MEPS tutorials

Mark Bounthavong

2023-07-22

Contents

About

This is a collection of tutorials that use data from the Agency for Healthcare Research and Quality (AHRQ) Medical Expenditure Panel Survey (MEPS).

These tutorials use R and RStudio for data loading, manipulation, analysis, and presentation.

Chapter 1 provides an introduction on how to load and import MEPS data into R.

Chapter 2 is an instruction on how to merge various MEPS data files.

Chapter 3 focuses on applying weights to the population.

Further chapters are forthcoming.

Chapter 1

Loading MEPS data into R

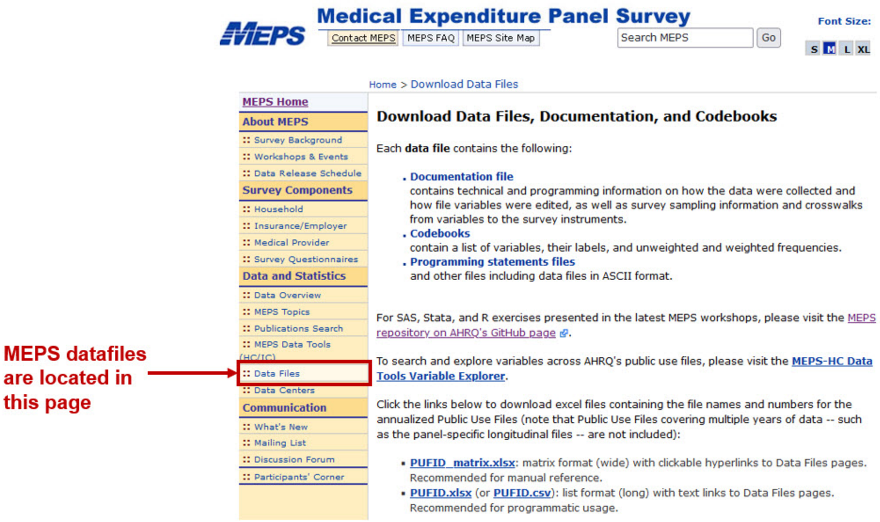
1.1 Introduction

The Agency for Healthcare Research and Quality (AHRQ) Medical Expenditure Panel Survey (MEPS) is a set of data on U.S. households about their healthcare expenditures. It includes data on the individual / household demographics, socioeconomic status, insurance coverage, and healthcare expenditures. Healthcare expenditures include data on health-related spending, medical conditions, prescriptions, and utilization (e.g., number of office-based visits). MEPS draws upon a nationally representative subsample from the National Health Interview Survey, which is conducted by the National Center for Health Statistics. Hence, MEPS provides researchers with the ability to generate estimates for the representative U.S. population.

1.2 MEPS Data

MEPS data are located on their website in their data files page. You can find data from 1996 to the most recent available year (during the writing of this tutorial, 2020 was the latest release).

The MEPS data files include the Full-Year Consolidated Data files, which is the calendar-year summary of the different longitudinal panels. The Full-Year Consolidated Data files contain information on the annual healthcare expenditures by the type of care; it contains data on spending, insurance coverage, health status, patient satisfaction, and several health conditions. The Full-Year Consolidated Data files also contains information from several surveys (e.g,. Diabetes Care Survey).



MEPS datafiles are located in this page

Download Data Files, Documentation, and Codebooks

Each **data file** contains the following:

- **Documentation file**
  contains technical and programming information on how the data were collected and how file variables were edited, as well as survey sampling information and crosswalks from variables to the survey instruments.
- **Codebooks**
  contain a list of variables, their labels, and unweighted and weighted frequencies.
- **Programming statements files**
  and other files including data files in ASCII format.

For SAS, Stata, and R exercises presented in the latest MEPS workshops, please visit the MEPS repository on AHRQ's GitHub page.

To search and explore variables across AHRQ's public use files, please visit the **MEPS-HC Data Tools Variable Explorer**.

Click the links below to download excel files containing the file names and numbers for the annualized Public Use Files (note that Public Use Files covering multiple years of data -- such as the panel-specific longitudinal files -- are not included):

- **PUFID_matrix.xlsx**: matrix format (wide) with clickable hyperlinks to Data Files pages. Recommended for manual reference.
- **PUFID.xlsx** (or **PUFID.csv**): list format (long) with text links to Data Files pages. Recommended for programmatic usage.

Figure 1.1: Figure 1 - Location of MEPS data files

1.3 Load MEPS data into R

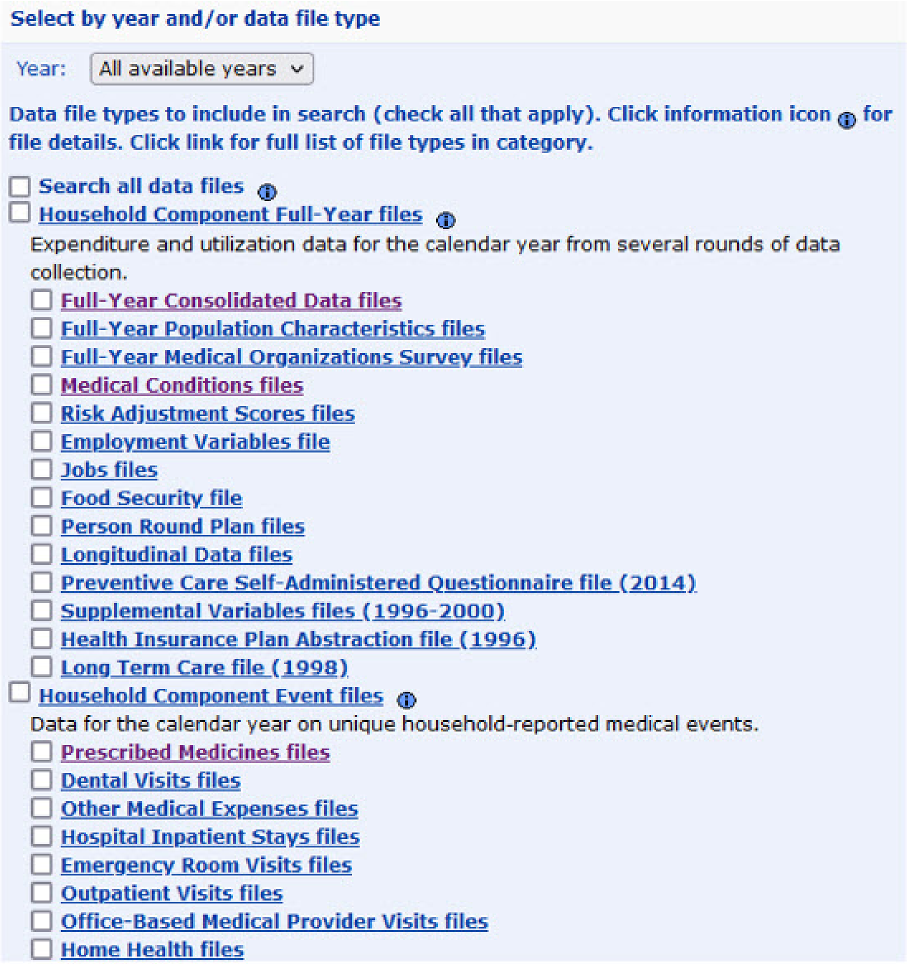
MEPS data can be downloaded onto your local storage and read into a statistical software program such as Stata or R. But you can also communicate directly with the AHRQ MEPS website to load your data rather than having to download it. We will load the Full-Year Consolidated Data file from 2020, which is named `HC-224`. To find out the name of the file, you will need to go MEPS data files site and click on the Full-Year Consolidated Data files. In this page (Figure 2), you can see the data file with the code `HC-224`, which is the Full-Year Consolidated Data file for 2020. When we enter this into our R code, we will use the file name `h224`.

You will need to download and install the `MEPS` package. The `MEPS` package will provide tools for you to load and manipulate the MEPS Data files. You will need to have R `devtools` package installed.

```
## Install the devtools package
# install.packages("devtools") ## You only need to install this once
# library("devtools")  ## You will need to reload the MEPS package each time you resta
# install_github("e-mitchell/meps_r_pkg/MEPS") ## This will install the MEPS package f
```

There are two methods to load MEPS data into R.

Method 1 requires that you know the file name. In this example, the MEPS 2020 Full-Year Consolidated Data file is named `h224`. We will use the `read_MEPS` function to load the MEPS data onto R.



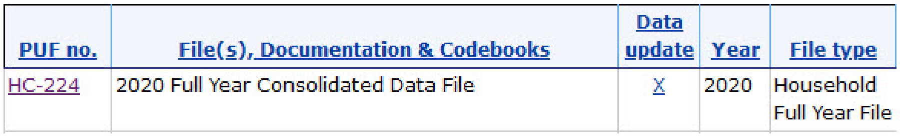
Select by year and/or data file type

Year: All available years ⌄

Data file types to include in search (check all that apply). Click information icon ⓘ for file details. Click link for full list of file types in category.

- □ **Search all data files** ⓘ
- □ **Household Component Full-Year files** ⓘ
  Expenditure and utilization data for the calendar year from several rounds of data collection.
  - □ **Full-Year Consolidated Data files**
  - □ **Full-Year Population Characteristics files**
  - □ **Full-Year Medical Organizations Survey files**
  - □ **Medical Conditions files**
  - □ **Risk Adjustment Scores files**
  - □ **Employment Variables file**
  - □ **Jobs files**
  - □ **Food Security file**
  - □ **Person Round Plan files**
  - □ **Longitudinal Data files**
  - □ **Preventive Care Self-Administered Questionnaire file (2014)**
  - □ **Supplemental Variables files (1996-2000)**
  - □ **Health Insurance Plan Abstraction file (1996)**
  - □ **Long Term Care file (1998)**
- □ **Household Component Event files** ⓘ
  Data for the calendar year on unique household-reported medical events.
  - □ **Prescribed Medicines files**
  - □ **Dental Visits files**
  - □ **Other Medical Expenses files**
  - □ **Hospital Inpatient Stays files**
  - □ **Emergency Room Visits files**
  - □ **Outpatient Visits files**
  - □ **Office-Based Medical Provider Visits files**
  - □ **Home Health files**

Figure 1.2: Figure 2 - Full-Year Consolidated Data files and other data types



PUF no.	File(s), Documentation & Codebooks	Data update	Year	File type
HC-224	2020 Full Year Consolidated Data File	X	2020	Household Full Year File

Figure 1.3: Figure 3 - H224 is the MEP 2020 Full-Year Consolidated Data file.

When using Method 2 to load the MEPS data, we don't need to know the file name, but we need to know the year and the data type. For example, for the Full-Year Consolidated Data file, we use the `year = 2020` and `type = "FYC"` option. For this method, we will also use the `read_MEPS` function to the MEPS data onto R.

The `tolower` function is used to change all the variable names from upper case to lower case. MEPS defaults the column names to upper case. I like to change this to lower case because it's easier for me to type.

```r
### Load the MEPS package
library("MEPS") ## You need to load the library every time you restart R

#### Method 1: Load data from AHRQ MEPS website
hc2020 = read_MEPS(file = "h224")

#### Method 2: Load data from AHRQ MEPS website
hc2020 = read_MEPS(year = 2020, type = "FYC")

## Change column names to lowercase
names(hc2020) <- tolower(names(hc2020))
```

There are over 1400 variables in the MEPS 2020 Full-Year Consolidated Data file. We can reduce this to the essential variables using the `subset` function. This will generate a smaller data frame that we will call `keep_meps`. The variables that we want to collect are the subject unique identifier (`dupersid`), the survey weights (`varpsu`, `varstr`, `perwt20f`), and the total healthcare expenditures for 2020 (`totexp20`).

```r
### Keep the subject's unique ID, survey weights, and total expenditures
keep_meps <- subset(hc2020, select = c(dupersid, varpsu, varstr, perwt20f, totexp20))

head(keep_meps) ## View the first six rows of the data frame
```

```
## # A tibble: 6 x 5
##   dupersid   varpsu varstr perwt20f totexp20
##   <chr>       <dbl>  <dbl>    <dbl>    <dbl>
## 1 2320005101      1   2079    8418.      459
## 2 2320005102      1   2079    5200.      564
## 3 2320006101      1   2028    2140.      140
## 4 2320006102      1   2028    2216.     4673
## 5 2320006103      1   2028    4157.      410
## 6 2320012102      2   2069    1961.     2726
```

Since MEPS uses a complex survey design, these weights are needed to estimate standard errors that are reflective of the representative sample of the U.S. population. We'll learn how to apply these survey weights to the MEPS data files in a future tutorial.

1.4 Conclusions

Loading MEPS data into R allows us to perform analysis easily and quickly. In this tutorial, you learned how to load MEPS data into R directly from the MEPS website. However, you can also download the MEPS data onto your local storage and use the `setwd` command to set the working directory.

In future tutorials, we'll learn how to apply the survey weights and perform descriptive analyses using the MEPS data files.

1.5 Acknowledgements

There are a lot of tutorials on how to use MEPS data with R. I found the AHRQ MEPS GitHub page to be an invaluable resource.

This is a work in progress, and I may update this in the future.

Chapter 2

Merging files

2.1 Introduction

We want to merge the Full-Year Consolidated Data file with the Medical Conditions file so that we can identify patients with a diagnosis of diabetes.

2.2 Load MEPS data

We need to lead the MEPS Full-Year Consolidated Data file and the Medical Conditions file from 2020. There are two methods to loading MEPS data into R. Method 1 requires you to know the name of the file. For example, the Medical Conditions file from 2020 is named `h222`. In Method 2, you need to know the `type =` of data you want to load. For example, the Medical Conditions file is named `CONDITIONS`.

I like to work with column names that are in the lower case, so I used the `tolower` function to change the column names from upper case to lower case.

```
### Load the MEPS package
library("MEPS") ## You need to load the library every time you restart R

#### Method 1: Load data from AHRQ MEPS website
hc2020 = read_MEPS(file = "h224")
mc2020 = read_MEPS(file = "h222")

#### Method 2: Load data from AHRQ MEPS website
hc2020 = read_MEPS(year = 2020, type = "FYC")
mc2020 = read_MEPS(year = 2020, type = "CONDITIONS")

## Change column names to lowercase
```

```
names(hc2020) <- tolower(names(hc2020))
names(mc2020) <- tolower(names(mc2020))
```

2.3 Merge MEPS data

Now that we have both the `h224` and `h222` file loaded into R, we can marge these files together. The Full-Year Consolidated Data file contains unique patients (e.g., each row is a unique patient); hence, the unique identifier `dupersid` is not repeatable. Figure 1 illustrates an example of a table with each row as a unique subject. Note how the `dupersid` does not repeat.

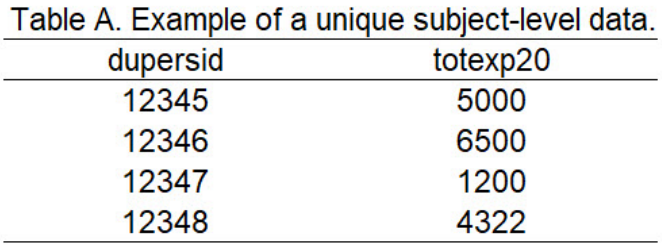


Table A. Example of a unique subject-level data.

dupersid	totexp20
12345	5000
12346	6500
12347	1200
12348	4322

Figure 2.1: Figure 1 - Example table with unique patients.

However, in the Medical Conditions file, the rows are for the number of unique diagnosis grouped by the patient. In other words, the Medical Conditions file will contain repeated `dupersid` for each diagnosis. For example, a person can have 5 diagnosis grouped by their `dupersid`. In Figure 2, we have an example table with a subject `dupersid = 12345` who has five diagnosis (`icd10cdx`).

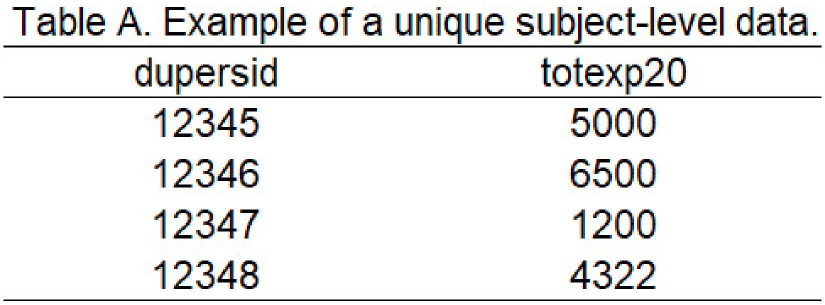
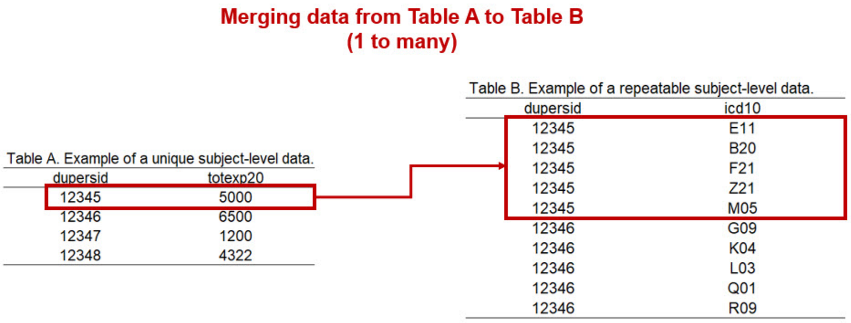


Table A. Example of a unique subject-level data.

dupersid	totexp20
12345	5000
12346	6500
12347	1200
12348	4322

Figure 2.2: Figure 2 - Example table where the unique patient identifier repeats.

When we merge the Full-Year Consolidated Data file (which is unique to the `dupersid`) with the Medical Conditions file (which has repeatable `dupersid`), we will merge using a 1 to many merge (`Figure 3a`). `Figure 3a` illustrates the merge between the unique subject-level table to the repeatable subject-level table.



Merging data from Table A to Table B
(1 to many)

Table A. Example of a unique subject-level data.

dupersid	totexp20
12345	5000
12346	6500
12347	1200
12348	4322

Table B. Example of a repeatable subject-level data.

dupersid	icd10
12345	E11
12345	B20
12345	F21
12345	Z21
12345	M05
12346	G09
12346	K04
12346	L03
12346	Q01
12346	R09

Figure 2.3: Figure 3a - Merging tables (1 to many).

But we also want to make sure that we include all the patients in the Full-Year Consolidated Data file. Not all patients will have a diagnostic code, so we need to be careful that we don't accidentally drop them from the query. `Figure 3b` illustrates our intention to merge all the data from the Full-Year Consolidated Data file with some of the data from the Medical Conditions file.

Now that we understand how we want to merge the data, we can proceed to write the code.

There are two methods to merge the data files.

Method 1: We use the `merge` function to merge the two MEPS data files. The `by =` option is where we enter the matching variable `dupersid`. We will call the merged data set `total`. Using the '`merge` function, we are telling R that we want to do a 1 to many match between the Full-Year Consolidated Data file and the Medical Conditions file using the `dupersid` as the matching variable. We have to include the `all.x = TRUE` argument because we want to make sure we include the patients without any diagnostic codes.

```r
## MERGE data - Medical conditions and household component
# merge two data frames by ID; there are two methods to do this:

#### Method 1: Native R function; Note: all.x means that we pull all dupersid, even the ones that
total <- merge(hc2020, mc2020, by = "dupersid", all.x = TRUE)
```

Method 2: We use the `left_join` function from the `dplyr` package to merge the two MEPS data files. The `by =` option is where we enter the matching variable `dupersid`. We will call the merged data set `total`. Using the `left_join`

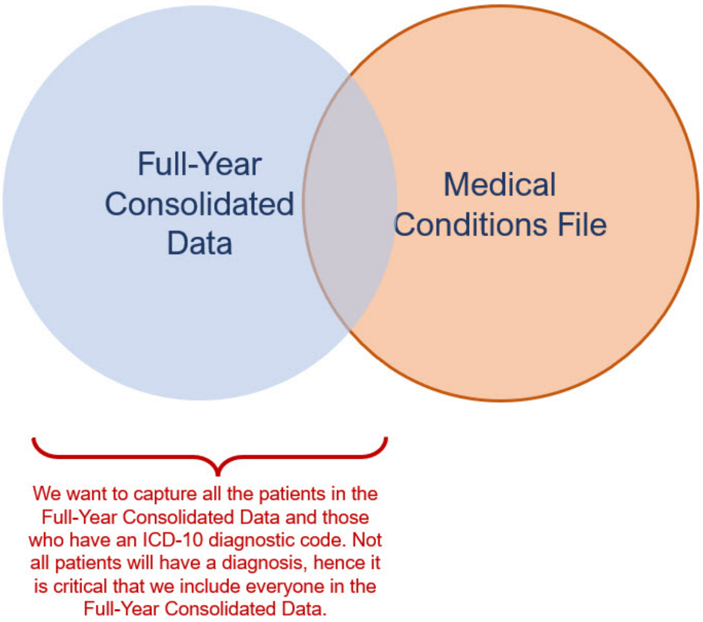

Full-Year Consolidated Data
Medical Conditions File
We want to capture all the patients in the Full-Year Consolidated Data and those who have an ICD-10 diagnostic code. Not all patients will have a diagnosis, hence it is critical that we include everyone in the Full-Year Consolidated Data.

Figure 2.4: Figure 3b - Merging tables with all patients in the Full-Year Consolidated Data file and some of the data from the Medical Conditions file.

function, we are telling R that we want to do a 1 to many match between the Full-Year Consolidated Data file and the Medical Conditions file using the `dupersid` as the matching variable. The `left_join` function is based on the SQL language syntax and operates in the same manner.

```
#### Method 2: Use SQL syntax (left_join)
library("dplyr")
total <- left_join(hc2020, mc2020, by = "dupersid")
```

Once the two data files are merged, we will have a data frame with repeatable `dupersid`. Notice that the `totexp20` variable from Table A is merged along with the `icd10` variable from Table B.

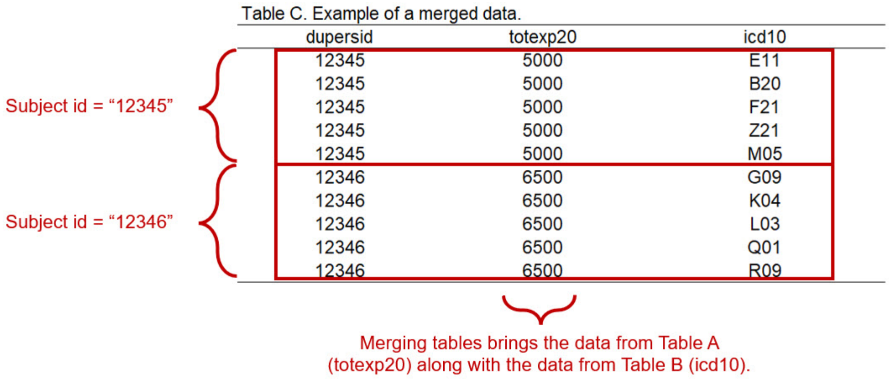


Table C. Example of a merged data.

dupersid	totexp20	icd10
12345	5000	E11
12345	5000	B20
12345	5000	F21
12345	5000	Z21
12345	5000	M05
12346	6500	G09
12346	6500	K04
12346	6500	L03
12346	6500	Q01
12346	6500	R09

Subject id = “12345”

Subject id = “12346”

Merging tables brings the data from Table A (totexp20) along with the data from Table B (icd10).

Figure 2.5: Figure 4 - Merging data from Table A to Table B.

2.4 Reduce dataframe to a few variables

Our `total` dataframe has 1481 variables and 80,802 observations. We want to make this dataframe manageable, so we'll create a limited dataframe with only the variables we're interested in. To do this, we'll use the `subset()` function.

For this exercise, we'll keep the `dupersid`, `varpsu.x`, `varsry.x`, `perwt20f.x`, and `icd10cdx` variables by using the `subset()` function. We'll call our reduced dataframe `keep_mep2`. (Note: The `*.x` indicates the table on the left. We want to keep the `varpsu`, `varstr`, and `perwt20f` from the `hc2020` table. The `mc2020` table has duplicate variables that are denoted by `*.y`.)

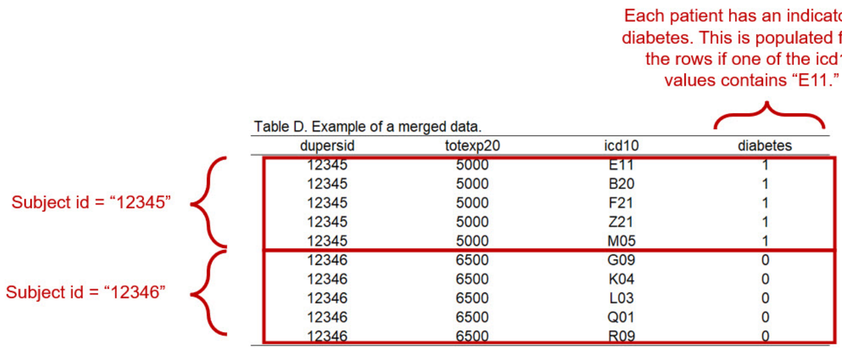
```
keep_meps2 <- subset(total, select = c("dupersid", "varpsu.x", "varstr.x", "perwt20f.x
```

2.5 Add an indicator for a specific ICD10 diagnostic code

Our data frame has multiple rows grouped by the patient's id (`dupersid`); these rows are based on the various ICD-10 diagnostic codes. For example, patient `12345` has 5 ICD-10 diagnostic codes; hence, they have 5 rows (`Figure 4`).

Suppose we want to generate a binary indicator to identify patients with an ICD-10 diagnosis for diabetes (`E11`). In our example (`Figure 4`), patient `12345` has an ICD10 code for diabetes (`E11`).

We can create an indicator variable that will be unique to the patient for having diabetes. What we want to see if a new variable that identifies a patients with the specific ICD-10 code of interest. `Figure 5` illustrates the indicator variable for diabetes as an additional column `diabetes_indicator`.



Each patient has an indicator for diabetes. This is populated for all the rows if one of the icd10 values contains "E11."

Table D. Example of a merged data.

dupersid	totexp20	icd10	diabetes
12345	5000	E11	1
12345	5000	B20	1
12345	5000	F21	1
12345	5000	Z21	1
12345	5000	M05	1
12346	6500	G09	0
12346	6500	K04	0
12346	6500	L03	0
12346	6500	Q01	0
12346	6500	R09	0

Subject id = "12345"
Subject id = "12346"

Figure 2.6: Figure 5 - Indicator variable for diabetes.

We create the indicator and call it `diabetes`, which is defined as `icd10cdx == "E11"`. We will code this as `0` for no diabetes and `1` for diabetes. Then, we count the number of time a patient as `E11` in their `icd10cdx` column. I added the following option to the code (`| is.na(total$icd10cdx)`) because I want to make sure that all patients in the `total` table that do not have an ICD-10 code for `E11` is coded as `0`. There may be some patients that have `NA` or missing data in the `icd10cdx` variable. If the `icd10cdx` value is `NA`, this may not be coded with a `0`. Hence, we have to add the `| is.na(total$icd10cdx` code to ensure that we get a value of `0`.

```
## Change to unique subject (each row is a unique subject)
#### Generate a variable to identify diabetes diagnosis for repeated rows
```

```
library("tidyverse")  ## Load tidyverse

keep_meps2$diabetes[total$icd10cdx != "E11" | is.na(total$icd10cdx)] = 0
keep_meps2$diabetes[total$icd10cdx == "E11"] = 1

table(keep_meps2$diabetes) ## Visualize the number of patients with diabetes and no diabetes
```

```
##
##     0     1
## 87345  2693
```

```
### This code chunk calculates the number of times E11 appears for a unique patient
keep_meps2 <- keep_meps2 %>%
  group_by(dupersid) %>%
  mutate(diabetes_indicator = sum(diabetes == "1", na.rm = TRUE)) %>%
  ungroup
table(keep_meps2$diabetes_indicator)
```

```
##
##     0     1     2
## 71804 17295   939
```

According to our results, there were 17,295 events where a patient had one diagnostic code for `E11` and 939 events where a patient had two diagnostic codes for `E11`. How did this occur? MEPS public files only list the first three digits of the ICD-10 code to protect the identity of the patient. The ICD-10 diagnostic code has more digits beyond the first three. For example, an ICD-10 diagnosis for Type 2 diabetes with diabetic chronic kidney disease is `E11.22`. Hence, there will be patients with unique ICD-10 codes that may appear identical because only the first three digits are present in the MEPS public files.

In our example, we have patients with 1 and 2 ICD-10 diagnostic codes for `E11`. We would like to create a binary indicator of diabetes, so we need to take the current information and transform the variable `diabetes` in the `keep_meps` dataframe into a new variable that only has `0` and `1`.

We can do this by combining the `mutate` function with the `ifelse` function. See the code below:

```
keep_meps2 <- keep_meps2 %>%
  group_by(dupersid) %>%
  mutate(diabetes_binary = ifelse(diabetes_indicator >= 1, 1, 0), na.rm = TRUE) %>%
  ungroup
table(keep_meps2$diabetes_binary)
```

```
##
##     0     1
## 71804 18234
```

Now, we have a new binary indicator variable. The `diabetes_binary` variable is coded `1` if the patient has the `E11` diagnostic code and `0` if the patient does not.

2.6 Collapse dataframe to a single unique patient

But since this is a dataframe with duplicated patients, we want to collapse this into a dataframe where each row is a single unique patient.

Since a lot of the variables in the dataframe are the same when grouped by the unique `dupersid`, we can estimate the mean and get the same value. For example, let's look at `Figure 5` again. For `dupersid == 12345`, there are five values for `totexp20`, which are:

- `5000` when `icd10` is `E11`,
- `5000` when `icd10` is `B20`,
- `5000` when `icd10` is `F21`,
- `5000` when `icd10` is `Z21`, and
- `5000` when `icd10` is `M05`

Averaging the `totexp20` for `dupersid == 12345` will result in a value of `5000`.

Hence, when we average the diabetes `diabetes_binary` variable, we will get a value of `1` or `0`.

Using this knowledge, we can collapse our data to a single `dupersid` and remove the duplicates.

The `icd10cdx` variable will yielded `NA` because it can't be collapsed numerically due to its `string` data type.

There are two methods to collapse the dataframe to unique patients:

Method 1: Use the `dplyr` package and the `summarize_all` function with the `list()` function.

```
#### Collapse the repeated rows to a single unique subject
meps_per <- keep_meps2 %>%
  group_by(dupersid) %>%
  summarize_all(list(mean))

table(meps_per$diabetes_binary)
```

```
##
##      0      1
## 25202  2603
```

Method 2: Use the `summarise` function. This method will generate a dataframe with two variables (`dupersid` and `diabetes_binary2`).

```
meps_per2 <- keep_meps2 %>%  ### An alternative method but only generates two variables (dupersic
  group_by(dupersid) %>%
  summarise(diabetes_binary2 = mean(diabetes_binary)) %>%
  as.data.frame()

table(meps_per2$diabetes_binary2)
```

```
##
##     0     1
## 25202  2603
```

For the rest of the tutorial, I'll use Method 1 because I want to keep the other variables.

`Figure 6` illustrates what our dataframe should look like after we collapsed the data to a single unique patient.

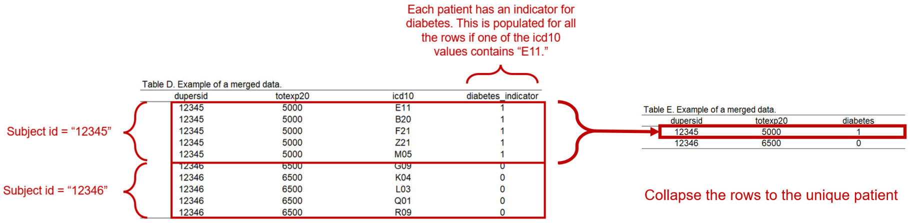


Table D. Example of a merged data.

dupersid	totexp20	icd10	diabetes_indicator
12345	5000	E11	1
12345	5000	B20	1
12345	5000	F21	1
12345	5000	Z21	1
12345	5000	M05	1
12346	6500	G09	0
12346	6500	K04	0
12346	6500	L03	0
12346	6500	Q01	0
12346	6500	R09	0

Table E. Example of a merged data.

dupersid	totexp20	diabetes
12345	5000	1
12346	6500	0

Each patient has an indicator for diabetes. This is populated for all the rows if one of the icd10 values contains "E11."
Subject id = "12345"
Subject id = "12346"
Collapse the rows to the unique patient

Figure 2.7: Figure 6 - Collapse rows to a unique patient with a diabetes indicator.

2.7 Conclusions

With this tutorial, we've learned how to merge two data files from MEPS and collapse them to a dataframe of unique patients. MEPS has additional data files that contain information that might be important for your work. For example, we can use these methods to merge the Prescription Drug file and create indicators for patients who are on opioids. However, you will need to carefully read through the documentation for each data file to understand what kind of information they contain. Feel free to explore using these strategies to merge additional MEPS data files to your existing cohort.

2.8 Acknowledgements

There are a lot of tutorials on how to use MEPS data with R. I found the AHRQ MEPS GitHub page to be an invaluable resource.

David Ranzolin has a great presentation on how to use the `mutate` function in R. I liked the examples he used, and the presentation is succint and informative.

Another great resource is by Joachim Schork, author and founder of Statistics Globe who wrote a great blog about collapsing data on a unique identifier.

I learned how to use the `left_join` function from this blog by Sharon Machlis on InfoWorld. She uses `dplyr` to invoke the `left_join` function which is a based on `SQL` language.

This is a work in progress, and I may update this in the future.

Chapter 3

Applying weights

3.1 Introduction

The Medical Expenditure Panel Survey (MEPS) is based on a complex survey design. Hence, it is necessary to apply survey weights to generate estimates that are representative of the United States (US) population. The weights take into account the stratification, clustering, sampling, and non-response based on the Current Population Survey. Although you can perform descriptive and complex analyses without the weights, they will not provide you with accurate standard errors of the population. Rather, not applying the weights will only yield standard errors for the sample.

3.2 Types of weights

In MEPS, there are three types of weights that are critical for most descriptive and multivariate analyses: person weight (`perwtXXf`), stratum (`varstr`), and cluster (`varpsu`). The `XX` is replaced by the year of the survey. For example, the person weight in 2020 is labelled as `perwt20f`.

3.3 Loading the data

Let's use the MEPS Full-Year Consolidated File from 2020. From our previous tutorial, you can load data using the `MEPS` library function `read_MEPS`. There are two methods that you can use to load data into R.

```
### Load the MEPS package
library("MEPS") ## You need to load the library every time you restart R

#### Method 1: Load data from AHRQ MEPS website
```

```
hc2020 = read_MEPS(file = "h224")

#### Method 2: Load data from AHRQ MEPS website
hc2020 = read_MEPS(year = 2020, type = "FYC")

## Change column names to lowercase
names(hc2020) <- tolower(names(hc2020))
```

Once the data has been loaded, we can look at how many variables there are.

```
## The number of columns represents the number of variables in the hc2020 dataframe.
ncol(hc2020)
```

```
## [1] 1451
```

We have over 1400 variable. This is a very large dataframe. We can reduce this to a manageable size by keeping only the variables that are important. Let's keep the unique patient identifier (`dupersid`), weights (`perwt20f`, `varstr`, and `varpsu`), and the total expenditures (`totexp20`).

```
## Create a smaller dataframe
keep_hc2020 <- subset(hc2020, select = c(dupersid, perwt20f, varstr, varpsu, totexp20,
head(keep_hc2020)
```

```
## # A tibble: 6 x 7
##   dupersid   perwt20f varstr varpsu totexp20 sex          povcat20
##   <chr>         <dbl>  <dbl>  <dbl>    <dbl> <dbl+lbl>    <dbl+lbl>
## 1 2320005101    8418.   2079      1      459 2 [2 FEMALE] 2 [2 NEAR POOR]
## 2 2320005102    5200.   2079      1      564 1 [1 MALE]   2 [2 NEAR POOR]
## 3 2320006101    2140.   2028      1      140 2 [2 FEMALE] 3 [3 LOW INCOME]
## 4 2320006102    2216.   2028      1     4673 1 [1 MALE]   1 [1 POOR/NEGATIVE]
## 5 2320006103    4157.   2028      1      410 1 [1 MALE]   3 [3 LOW INCOME]
## 6 2320012102    1961.   2069      2     2726 2 [2 FEMALE] 3 [3 LOW INCOME]
```

We can add labels to the `sex` variable where `1 = male` and `2 = female`.

3.4 Perform descriptive analysis

Now that we have a smaller dataframe with the variables of interest, let's apply the survey weights to some descriptive analysis.

Suppose you were interested in the average age of the cohort. You will need to apply the survey weights to generate the mean and standard deviation. The `survey` package comes with the `svydesign` function, which uses the survey weights in the Full-Year Consolidated File data and applies them to the cohort in preparation for analyses.

First, you will need to set the options to `adjust`, which centers the single-PSU strata arund the grand mean rather than the stratum mean. With MEPS data, we are using single-PSU (or “lonely” PSU), which is used to estimate the variance by calculating the difference of the sum of the statum’s PSU and the average statum’s PSU. The, we use the `svydesign` function to generate a complex survey design dataset (which we will call `mepsdsgn`) for analysis by applying the survey weights.

```
## Load the "survey" package
library("survey")

## Apply the survey weights to the dataframe using the svydesign function
options(survey.lonely.psu = 'adjust')

mepsdsgn = svydesign(
  id = ~varpsu,
  strata = ~varstr,
  weights = ~perwt20f,
  data = keep_hc2020,
  nest = TRUE)
```

Once the survey weights have been applied, we can use the `survey` functions to perform some descriptive analysis on the `mepsdsgn` data.

First, let’s see how many patients we have that is representative of the US population by sex. We use the `svytable` function to generate the weight sample for males and females. Adding these together will yield the weighted sample of the US population.

```
## Weighted sample of the population stratified by sex
svytable(~sex, design = mepsdsgn)
```

```
## sex
##   1 - Male 2 - Female
##  160960989  167584308
```

Using the survey weights, there are 160,960,989 males and 167,584,308 females. In total, there are 328,545,297 weighted subjects in the `mepsdsgn` data.

Let’s move on and estimate the average total expenditures for the total sample.

```
## Estimate the weighted mean total expenditure for the sample
svymean(~totexp20, design = mepsdsgn)
```

```
##            mean     SE
## totexp20 6266.1 164.38
```

The `svymean` function generates the appropriate average and standard error (SE) of the total sample that is representative of the US population. In 2020, the average total expenditure was $6266 (SE, 164).

In our `mepsdsgn` data, we have sex, which is a binary variable. Let’s estimate the total expenditures between males and females in the MEPS Full-Year Consolidated data. To estimate the mean between two groups, we'll need to use the `svyby` function along with the `svymean` function.

```
## Estimate the weight mean total expenditure for males and females
svyby(~totexp20, ~sex, mepsdsgn, svymean)
```

```
##                   sex totexp20       se
## 1 - Male     1 - Male 5861.278 243.5624
## 2 - Female 2 - Female 6654.998 205.0776
```

The average total expenditures for male and female are $5861 (SE, 244) and $6655 (SE 205), respectively.

We can perform crosstabulations with the `svytable` function. Let's look at the distribution of males and females across various poverty categories. In the MEPS codebook, poverty category are groups as: 1 = Poor/Negative, 2 = Near Poor, 3 = Low Income, 4 = Middle Income, and 5 = High Income.

```
## Crosstab sex and poverty category
svytable(~sex + povcat20, design = mepsdsgn)
```

```
##             povcat20
## sex                 1        2        3        4        5
##   1 - Male   16644995  5955576 18910001 45083571 74366846
##   2 - Female 21002826  6672752 21753502 47750327 70404901
```

To generate the proportions, you will need to use `prop.table`. We add the `margin = 1` option to calculate the column total.

```
prop.table(svytable(~sex + povcat20, design = mepsdsgn), margin = 1) ### margin = 1 ca
```

```
##             povcat20
## sex                   1          2          3          4          5
##   1 - Male   0.10341012 0.03700012 0.11748189 0.28009005 0.46201783
##   2 - Female 0.12532693 0.03981729 0.12980632 0.28493316 0.42011631
```

We can combine these into a contingency table using the `tbl_svysummary` function from the `gtsummary` package. We will also use the `tidyverse` package to manipulate the data more easily.

```
## Load libraries
library("tidyverse")
library("gtsummary")

## Contingency table (crosstabulations between sex and poverty category)
mepsdsgn %>%
  tbl_svysummary(by = sex, percent = "column", include = c(povcat20))
```

Characteristic	**1 - Male**, N = 160,960,989	**2 - Female**,
FAMILY INC AS % OF POVERTY LINE - CATEGORICAL		
1	16,644,995 (10%)	21,002,8
2	5,955,576 (3.7%)	6,672,75
3	18,910,001 (12%)	21,753,5
4	45,083,571 (28%)	47,750,3
5	74,366,846 (46%)	70,404,9

Based on these weighted sample numbers, there are more males who are in the High Income category compared to females (46% versus 42%).

3.5 Conclusions

The MEPS data uses weights to generate estimations that are reflective of the US population. The survey package from R will allow us to apply these weights using the svydesign function, which requires us to enter the patient weight, stratum, and cluster values. Once these are applied, we can use the suite of functions from the survey package to perform descriptive analysis on the population. The svymean generates the population average and the svytable generates the population frequencies. Having a good understanding of how these weights are used with MEPS data will allow you to generate estimates of the population in your epidemiology work.

3.6 Acknowledgements

The survey package and functions were developed by Thomas Lumley and can be found here.

The gtsummary package and instructions are developed by Daniel D. Sjoberg, Joseph Larmarange, Michael Curry, Jessica Lavery, Karissa Whiting, Emily C. Zabor, which can be found at their website.

This is a work in progress, and I expect to make updates in the future.

Chapter 4

Footnotes and citations

4.1 Footnotes

Footnotes are put inside the square brackets after a caret `^[]`. Like this one [1].

4.2 Citations

Reference items in your bibliography file(s) using `@key`.

For example, we are using the **bookdown** package [Xie, 2022] (check out the last code chunk in index.Rmd to see how this citation key was added) in this sample book, which was built on top of R Markdown and **knitr** [Xie, 2015] (this citation was added manually in an external file book.bib). Note that the `.bib` files need to be listed in the index.Rmd with the YAML `bibliography` key.

The RStudio Visual Markdown Editor can also make it easier to insert citations: https://rstudio.github.io/visual-markdown-editing/#/citations

[1] This is a footnote.

Chapter 5

Blocks

5.1 Equations

Here is an equation.

$$f\left(k\right) = \binom{n}{k} p^k \left(1-p\right)^{n-k} \tag{5.1}$$

You may refer to using `\@ref(eq:binom)`, like see Equation (5.1).

5.2 Theorems and proofs

Labeled theorems can be referenced in text using `\@ref(thm:tri)`, for example, check out this smart theorem 5.1.

Theorem 5.1. *For a right triangle, if c denotes the* length *of the hypotenuse and a and b denote the lengths of the* **other** *two sides, we have*

$$a^2 + b^2 = c^2$$

Read more here https://bookdown.org/yihui/bookdown/markdown-extensions-by-bookdown.html.

5.3 Callout blocks

The R Markdown Cookbook provides more help on how to use custom blocks to design your own callouts: https://bookdown.org/yihui/rmarkdown-cookbook/custom-blocks.html

Chapter 6

Sharing your book

6.1 Publishing

HTML books can be published online, see: https://bookdown.org/yihui/bookdown/publishing.html

6.2 404 pages

By default, users will be directed to a 404 page if they try to access a webpage that cannot be found. If you'd like to customize your 404 page instead of using the default, you may add either a `_404.Rmd` or `_404.md` file to your project root and use code and/or Markdown syntax.

6.3 Metadata for sharing

Bookdown HTML books will provide HTML metadata for social sharing on platforms like Twitter, Facebook, and LinkedIn, using information you provide in the `index.Rmd` YAML. To setup, set the `url` for your book and the path to your `cover-image` file. Your book's `title` and `description` are also used.

This `gitbook` uses the same social sharing data across all chapters in your book- all links shared will look the same.

Specify your book's source repository on GitHub using the `edit` key under the configuration options in the `_output.yml` file, which allows users to suggest an edit by linking to a chapter's source file.

Read more about the features of this output format here:

https://pkgs.rstudio.com/bookdown/reference/gitbook.html

Or use:

```
?bookdown::gitbook
```

Bibliography

Yihui Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition, 2015. URL http://yihui.org/knitr/. ISBN 978-1498716963.

Yihui Xie. *bookdown: Authoring Books and Technical Documents with R Markdown*, 2022. URL https://CRAN.R-project.org/package=bookdown. R package version 0.29.