Using IPUMS data in R with ipumsr

Derek Burk, Dan Ehrlich, & Kara Fisher

10/12/2021

Who we are

Derek Burk, PhD

Sociology

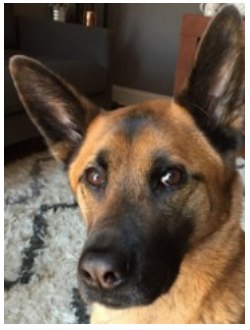
Dan Ehrlich, MA

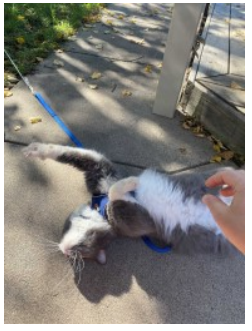
Anthropology

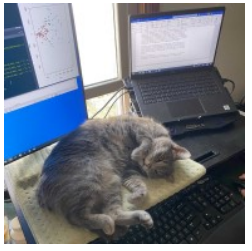
Kara Fisher, MPH

Public Health

Who we are







INSTITUTE FOR
SOCIAL RESEARCH & DATA INNOVATION

IPUMS MPC MN RDC LIFE COURSE CENTER

Overview

1. What is IPUMS?
2. What is ipumsr, and why use it?
3. Reading data into R
4. Exploring and manipulating metadata
5. Brief analysis example
6. Working with IPUMS geographic data
7. Preview of IPUMS API functionality
8. Q & A

Overview

1. What is IPUMS?
2. What is ipumsr, and why use it?
3. Reading data into R
4. Exploring and manipulating metadata
5. Brief analysis example
6. Working with IPUMS geographic data
7. Preview of IPUMS API functionality
8. Q & A

What is IPUMS?

IPUMS is **data**

from censuses and surveys around the world,

integrated across space and time,

thoroughly documented,

and available for free at ipums.org

Poll: Which IPUMS data collections have you used already?

- U.S. Census and American Community Survey **microdata** from 1850 to the present.
- 180,755,919 unique person records from decennial census and American Community Survey.
- 191,983,898 historical person records from full count decennial census from 1850-1940 (1890 census lost due to fire).
- https://usa.ipums.org/usa/

- Current Population Survey **microdata** from 1962 to the present.
- Monthly labor force surveys and supplements.
- https://cps.ipums.org/cps/

- Census **microdata** covering 102 countries from 1960 to the present
- International historic **microdata** from the 19th and early 20th centuries for *9* countries.
- Labor Force surveys provide high resolution **microdata** about work conditions
    - Administered quarterly (usually) with records going back at least 10 years (usually)
    - Currently available for Italy (2011-2020) & Spain (2005-2020)
    - Mexico (2005-2020) coming soon!
- https://international.ipums.org/international/

- Demographic and Health Surveys (DHS) provide integrated **microdata** for analysis across time and space.
    - From the 1980s to the present.
    - Covering Africa and South Asia
- Performance Monitoring for Action (PMA) surveys
    - Focus on fertility, contraception, hygiene, and health
    - Administered frequently to monitor trends in select high-fertility countries.
    - https://ipums.github.io/pma-data-hub/index.html#category:PMA_Publications
- https://globalhealth.ipums.org/

- Health **survey** data from the National Health Interview Survey (NHIS) from the 1960s to the present and the Medical Expenditure Panel Survey (MEPS) from 1996.
- Supplements on cost of healthcare.
- https://healthsurveys.ipums.org/

- Scientists and Engineers Statistical Data System (SESTAT), the leading surveys for studying the science and engineering (STEM) workforce in the United States
- Data from the National Surveys of College Graduates (NSCG), Recent College Graduates (NSRCG) and Doctorate Recipients (SDR) are integrated from 1993 to the present.
- https://highered.ipums.org/highered/

- Historical and contemporary time use data from 1965 to the present.
- Extensive time diary data from respondents in the US and 7 other countries.
- https://timeuse.ipums.org/

- Summary tables and time series of population, housing, agriculture, and economic data
- GIS Shapefiles for all levels of US geography, including tracts, from 1790 to the present
- https://www.nhgis.org/

- **Summary** data tables from population and housing censuses as well as agricultural censuses from around the world
- Integrated GIS **shapefiles.**
- https://ihgis.ipums.org/

Poll: Which IPUMS data collections are you most excited to use in the future ?

So what is IPUMS?

- IPUMS is **a lot** of data
- United in consistently documented metadata

So what is IPUMS?

- IPUMS is **a lot** of data
- United in consistently documented metadata
- *What's the best way to interact with IPUMS data?*

Overview

1. What is IPUMS?
2. What is ipumsr, and why use it?
3. Reading data into R
4. Exploring and manipulating metadata
5. Brief analysis example
6. Working with IPUMS geographic data
7. Preview of IPUMS API functionality
8. Q & A

What is ipumsr?

- R package developed by Greg Freedman Ellis
- Released in 2017
- Over 90,000 CRAN downloads
- Includes functions for
    - Reading IPUMS data
    - Exploring and manipulating IPUMS metadata
    - **SOON**: Interacting with the IPUMS API

Why use ipumsr?

- One package for IPUMS microdata, aggregate data, and geography


One package to rule them all

Why use ipumsr?

- One package for IPUMS microdata, aggregate data, and geography
- Specialized functions for viewing and manipulating IPUMS metadata
- Bundled how-to guides (vignettes)
- Potential to add more features (e.g. API support); let us know what you want!
    - File an issue at https://github.com/mnpopcenter/ipumsr/issues
    - Email ipums+cran@umn.edu

Why use ipumsr?

And finally...

- It's fast!
  - Time to read 3 million rows with 13 variables:

Function	Time (seconds)	With metadata?
data.table::fread()	2.5	No
vroom::vroom()	2.5	No
ipumsr::read_ipums_micro()	3.3	Yes
haven::read_dta()	7.8	Yes
haven::read_sav()	9.4	Yes
readr::read_csv()	9.8	No

Poll: Have you ever used ipumsr before?

Installing ipumsr

```
install.packages("ipumsr")
```

Or if you want the development version

- www.github.com/mnpopcenter/ipumsr

```
if (!require(remotes)) install.packages("remotes")
remotes::install_github("mnpopcenter/ipumsr", ref = "api-alpha-dev")
```

To run the code in this webinar

Install R packages (as needed)

```
# install.packages(ipumsr)

## Tidyverse
install.packages("dplyr")
install.packages("ggplot2")
install.packages("stringr")
install.packages("purrr")

## HTML tables
install.packages("DT")

## gis
install.packages("sf")
```

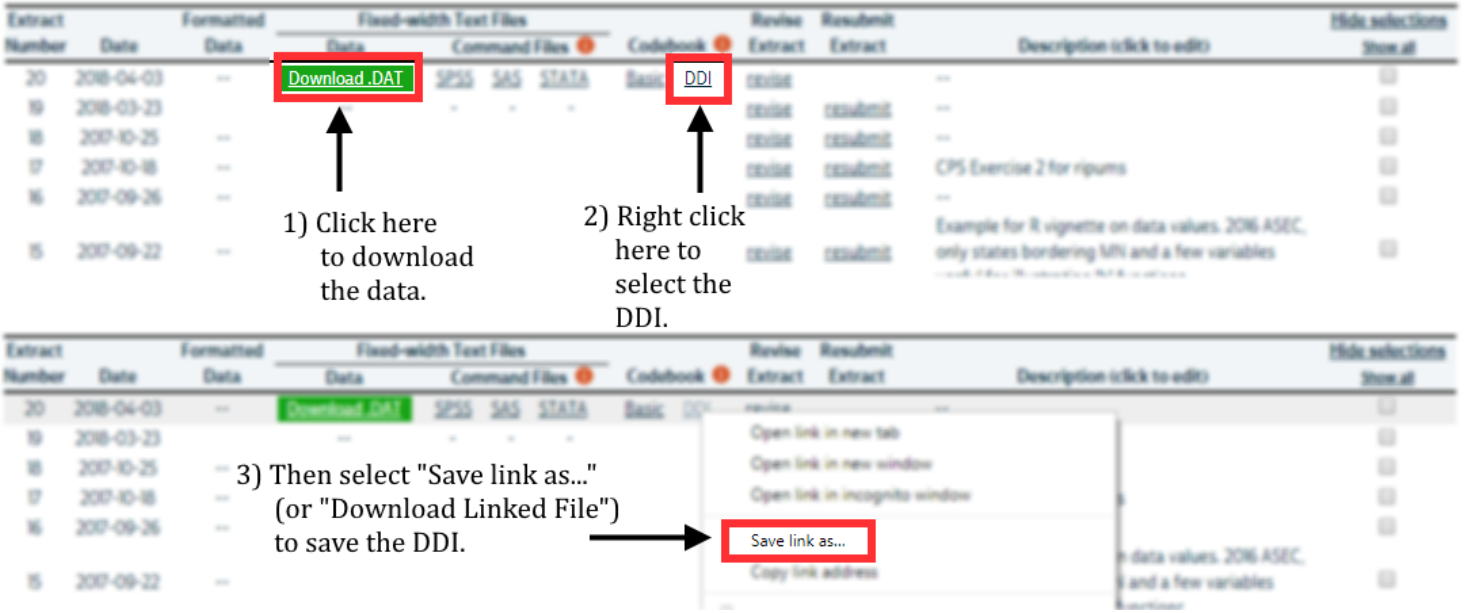
Loading packages

```
library(ipumsr)
library(dplyr)
library(ggplot2)
library(stringr)
library(sf)
library(purrr)
```

Overview

1. What is IPUMS?
2. What is ipumsr, and why use it?
3. Reading data into R
4. Exploring and manipulating metadata
5. Brief analysis example
6. Working with IPUMS geographic data
7. Preview of IPUMS API functionality
8. Q & A

Downloading your data extract



- You must download both the data and DDI codebook
- Save both files in the same folder

Downloading your data extract

- Optional: "R" link contains code to read in your data with ipumsr

Extract Number	Date	Formatted Data	Fixed-width Text Files						Revise Extract	Resubmit Extract	Description (click to edit)
			Data	Command Files				Codebook			
59	2021-09-27	--	Download .DAT	SPSS	SAS	Stata	R	Basic DDI			Family structure Saint Lucia 1980

Read in the data

- Using functions `read_ipums_ddi()` and `read_ipums_micro()`

```
ddi <- read_ipums_ddi("usa_00013.xml")

data <- read_ipums_micro(ddi)
#> Use of data from IPUMS-USA is subject to conditions including that
#> cite the data appropriately. Use command `ipums_conditions()` for
```

- This also works:

```
data <- read_ipums_micro("usa_00013.xml")
#> Use of data from IPUMS-USA is subject to conditions including that
#> cite the data appropriately. Use command `ipums_conditions()` for
```

- Note: supply the codebook, *not* the data file, to `read_ipums_micro()`

Why do I point to the codebook to read in the data?

The data file is the **raw ingredients**, the codebook is the **recipe**

The data file is just raw ingredients

```
196001003364550000001000027   120001000001000005050
196001003364550000001000027   120002000001000006060
196001003364560000001000027   120001000001000006060
196001003364560000001000027   120002000001000006060
196001003364560000001000027   120003000001000002022
196001003364560000001000027   120004000001000000001
196001003364570000000990027   120001000000990006060
196001003364570000000990027   120002000000990004040
196001003364570000000990027   120003000000990004040
196001003364580000001000027   120001000001000006060
```

The DDI codebook is the recipe

```
names(ddi)
#>  [1] "file_name"         "file_path"
#>  [3] "file_type"         "ipums_project"
#>  [5] "extract_date"      "extract_notes"
#>  [7] "rectypes"          "rectype_idvar"
#>  [9] "rectypes_keyvars" "var_info"
#> [11] "conditions"        "citation"
#> [13] "file_encoding"
```

```
ddi$file_name
#> [1] "usa_00013.dat"
```

The DDI codebook is the recipe

```
ddi$var_info
#> # A tibble: 12 x 10
#>    var_name var_label                                    var_desc
#>    <chr>    <chr>                                        <chr>
#>  1 YEAR     Census year                                  "YEAR repor
#>  2 DATANUM  Data set number                              "DATANUM ide
#>  3 SERIAL   Household serial number                      "SERIAL is a
#>  4 HHWT     Household weight                             "HHWT indica
#>  5 STATEFIP State (FIPS code)                            "STATEFIP re
#>  6 CONSPUMA Consistent PUMA, 1980-1990-2000              "CONSPUMA id
#>  7 GQ       Group quarters status                        "GQ classif
#>  8 PHONE    Telephone availability                       "PHONE indic
#>  9 PERNUM   Person number in sample unit                 "PERNUM numb
#> 10 PERWT    Person weight                                "PERWT indic
#> 11 EDUC     Educational attainment [general version]     "EDUC indica
#> 12 EDUCD    Educational attainment [detailed version]    "EDUC indica
```

Overview

1. What is IPUMS?
2. What is ipumsr, and why use it?
3. Reading data into R
4. Exploring and manipulating metadata
5. Brief analysis example
6. Working with IPUMS geographic data
7. Preview of IPUMS API functionality
8. Q & A

What's in my extract again?

Maybe I wrote an informative extract description?

```
ddi$extract_notes %>% strwrap(60)
#> [1] "User-provided description: Revision of(Revision of(Revision"
#> [2] "of(Revision of(my extract)))"
```

No such luck 😞

What's in my extract again?

We can print the names of our variables:

```
names(data)
#>  [1] "YEAR"     "DATANUM"  "SERIAL"   "HHWT"
#>  [5] "STATEFIP" "CONSPUMA" "GQ"       "PHONE"
#>  [9] "PERNUM"   "PERWT"    "EDUC"     "EDUCD"
```

But often variable names aren't self-explanatory.

Let's leverage that attached metadata!

Available metadata

Variable labels and descriptions:

```
ipums_var_label(ddi, PHONE)
#> [1] "Telephone availability"

ipums_var_desc(ddi, PHONE) %>% strwrap(60)
#> [1] "PHONE indicates whether residents of the housing unit had"
#> [2] "telephone access."
```

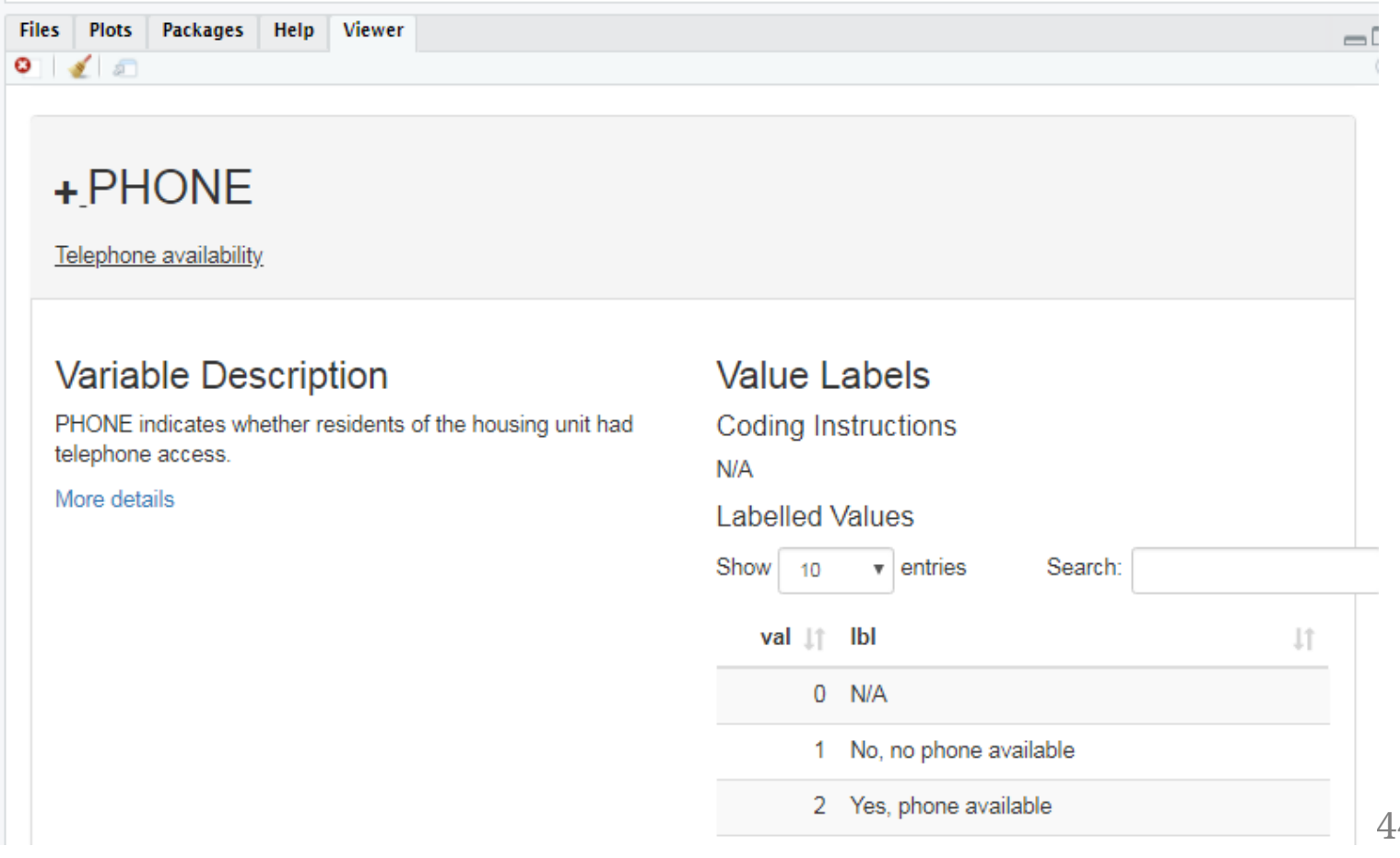
Available metadata

Value labels:

```
ipums_val_labels(ddi, PHONE)
#> # A tibble: 4 x 2
#>     val lbl
#>   <dbl> <chr>
#> 1     0 N/A
#> 2     1 No, no phone available
#> 3     2 Yes, phone available
#> 4     8 Suppressed (2012 and 2015 ACS)
```

An interactive view of metadata

```
ipums_view(ddi)
```



Wrangling value labels

- IPUMS value labels don't translate perfectly to R factors
    - Every value in a factor must be labeled
    - Factor values always count up from 1
- `ipumsr` uses `haven::labelled()` objects to preserve values and labels, but these objects can be tricky to work with
- ipumsr helper functions allow you to leverage info from values and labels

haven::labelled columns at a glance

```
data
#> # A tibble: 1,476,443 x 12
#>     YEAR DATANUM SERIAL  HHWT       STATEFIP CONSPUMA
#>    <int>   <dbl>  <dbl> <dbl>      <int+lbl>    <dbl>
#>  1  1960       1 336455   100 27 [Minnesota]       NA 1 [Household
#>  2  1960       1 336455   100 27 [Minnesota]       NA 1 [Household
#>  3  1960       1 336456   100 27 [Minnesota]       NA 1 [Household
#>  4  1960       1 336456   100 27 [Minnesota]       NA 1 [Household
#>  5  1960       1 336456   100 27 [Minnesota]       NA 1 [Household
#>  6  1960       1 336456   100 27 [Minnesota]       NA 1 [Household
#>  7  1960       1 336457    99 27 [Minnesota]       NA 1 [Household
#>  8  1960       1 336457    99 27 [Minnesota]       NA 1 [Household
#>  9  1960       1 336457    99 27 [Minnesota]       NA 1 [Household
#> 10  1960       1 336458   100 27 [Minnesota]       NA 1 [Household
#> # ... with 1,476,433 more rows
```

Get rid of haven::labelled columns

```
as_factor(data)
#> # A tibble: 1,476,443 x 12
#>     YEAR DATANUM SERIAL  HHWT STATEFIP  CONSPUMA GQ
#>    <int>   <dbl>  <dbl> <dbl> <fct>        <dbl> <fct>
#>  1  1960       1 336455   100 Minnesota       NA Households under
#>  2  1960       1 336455   100 Minnesota       NA Households under
#>  3  1960       1 336456   100 Minnesota       NA Households under
#>  4  1960       1 336456   100 Minnesota       NA Households under
#>  5  1960       1 336456   100 Minnesota       NA Households under
#>  6  1960       1 336456   100 Minnesota       NA Households under
#>  7  1960       1 336457    99 Minnesota       NA Households under
#>  8  1960       1 336457    99 Minnesota       NA Households under
#>  9  1960       1 336457    99 Minnesota       NA Households under
#> 10  1960       1 336458   100 Minnesota       NA Households under
#> # ... with 1,476,433 more rows
```

Get rid of haven::labelled columns

```
zap_labels(data)
#> # A tibble: 1,476,443 x 12
#>     YEAR DATANUM SERIAL  HHWT STATEFIP CONSPUMA    GQ PHONE PERNUM
#>    <int>   <dbl>  <dbl> <dbl>    <int>    <dbl> <int> <int>  <dbl>
#>  1  1960       1 336455   100       27       NA     1     2      1
#>  2  1960       1 336455   100       27       NA     1     2      2
#>  3  1960       1 336456   100       27       NA     1     2      1
#>  4  1960       1 336456   100       27       NA     1     2      2
#>  5  1960       1 336456   100       27       NA     1     2      3
#>  6  1960       1 336456   100       27       NA     1     2      4
#>  7  1960       1 336457    99       27       NA     1     2      1
#>  8  1960       1 336457    99       27       NA     1     2      2
#>  9  1960       1 336457    99       27       NA     1     2      3
#> 10  1960       1 336458   100       27       NA     1     2      1
#> # ... with 1,476,433 more rows
```

Using ipumsr label helper functions

lbl_na_if()

- `lbl_na_if()` allows you to set certain values or labels to missing

```
ipums_val_labels(data$PHONE)
#> # A tibble: 4 x 2
#>     val lbl
#>   <int> <chr>
#> 1     0 N/A
#> 2     1 No, no phone available
#> 3     2 Yes, phone available
#> 4     8 Suppressed (2012 and 2015 ACS)
```

lbl_na_if()

```
data$PHONE2 <- lbl_na_if(data$PHONE, ~.val %in% c(0, 8)) %>%
  as_factor()
```

...

before ([val] label)	after	count
[0] N/A		41133
[1] No, no phone available	No, no phone available	30852
[2] Yes, phone available	Yes, phone available	1404458
[8] Suppressed (2012 and 2015 ACS)		0

lbl_na_if()

```
data$PHONE2 <- lbl_na_if(
  data$PHONE,
  function(.val, .lbl) .val %in% c(0, 8)
) %>%
  as_factor()
```

...

before ([val] label)	after	count
[0] N/A		41133
[1] No, no phone available	No, no phone available	30852
[2] Yes, phone available	Yes, phone available	1404458
[8] Suppressed (2012 and 2015 ACS)		0

lbl_na_if()

- Works with both values (`.val`) and labels (`.lbl`)

```
drop_labels <- c("N/A", "Suppressed (2012 and 2015 ACS)")

data$PHONE3 <- lbl_na_if(data$PHONE, ~.lbl %in% drop_labels) %>%
  as_factor()
```

lbl_collapse()

- `lbl_collapse()` allows you to take advantage of the hierarchical structure of value labels

```
ipums_val_labels(data$EDUCD)
#> # A tibble: 44 x 2
#>      val lbl
#>    <int> <chr>
#>  1     0 N/A or no schooling
#>  2     1 N/A
#>  3     2 No schooling completed
#>  4    10 Nursery school to grade 4
#>  5    11 Nursery school, preschool
#>  6    12 Kindergarten
#>  7    13 Grade 1, 2, 3, or 4
#>  8    14 Grade 1
#>  9    15 Grade 2
#> 10    16 Grade 3
#> # ... with 34 more rows
```

lbl_collapse()

- Maybe this is too much detail, so we want to collapse the last digit

```
data$EDUCD2 <- lbl_collapse(data$EDUCD, ~.val %/% 10) %>%
  as_factor(ordered = TRUE)
```

. . .

before ([val] label)	after	count
[0] N/A or no schooling	N/A or no schooling	11811
[1] N/A	N/A or no schooling	50098
[2] No schooling completed	N/A or no schooling	50114
[10] Nursery school to grade 4	Nursery school to grade 4	40324

Still too detailed for a graph

```
data$EDUCD %>%
  lbl_collapse(~.val %/% 10) %>%
  ipums_val_labels()
#> # A tibble: 13 x 2
#>      val lbl
#>    <dbl> <chr>
#>  1     0 N/A or no schooling
#>  2     1 Nursery school to grade 4
#>  3     2 Grade 5, 6, 7, or 8
#>  4     3 Grade 9
#>  5     4 Grade 10
#>  6     5 Grade 11
#>  7     6 Grade 12
#>  8     7 1 year of college
#>  9     8 2 years of college
#> 10     9 3 years of college
#> 11    10 4 years of college
#> 12    11 5+ years of college
#> 13    99 Missing
```

lbl_relabel()

- Maybe the education variable is still too specific.

```
college_regex <- "^[123] year(s)? of college$"
data$EDUCD3 <- data$EDUCD %>%
  lbl_collapse(~.val %/% 10) %>%
  lbl_relabel(
    lbl(2, "Less than High School") ~.val > 0 & .val < 6,
    lbl(3, "High school") ~.lbl == "Grade 12",
    lbl(4, "Some college") ~str_detect(.lbl, college_regex),
    lbl(5, "College or more") ~.val %in% c(10, 11)
  ) %>%
  as_factor()
```

lbl_relabel()

- Maybe the education variable is still too specific.

```
college_regex <- "^[123] year(s)? of college$"
data$EDUCD3 <- data$EDUCD %>%
  lbl_collapse(~.val %/% 10) %>%
  lbl_relabel(
    lbl(2, "Less than High School") ~.val > 0 & .val < 6,
    lbl(3, "High school") ~.lbl == "Grade 12",
    lbl(4, "Some college") ~str_detect(.lbl, college_regex),
    lbl(5, "College or more") ~.val %in% c(10, 11)
  ) %>%
  as_factor()
```

lbl_relabel()

- Maybe the education variable is still too specific.

```
college_regex <- "^[123] year(s)? of college$"
data$EDUCD3 <- data$EDUCD %>%
  lbl_collapse(~.val %/% 10) %>%
  lbl_relabel(
    lbl(2, "Less than High School") ~.val > 0 & .val < 6,
    lbl(3, "High school") ~.lbl == "Grade 12",
    lbl(4, "Some college") ~str_detect(.lbl, college_regex),
    lbl(5, "College or more") ~.val %in% c(10, 11)
  ) %>%
  as_factor()
```

lbl_relabel()

```
levels(data$EDUCD3)
#> [1] "N/A or no schooling"
#> [2] "Less than High School"
#> [3] "High school"
#> [4] "Some college"
#> [5] "College or more"
#> [6] "Missing"
```

Overview

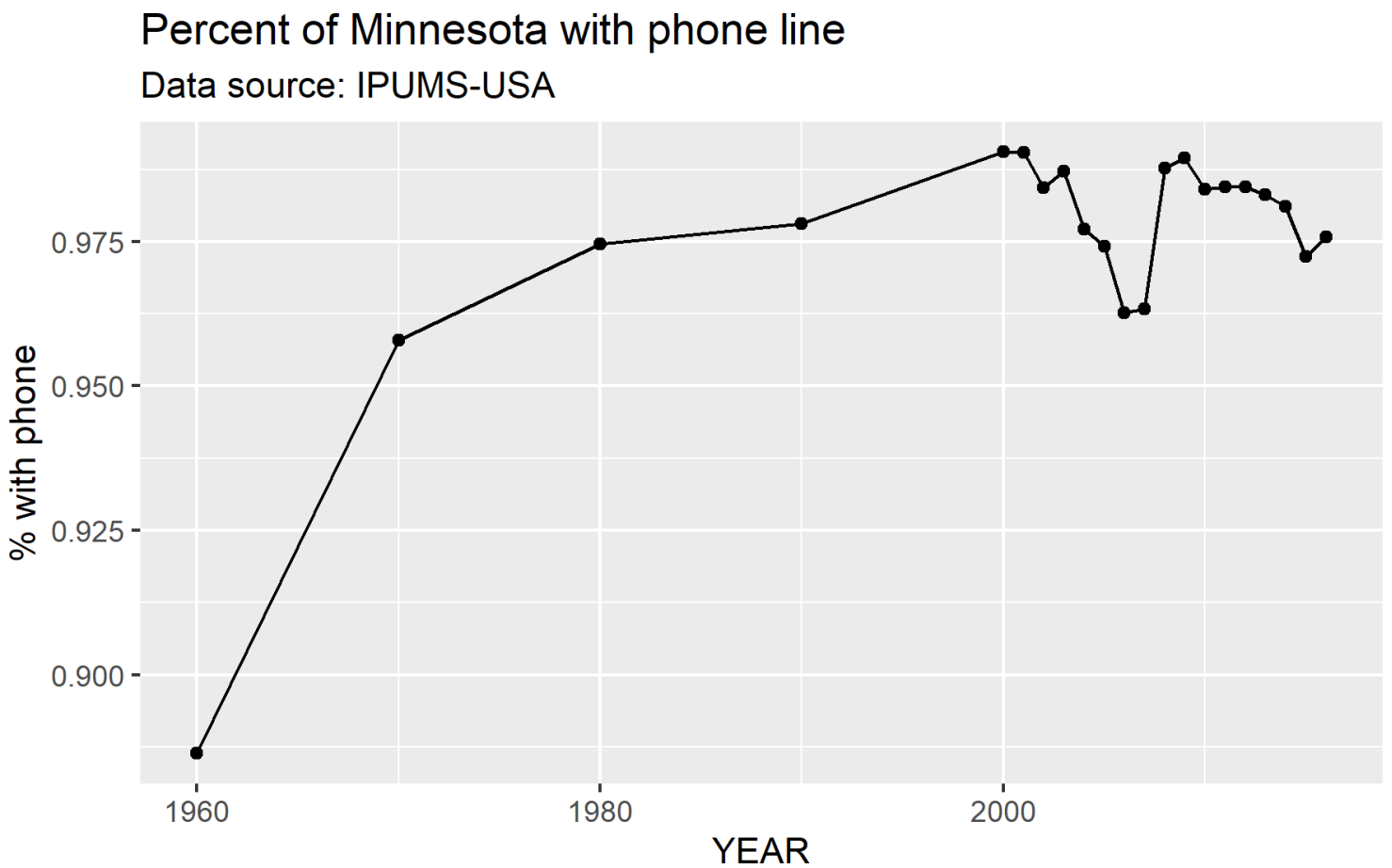
1. What is IPUMS?
2. What is ipumsr, and why use it?
3. Reading data into R
4. Exploring and manipulating metadata
5. Brief analysis example
6. Working with IPUMS geographic data
7. Preview of IPUMS API functionality
8. Q & A

Phone availability

- Now that they're factors, ready for use as regular R data

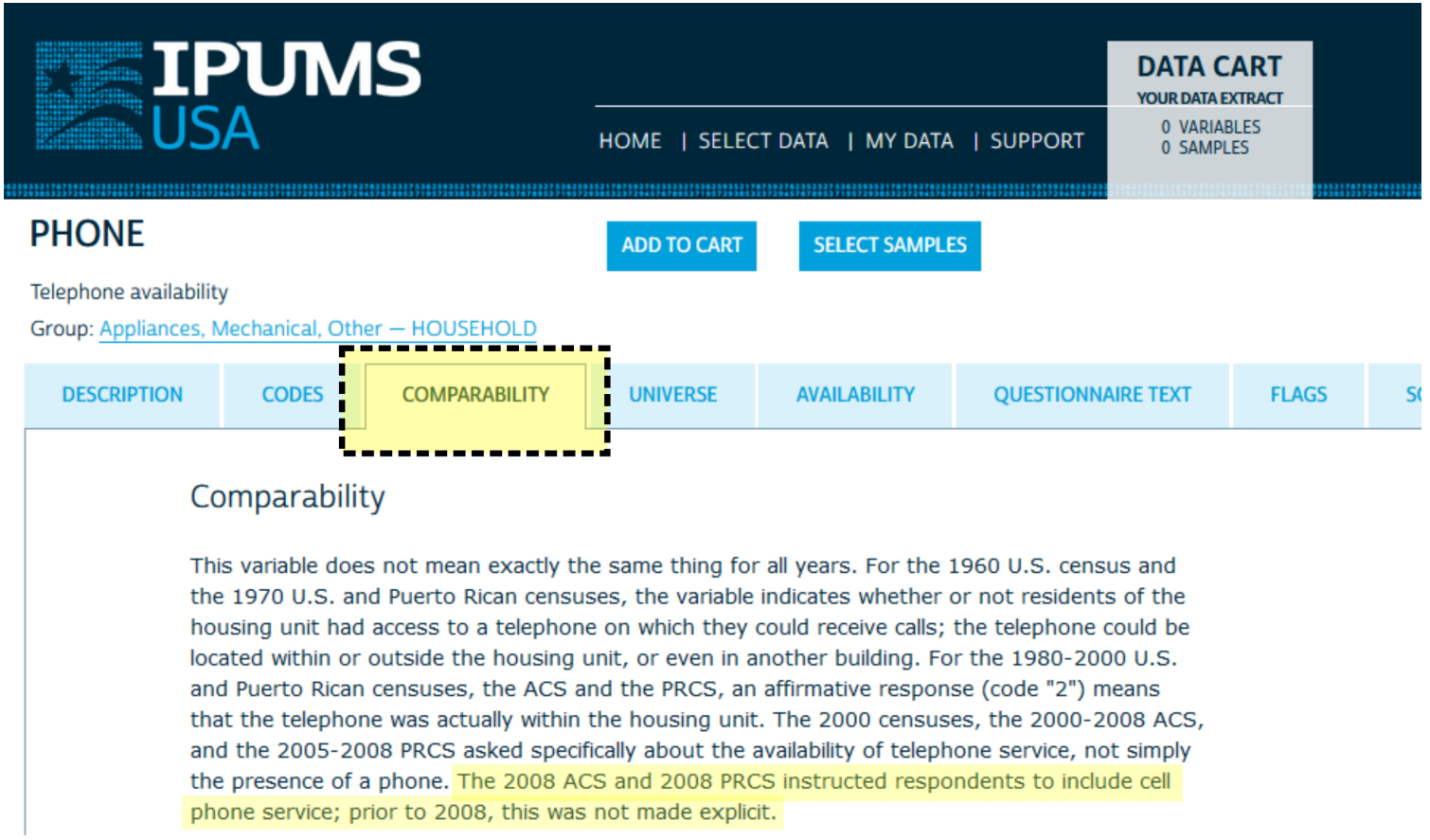
```
graph_data <- data %>%
  group_by(YEAR) %>%
  summarize(
    `% with phone` = weighted.mean(
      PHONE2 == "Yes, phone available", PERWT, na.rm = TRUE
    ),
    .groups = "drop"
  )

ggplot(graph_data, aes(x = YEAR, y = `% with phone`)) +
  geom_point() +
  geom_line() +
  labs(
    title = "Percent of Minnesota with phone line",
    subtitle = paste0("Data source: ", ddi$ipums_project),
    caption = paste(
      strwrap(ipums_var_desc(ddi, PHONE), 90),
      collapse = "\n"
    )
  )
```

Phone availability

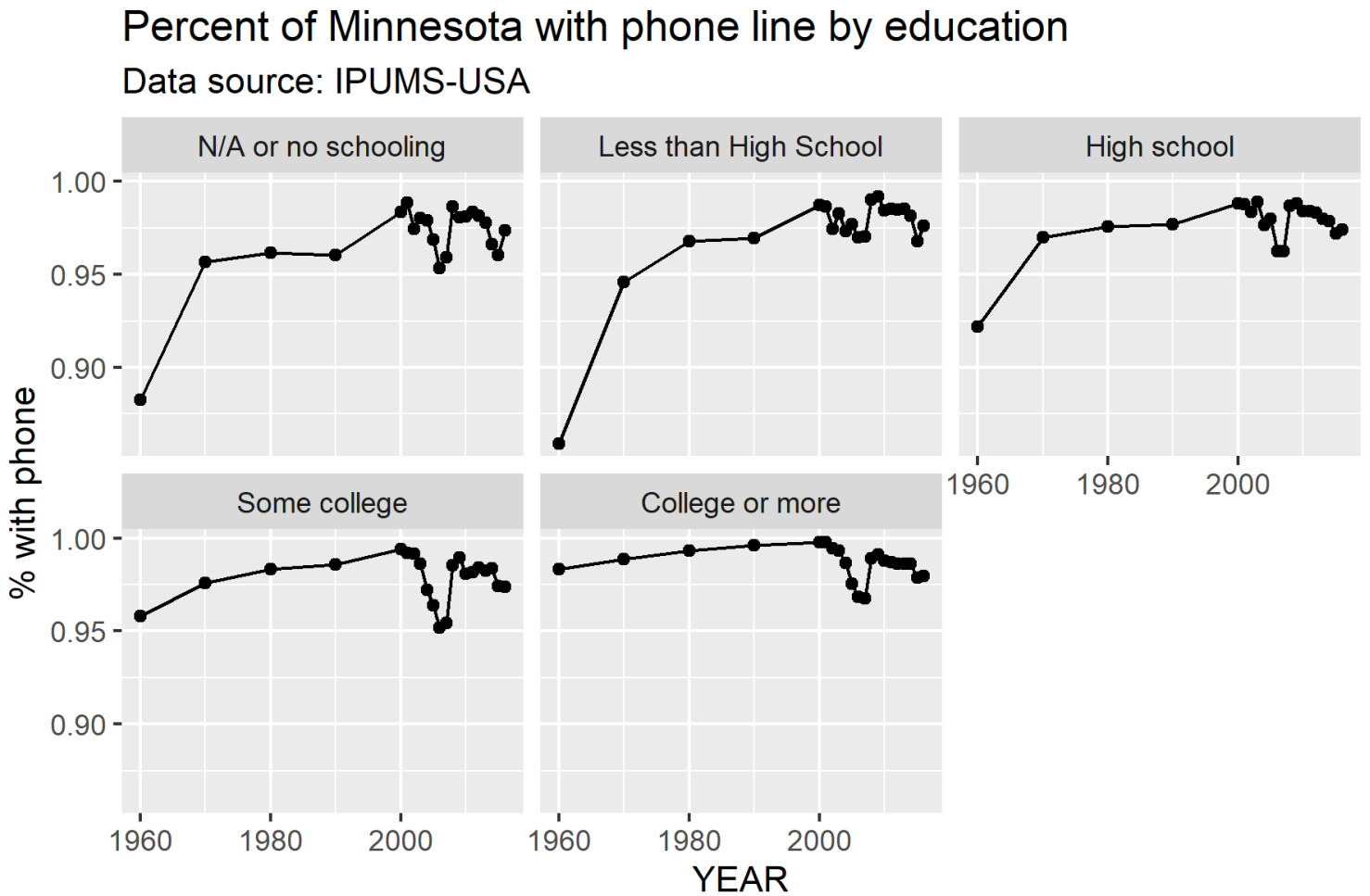


PHONE indicates whether residents of the housing unit had telephone access.

Interpretation



Phone availability by education



Overview

1. What is IPUMS?
2. What is ipumsr, and why use it?
3. Reading data into R
4. Exploring and manipulating metadata
5. Brief analysis example
6. Working with IPUMS geographic data
7. Preview of IPUMS API functionality
8. Q & A

Getting geographic data

- For IPUMS USA (and several other projects), we provide geographic boundaries as well. For many areas, this includes harmonizing boundary changes over time.
- Our extract includes the variable CONSPUMA, for "Consistent Public Use Microdata Areas"
- Note: CONSPUMA units are large
    - For finer geographic detail, check out IPUMS NHGIS

Getting geographic data

Loading shape data

- `ipumsr` provides support for both sf and sp data; we'll use sf here
- Load with the `ipums_read_sf()` function (mostly just a wrapper around `sf::read_sf()`)

```
shape_data <- read_ipums_sf("shape/")
#> options:        ENCODING=latin1
#> Reading layer `ipums_conspuma' from data source
#>   `C:\Users\derek\Documents\ipumsr-webinar\shape\ipums_conspuma.sh
#>   using driver `ESRI Shapefile'
#> Simple feature collection with 543 features and 3 fields
#> Geometry type: MULTIPOLYGON
#> Dimension:     XY
#> Bounding box:  xmin: -7115713 ymin: -1337508 xmax: 2258225 ymax: 4
#> Projected CRS: USA_Contiguous_Albers_Equal_Area_Conic
```

Loading shape data

```
as_tibble(shape_data)
#> # A tibble: 543 x 4
#>    CONSPUMA STATEFIP State
#>       <dbl> <chr>    <chr>
#>  1      540 02       Alaska     (((-3043869 3470021, -3043834 346
#>  2      541 02       Alaska     (((-2291901 2327688, -2290772 232
#>  3      542 15       Hawaii     (((-6021813 59835.48, -6021818 59
#>  4      491 51       Virginia   (((1720696 104240.2, 1720648 1041
#>  5      492 51       Virginia   (((1745400 116778.3, 1745641 1167
#>  6      493 51       Virginia   (((1728178 119921.5, 1728243 1198
#>  7      494 51       Virginia   (((1717473 129180.9, 1717834 1289
#>  8      495 53       Washington (((-1584859 1462223, -1583276 146
#>  9      496 53       Washington (((-2031282 1366306, -2031348 136
#> 10      497 53       Washington (((-1970094 1408574, -1970093 140
#> # ... with 533 more rows
```

Joining shape data

- `ipumsr` has helpers for merging data that work with both sf and sp structures

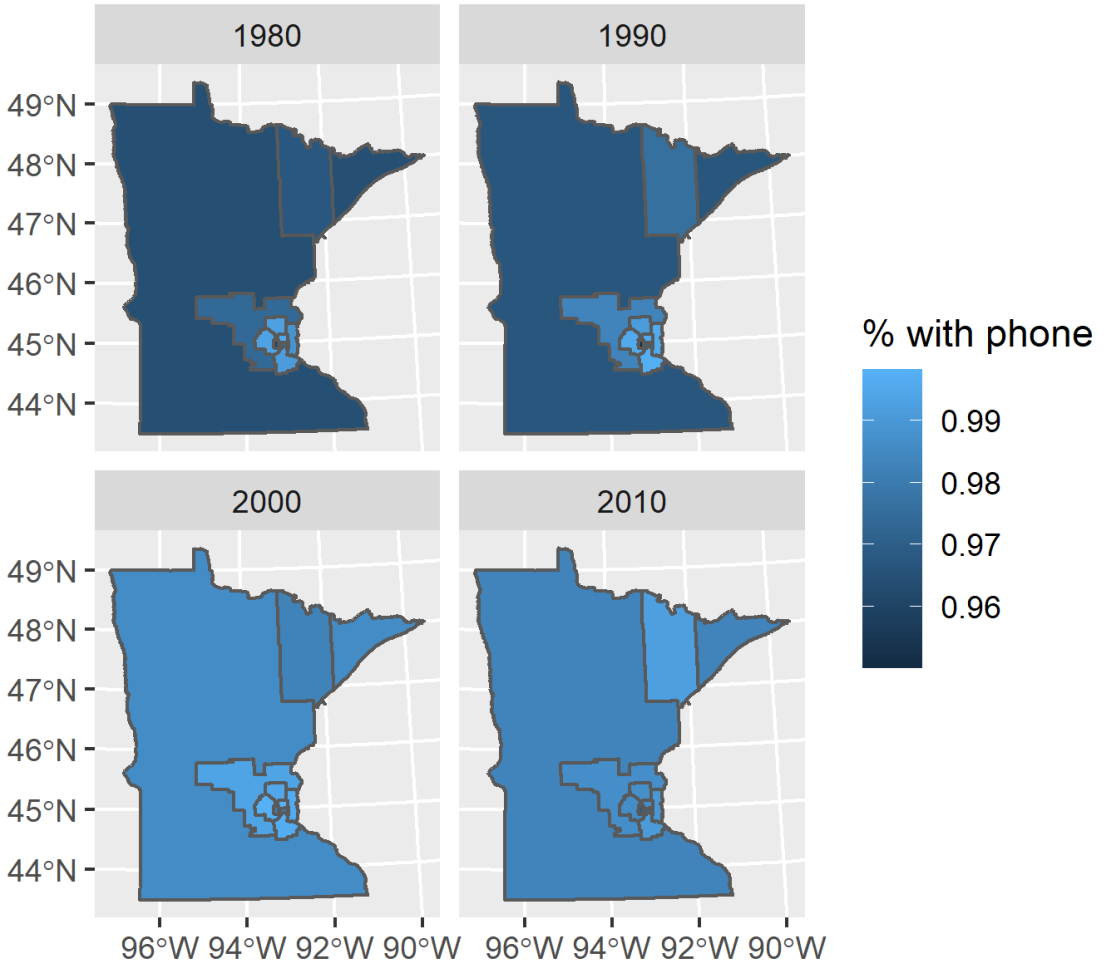
```
conspuma_data <- data %>%
  group_by(CONSPUMA, YEAR) %>%
  summarize(
    `% with phone` = weighted.mean(
      PHONE2 == "Yes, phone available", PERWT, na.rm = TRUE
    ),
    .groups = "drop"
  )

conspuma_data <- ipums_shape_inner_join(
  conspuma_data,
  shape_data,
  by = "CONSPUMA"
)
#> Some observations were lost in the join (533 observations in the s
#> 11 obervation in data). See `join_failures(...)` for more details.
```

Plotting shape data

- Since the addition of `geom_sf()`, ggplot2 can plot sf data:

```
graph_data <- conspuma_data %>%
  filter(YEAR %in% c(1980, 1990, 2000, 2010))

ggplot(graph_data, aes(fill = `% with phone`)) +
  facet_wrap(~YEAR) +
  geom_sf()
```

Plotting shape data



Overview

1. What is IPUMS?
2. What is ipumsr, and why use it?
3. Reading data into R
4. Exploring and manipulating metadata
5. Brief analysis example
6. Working with IPUMS geographic data
7. Preview of IPUMS API functionality
8. Q & A

API Timeline

- Currently in internal testing
- Beta testing before the end of 2021
- IPUMS USA public launch early 2022

What can I do with the API?

- Define and submit extract requests
- Check extract status or "wait" for an extract to finish
- Download completed extracts
- Get info on past extracts
- Share extract definitions

What can't I do with the API?

- Bypass the extract system entirely
- Explore what data are available
- Use all features of the extract system (at least not right away)

Pipe-friendly example

```
my_extract <- define_extract(
  "usa",
  "Occupation by sex and age",
  c("us2017a", "us2018a"),
  c("SEX", "AGE", "IND", "OCC")
)
```

Extract definition to data in one piped expression!

```
data <- my_extract %>%
  submit_extract() %>%
  wait_for_extract() %>%
  download_extract() %>%
  read_ipums_micro()
```

Pipe-friendly example

```
my_extract <- define_extract(
  "usa",
  "Occupation by sex and age",
  c("us2017a", "us2018a"),
  c("SEX", "AGE", "IND", "OCC")
)
```

Extract definition to data in one piped expression!

```
data <- my_extract |>
  submit_extract() |>
  wait_for_extract() |>
  download_extract() |>
  read_ipums_micro()
```

Pipe-friendly example

```
my_extract %>%
  submit_extract() %>%
  wait_for_extract() %>%
  download_extract()
```

```
#> Successfully submitted IPUMS USA extract number 58
#> Checking extract status...
#> Waiting 10 seconds...
#> Checking extract status...
#> Waiting 20 seconds...
#> Checking extract status...
#> Waiting 40 seconds...
#> Checking extract status...
#> Waiting 80 seconds...
#> Checking extract status...
#> Extract ready to download
#> DDI codebook file saved to usa_00058.xml
#> Data file saved to usa_00058.dat.gz
```

Revise and resubmit

Get definition of my most recent extract:

```
old_extract <- get_recent_extracts_info_list("usa", 1)[[1]]
```

Or if we know the number of the extract:

```
old_extract <- get_extract_info("usa:33")
```

Revise and resubmit

Then add a variable and resubmit:

```
old_extract %>%
  revise_extract(vars_to_add = "EDUC") %>%
  submit_extract()
```

Share your extract definition

```
save_extract_as_json(my_extract, "my_extract.json")
```

Another user can read that definition back in with:

```
cloned_extract <- define_extract_from_json("my_extract.json", "usa")
```

Overview

1. What is IPUMS?
2. What is ipumsr, and why use it?
3. Reading data into R
4. Exploring and manipulating metadata
5. Brief analysis example
6. Working with IPUMS geographic
7. Preview of IPUMS API functionality
8. Q & A

Resources

- Email us: ipums+cran@umn.edu
- Post on the IPUMS User Forum: https://forum.ipums.org/
- Create an issue on GitHub: https://github.com/mnpopcenter/ipumsr
- This presentation: https://github.com/dtburk/ipumsr-webinar
- ipumsr website, with vignettes: http://tech.popdata.org/ipumsr/index.html
- IPUMS tutorials page: https://www.ipums.org/support/tutorials
- IPUMS NHGIS API documentation: https://developer.ipums.org/docs/workflows/
- *Geocomputation with R* book: https://geocompr.robinlovelace.net/