs                   x     y value
   <ord>             <dbl> <dbl> <dbl>
 1 First 2 Components     1     1 0.902
 2 First 2 Components     2     1 0.902
 3 First 2 Components     3     1 0.902
 4 First 2 Components     4     1 0.899
 5 First 2 Components     5     1 0.892
 6 First 2 Components     6     1 0.886
 7 First 2 Components     7     1 0.877
 8 First 2 Components     8     1 0.858
 9 First 2 Components     9     1 0.823
10 First 2 Components    10     1 0.772
# ℹ 4,051,190 more rows
```

Jailhouse Rock

```
p ← ggplot(data = recovered_imgs,
           mapping = aes(x = x, y = y, fill = value))
p_out ← p + geom_raster() +
  scale_y_reverse() +
  scale_fill_gradient(low = "black", high = "white") +
  facet_wrap(~ pcs, ncol = 4) +
  guides(fill = FALSE) +
  labs(title = "Recovering the content of an 800x600 pixel image\nfrom a Principal Components Analy
  theme(strip.text = element_text(face = "bold", size = rel(1.2)),
        plot.title = element_text(size = rel(1.5)))
```


Recovering the content of an 800x600 pixel image
from a Principal Components Analysis of its pixels
First 2 Components
First 3 Components
First 4 Components
First 5 Components
First 10 Components
First 20 Components
First 50 Components
First 100 Components
y
x
0
200
400
600
800

Back to text

```r
# install.packages("text2map")
# remotes::install_gitlab("culturalcartography/text2map.corpora")
# install.packages("quanteda")
# install.packages("textclean")

library(text2map)
library(text2map.corpora)
library(quanteda)

jfk <- tibble(
  text = c("We choose to go to the moon",
           "We choose to go to the moon",
           "We choose to go to the moon in this decade and do other things"))
jfk
```

```
# A tibble: 3 × 1
  text
  <chr>
1 We choose to go to the moon
2 We choose to go to the moon
3 We choose to go to the moon in this decade and do other things
```

A Term Co-Occurence Matrix

```
jfk ▷
  pull(text) ▷
  tokens() ▷
  fcm(context = "window",
      window = 5)
```

```
Feature co-occurrence matrix of: 13 by 13 features.
        features
features We choose to go the moon in this decade and
  We      0      3  6  3   3    0  0    0      0   0
  choose  0      0  6  3   3    3  0    0      0   0
  to      0      0  6  6   6    6  2    1      1   0
  go      0      0  0  0   3    3  1    1      0   0
  the     0      0  0  0   0    3  1    1      1   1
  moon    0      0  0  0   0    0  1    1      1   1
  in      0      0  0  0   0    0  0    1      1   1
  this    0      0  0  0   0    0  0    0      1   1
  decade  0      0  0  0   0    0  0    0      0   1
  and     0      0  0  0   0    0  0    0      0   0
[ reached max_feat  ...  3 more features, reached max_nfeat  ...  3 more features ]
```

We'd like to learn whether any of these co-occurences are *particularly* informative. This is where the linear algebra comes in. We can decompose this matrix of counts and ones like it.

Star Trek TNG Data

```
data("corpus_tng_season5", package = "text2map.corpora")
corpus_tng_season5
```

```
# A tibble: 10,834 × 5
   number title              airdate    character line
    <int> <chr>              <chr>      <chr>     <chr>
 1    101 Redemption, Part 2 1991-09-23 Worf      Aft shields buckling!
 2    101 Redemption, Part 2 1991-09-23 Kurn      Transfer auxiliary power to s…
 3    101 Redemption, Part 2 1991-09-23 Worf      Aft shields are gone!
 4    101 Redemption, Part 2 1991-09-23 Worf      We cannot win. We must withdr…
 5    101 Redemption, Part 2 1991-09-23 Kurn      Keep your place!
 6    101 Redemption, Part 2 1991-09-23 Kurn      New course... three zero seve…
 7    101 Redemption, Part 2 1991-09-23 Helmsman  But sir---!
 8    101 Redemption, Part 2 1991-09-23 Kurn      GhoS!
 9    101 Redemption, Part 2 1991-09-23 Worf      We are entering the star's co…
10    101 Redemption, Part 2 1991-09-23 Kurn      Stand-by to enter warp on my …
# ℹ 10,824 more rows
```

Prepare the text

```
df_trek ← corpus_tng_season5 ▷
  mutate(text = stringi::stri_trans_general(line, "Any-Latin; Latin-ASCII"),
         text = textclean::replace_contraction(text),
         text = tolower(text),
         text = str_replace_all(text, "[[:punct:]]+", " "),
         text = str_replace_all(text, "\\s+", " "))

df_trek ▷
  select(line, text) ▷
  sample_n(10)
```

```
# A tibble: 10 × 2
   line                                                                text
   <chr>                                                               <chr>
 1 Do you know where we can find Orta?                                 "do …
 2 Deanna, I love you, but you do make everything sound like an epitaph.   "dea…
 3 If we have to evacuate, anything is possible...                     "if …
 4 We're down to nine percent. Will, we don't have enough to get back.     "we …
 5 You don't understand. We have a very high success rate in treating dev… "you…
 6 Aye, sir.                                                           "aye…
 7 You've been using this ship to transport a sentient being as property?  "you…
 8 There's nothing here that's unfamiliar.                             "the…
 9 I wrote the orders. I thought she'd be valuable to you.             "i w…
10 Will explode. But if we stay here we're dead, anyway.               "wil…
```

Make a TCM

```
tcm ← df_trek$text ▷
  tokens() ▷
  fcm(context = "window", window = 10) ▷
  as("dgCMatrix")

tcm
```

```
7932 x 7932 sparse Matrix of class "dgCMatrix"

aft             . 5 1 . .  .    .   2  1   1  . . . . .   . . . . . .
shields         . 6 1 2 1  8   21  17  3  14  . . . . .   1 . . . 1 .
buckling        . . . . .  .    .   .  .   .  . . . . .   . . . . . .
transfer        . . . . 1 10   15   3  .   6  . . . . .   . . 1 . . .
auxiliary       . . . . .  1    1   1  .   .  . . . . .   . . . . . .
power           . . . . .  4   73  18  .  21  . . . . 1   2 . 2 1 5 .
to              . . . . .  . 1312 595 13 864 19 3 53 10 34 324 17 34 51 63 18
are             . . . . .  .    . 236 13 604  7 .  9  .  9  86  3 10 15 21  3
gone            . . . . .  .    .   .  .   6  . . . . .   2 . . . . .
we              . . . . .  .    .   .  . 478 13 3 51  6 21  74  2 14 34 37  4
cannot          . . . . .  .    .   .  .   . 4 2  3  2  .   8  2 . . . . .
win             . . . . .  .    .   .  .   .  . .  1  1  .   . . . . . . .
must            . . . . .  .    .   .  .   .  . .  4  1  1  10 . . 1 2 1
withdraw        . . . . .  .    .   .  .   .  . . . . .   1 . . . . .
keep            . . . . .  .    .   .  .   .  . . . . .  10  1 . 1 1 .
your            . . . . .  .    .   .  .   .  . . . . . 104  7 5 5 3 .
place           . . . . .  .    .   .  .   .  . . . . .   . . 1 . . .
```

Weight the matrix

"Pointwise mutual information", or PMI, measures whether events *x* and *y* co-occur more than they independently occur. *PPMI* focuses on the *positive* cases, i.e. of relatedness rather than unrelatedness, assigning zero to "unrelated" words.

```
weight_ppmi ← function(tcm) {
  # correct zero self-occureneces
  diag(tcm) ← diag(tcm) + 1
  # weight by PMI
  tcm ← log(tcm %*% diag(1/diag(tcm)))
  # positive PMI only
  tcm[tcm<0] ← 0
  return(Matrix(tcm, sparse = TRUE))
  }
```

Weight the matrix

We will keep just the words in our matrix with more than 30 occurrences, otherwise doing some upcoming math will take forever.

```
tcm ← tcm[colSums(tcm) > 30, colSums(tcm) > 30]
tcm_ppmi ← weight_ppmi(tcm)
```

How to measure similarity?

Having things in a matrix of this sort lets us compute measures of distance between words. One method, for instance, is cosine similarity:

```
vec1 ← tcm_ppmi["captain",, drop = FALSE] # NB double comma
vec2 ← tcm_ppmi["picard",,  drop = FALSE]
vec3 ← tcm_ppmi["crusher",, drop = FALSE]

text2vec::sim2(vec1, vec2)
```

```
1 x 1 sparse Matrix of class "dgCMatrix"
         features
features  picard
  captain  0.266
```

```
text2vec::sim2(vec1, vec3)
```

```
1 x 1 sparse Matrix of class "dgCMatrix"
         features
features  crusher
  captain  0.0371
```

The magic of SVD

```
# Be patient, it's still big
svd_tng ← svd(tcm_ppmi)

# Get the first 100 cols of the V matrix
wv_tng ← svd_tng$v[,1:100]
rownames(wv_tng) ← rownames(tcm_ppmi)
```

What is this?

We started with a term x term-context matrix with > 3,000 terms and > 3,000 contexts. Now we have …

```
dim(wv_tng)
```

```
[1] 3323  100
```

… a term-by-dimension matrix context matrix with > 3,000 terms and exactly 100 dimensions. (The 100 most “explanatory” ones.) This is a much simpler representation of the data that holds on to as much information from the original as possible.

What is this?

```
wv_tng[1:10,1:5]
```

```
           [,1]     [,2]      [,3]     [,4]      [,5]
to     2.84e-17 7.00e-17 -1.08e-16 2.33e-16 -1.01e-17
are    1.36e-03 3.35e-03 -5.18e-03 1.12e-02 -4.84e-04
we     1.03e-03 2.16e-03 -2.19e-03 8.35e-03  1.08e-03
cannot 3.08e-03 6.15e-03 -1.07e-04 2.66e-02 -7.51e-04
must   6.03e-03 1.17e-02  3.59e-03 5.27e-02 -2.74e-03
keep   9.50e-03 1.70e-02  9.23e-03 8.16e-02  4.73e-03
your   1.67e-03 4.10e-03 -6.34e-03 1.37e-02 -5.92e-04
place  6.28e-03 8.27e-03 -6.52e-03 4.95e-02 -3.93e-04
new    4.25e-03 8.38e-03 -1.20e-03 3.61e-02  9.20e-04
course 1.15e-02 1.67e-02  1.25e-02 9.63e-02  2.98e-03
```

We can do more math on this matrix

The classic example is `(v_king - v_man) + v_woman = v_queen`

Consider:

```
v_picard ← wv_tng["picard",, drop = FALSE]
v_jean ← wv_tng["jean",, drop = FALSE]
v_william ← wv_tng["william",, drop = FALSE]

length(v_picard)
```

```
[1] 100
```

```
v_picard[1:5]
```

```
[1]  0.04086 -0.01899 -0.00285  0.01942  0.01896
```

We can do more math on this matrix

Now, if we create a *synthetic vector*:

```
vec_synth ← (v_picard - v_jean) + v_william

vec_synth[1:5]
```

```
[1]  0.02291 -0.05408  0.00663  0.02758  0.02521
```

And then we ask, hey, find me the word vectors in the matrix that are closest in metric space to that synthetic vector.

```
## This calculates the distance to every word
closest ← text2vec::sim2(wv_tng, vec_synth, method = "cosine")

closest[,1] ▷
  sort(decreasing = TRUE) ▷
  head(n = 5)
```

```
 picard   riker william welcome macduff
  0.750   0.482   0.461   0.401   0.369
```

Something more contentful

```
focal ← c("boy", "girl")
vecs ← wv_tng[focal,]
sims ← text2vec::sim2(vecs, wv_tng, method = "cosine")

df_sims ← tibble(
  word1 = sims[1,],
  word2 = sims[2,],
  term = colnames(sims)
)

df_sims
```

```
# A tibble: 3,323 × 3
      word1    word2 term
      <dbl>    <dbl> <chr>
 1  0.00598 -0.0185  to
 2  0.00598 -0.0185  are
 3 -0.0489  -0.0281  we
 4 -0.0315   0.00160 cannot
 5 -0.0278   0.00905 must
 6 -0.0689  -0.00278 keep
 7  0.00598 -0.0185  your
 8 -0.116   -0.0630  place
 9 -0.0478  -0.00781 new
10 -0.138   -0.0464  course
# ℹ 3,313 more rows
```

Something more contentful

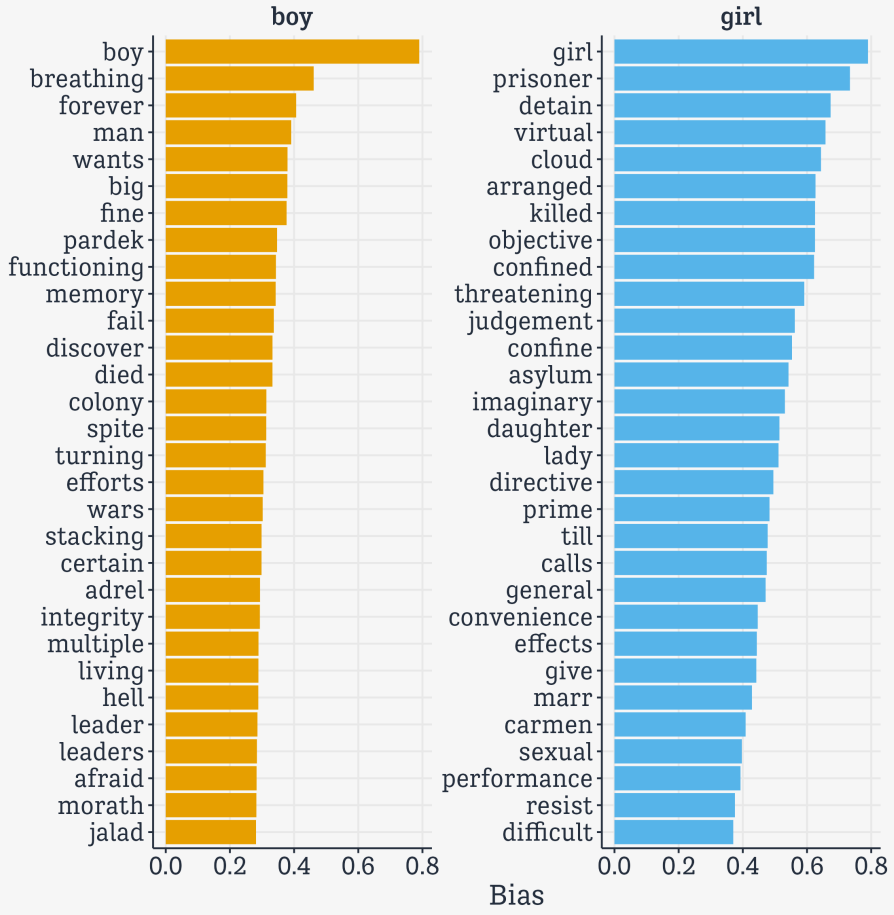
```
df_plot ← df_sims ▷
  mutate(gender_lean = word1 - word2,
         gender = ifelse(gender_lean > 0, "boy", "girl"),
         gender_lean = abs(gender_lean))
```

Gender Bias in Star Trek TNG Dialog

```
out ← df_plot ▷
  group_by(gender) ▷
  slice_max(gender_lean, n = 30) ▷
  mutate(term = fct_reorder(term, gender_lean)) ▷
  ggplot(mapping = aes(gender_lean, term,
                       fill = gender,
                       label = term)) +
  geom_col() +
  facet_wrap(~ gender, scales = "free") +
  guides(fill = "none") +
  labs(x = "Bias", y = "")
```

Gender Bias in Star Trek TNG Dialog

```
print(out)
```



SVD is the most basic method

There are many other, more sophisticated (and more computationally intensive) methods for word-embeddings like this. They form the basis for modern ML methods for Large-Language Models.