MEPS tutorials

Mark Bounthavong

2024-01-21

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.

Book link

The GitHub link to the MEPS tutorials collection: https://mbounthavong.github.io/MEPS_tutorials_book/

Table of contents

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.

Chapter 4 using condition-event link file

Chapter 5 simple trend analysis with linear models

Chapter 6 interrupted time series analysis

Chapter 7 helpful notes

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).

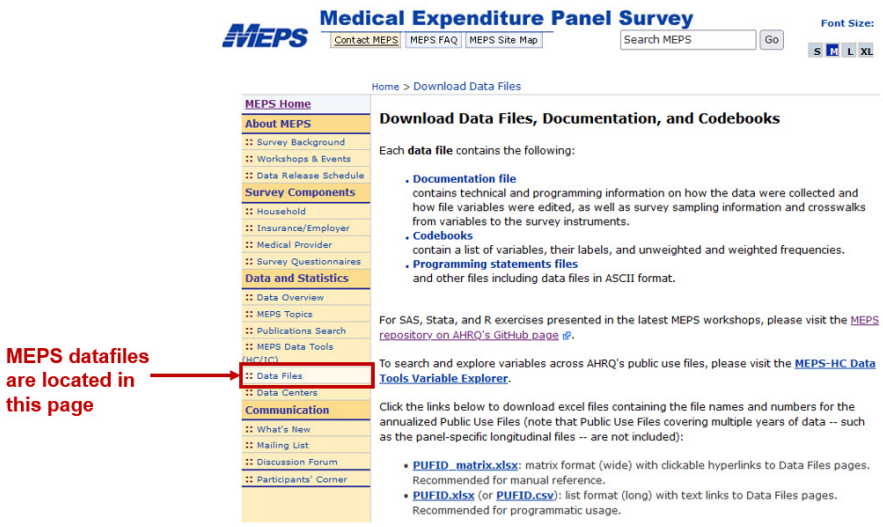


Figure 1.1: Figure 1 - Location of MEPS data files

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`

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.

function to load the MEPS data onto R.

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.

```
### 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`).

```
### 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`).

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.

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.

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.

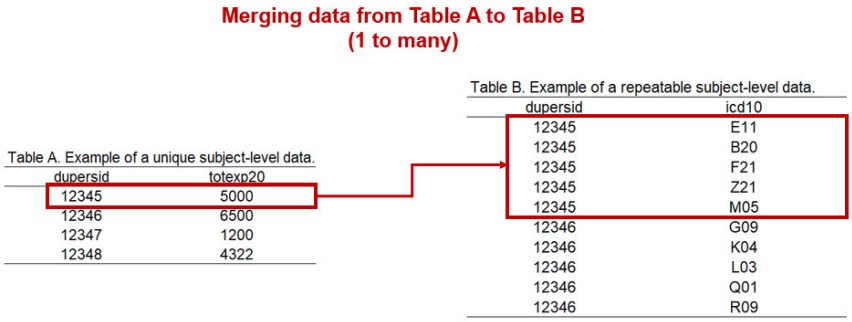


Figure 2.3: Figure 3a - Merging tables (1 to many).

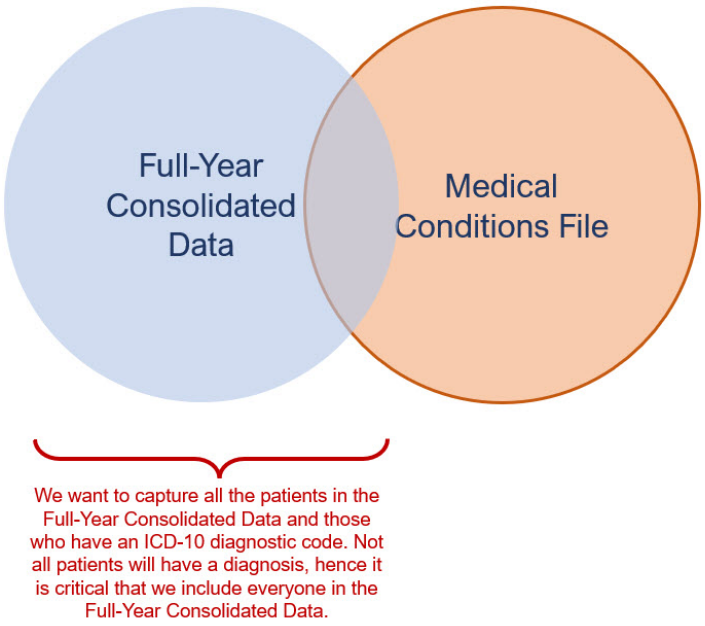


Figure 2.4: Figure 3b - Merging tables with all patients in the Full-Year Consolidated Data file and some of the data from the Medical Conditions file.

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.

```
## 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` 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.

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`.)

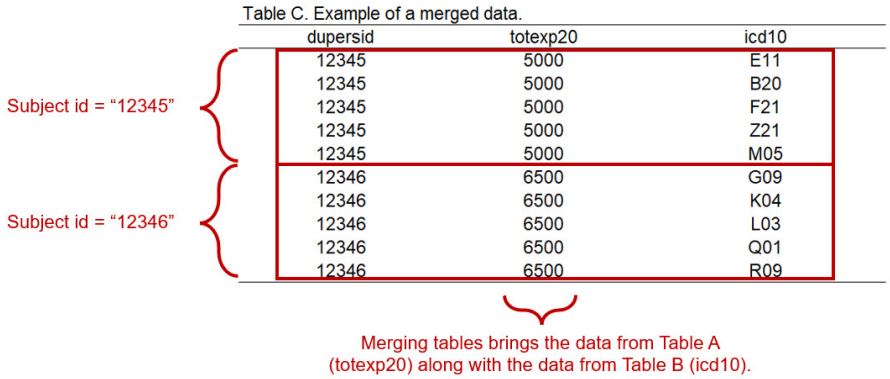


Figure 2.5: Figure 4 - Merging data from Table A to Table B.

```
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`.

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`.

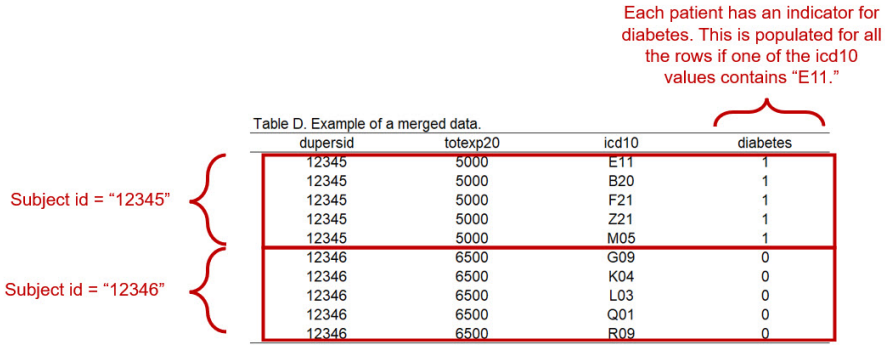


Table D. Example of a merged data.

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

Figure 2.6: Figure 5 - Indicator variable for diabetes.

```
## 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 (dupersid
  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.

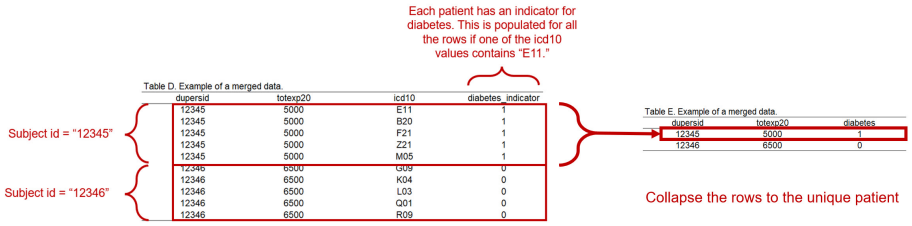


Figure 2.7: Figure 6 - Collapse rows to a unique patient with a diabetes indicator.

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

Using condition-event link (CLNK) file

4.1 Introduction

The Agency for Healthcare Research and Quality (AHRQ) Medical Expenditure Panel Survey (MEPS) categorizes expenditures into different components. Healthcare expenditures (e.g., costs and utilization) are provided for each individual respondent in MEPS. For instance, healthcare expenditures are categorized as total healthcare expenditures (`totexp21`), office-based expenditures (`obvexp21`), outpatient expenditures (`opvexp21`), inpatient expenditures (`iptexp21`), and prescription expenditures (`rxexp21`) to name a few. Reading the long but detailed documentation is a good way to learn more about the different expenditure categories. Moreover, you can also review **Appendix 3** of the documentation (see Figure below).

These costs and utilization provide information about the annual expenditures associated with each category for each individual respondent. But this doesn't provide disease-specific expenditures. For example, an individual may have an annual office-based visits healthcare cost of $10,000, but part of this costs may be due to a specific disease such as migraine. How much of the $10,000 is due to migraine-related office-based visits? One can answer this question using the condition-event link (`CLNK`) file.

The `CLNK` file has a unique variable that can be used to link each record on the Medical Conditions file with event files from the respective year (e.g., `HC-229D` through `HC-229H`). One of these event files that we're interested in is the office-based event file (`HC-229G`). The `CLNK` file contains 6 variables:

- `dupersid` - 10-digit unique identifier

Summary of Utilization and Expenditure Variables by Health Service Category

HEALTH SERVICE CATEGORY	UTILIZATION VARIABLE(S)	EXPENDITURE VARIABLE(S)
All Health Services	--	TOT***21
Total Office Based Visits (Physician + Non-physician + Unknown)	OBTOTV21	OBV***21
Office Based Visits to Physicians	OBDRV21	OBD***21
Total Outpatient Visits (Physician + Non-physician + Unknown)	OPTOTV21	--
Sum of Facility and SBD Expenses	--	OPT***21
Facility Expense	--	OPF***21
SBD Expense	--	OPD***21
Outpatient Visits to Physicians	OPDRV21	--
Facility Expense	--	OPV***21
SBD Expense	--	OPS***21
Total Emergency Room Visits	ERTOT21	--
Sum of Facility and SBD Expenses	--	ERT***21
Facility Expense	--	ERF***21
SBD Expense	--	ERD***21
Total Inpatient Stays	IPDIS21, IPNGTD21	--
Sum of Facility and SBD Expenses	--	IPT***21
Facility Expense	--	IPF***21
SBD Expense	--	IPD***21
Total Prescription Medicines	RXTOT21	RX***21
Total Dental Visits	DVTOT21	DVT***21
Total Home Health Care	HHTOTD21	--
Agency Sponsored	HHAGD21	HHA***21
Paid Independent Providers	HHINDD21	HHN***21
Informal	HHINFD21	--
Vision Aids	--	VIS***21
Other Medical Supplies and Equipment	--	OTH***21

KEY: To complete variable name, replace * with a particular source of payment category as identified in the following tables:**

Source of Payment Category	***
Total payments (sum of all sources)	EXP
Out of Pocket	SLF
Medicare	MCR
Medicaid	MCD
Private Insurance	PRV
Veteran's Administration/CHAMPVA	VA
TRICARE	TRI
Other Federal Sources	OFD
Other State and Local Sources	STL
Workers' Compensation	WCP
Other Unclassified Sources	OSR

Figure 4.1: MEPS Appendix 3 - Expenditure variables

- `condidx` - 13-digit unique identifier for a condition
- `evntidx` - 16-digit unique identifier for each event
- `clnkidx` - 29-digit unique identifier for each record; combines `condidx` and `evntidx`
- `eventype` - indicates the type of event record (see Figure)
- `panel` - indicate the panel when the interview occurred

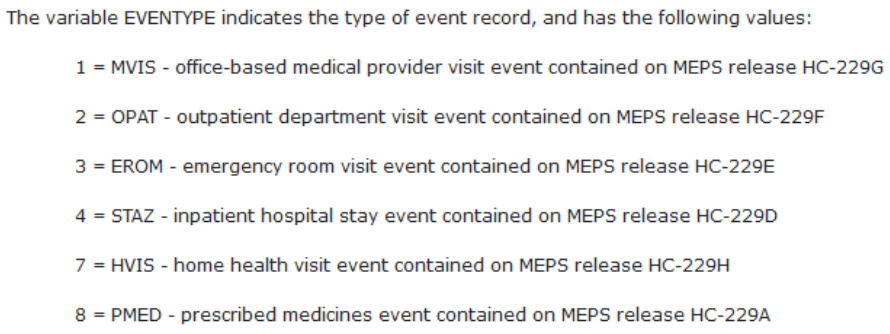


Figure 4.2: Type of event record ('eventype')

Using these files, we can acquire disease-specific expenditures from MEPS data, which may be important for those of us who are interested in these expenditures.

4.2 Motivating example - Migraine-specific expenditures

In this motivating example, we will review how to use MEPS to find the office-based expenditures and inpatient expenditures specific to migraine.

4.2.1 Part 1 - Setup

We will need to install several packages. The AHRQ MEPS GitHub site is a great source for documents, tutorials, codes, and updates. I learned a ton going through their exercises, and a lot of the code you'll see in this tutorial come from those resources.

```
# To install "MEPS" package in R, you need to do a couple of things.
### Step 1: Install the "devtools" package.
#install.packages("devtools")


### Step 2: Install the "MEPS" package from the AHRQ MEPS GitHub site.
#devtools::install_github("e-mitchell/meps_r_pkg/MEPS")


### Step 3: Load the MEPS package
library("MEPS") ## You need to load the library every time you restart R

### Step 4: Load the other libraries
library("survey")
library("foreign")
library("tidyverse")
library("psych")
```

Next, we set the global options.

```
# Set global options
options(survey.lonely.psu = "adjust") # survey option for lonely PSUs
options(dplyr.width = Inf) # Columns are not truncated
options(digits = 10) # Do not use scientific notation for large number
```

Once that's done, we will download the data directly from the MEPS site using the `read_MEPS` function. There are two ways to do this:

```
# There are two ways to load data from AHRQ MEPS website:
#### Method 1: Load data from AHRQ MEPS website
hc2021 = read_MEPS(file = "h233") # Full-year consolidated file
ob2021 = read_MEPS(file = "h229g") # Office-based visits
inpat2021 = read_MEPS(file = "h229d") # Inpatient stays
cond2021 = read_MEPS(file = "h231") # Medical conditions file
clnk2021 = read_MEPS(file = "h229IF1") # Condition-Event Link File (CLNK)


#### Method 2: Load data from AHRQ MEPS website
hc2021 = read_MEPS(year = 2021, type = "FYC") # Full-year consolidated file
ob2021 = read_MEPS(year = 2021, type = "OB") # Office-based visits
inpat2021 = read_MEPS(year = 2021, type = "IP") # Inpatient stays
cond2021 = read_MEPS(year = 2021, type = "COND") # Medical conditions file
clnk2021 = read_MEPS(year = 2021, type = "CLNK") # Condition-Event Link File (CLNK)
```

After the data are loaded, you can change the column names from upper case to lower case.

```
## Change column names to lowercase
names(hc2021) <- tolower(names(hc2021))
names(ob2021) <- tolower(names(ob2021))
names(inpat2021) <- tolower(names(inpat2021))
names(cond2021) <- tolower(names(cond2021))
names(clnk2021) <- tolower(names(clnk2021))
```

Each of these tables will have a lot of variables. To make things easier and cleaner, let's reduce the size of the tables to only include the essential variables.

```
# Keep only the variables of interest
hc2021x = hc2021 %>%
  select(dupersid, totexp21, obvexp21, iptexp21, perwt21f, varpsu, varstr, sex, racev1x)

ob2021x = ob2021 %>%
  select(dupersid, evntidx, eventrn, obdateyr, obdatemm, obxp21x, perwt21f, varpsu, varstr)

inpat2021x = inpat2021 %>%
  select(dupersid, evntidx, eventrn, numnighx, ipxp21x, perwt21f, varpsu, varstr)

cond2021x = cond2021 %>%
  select(dupersid, condidx, icd10cdx, ccsr1x:ccsr3x)
```

Next, we want identify migraine condition from the `cond2021x` file, which is the medical conditions file. The CCSR code for migraine is `NVS010` (note: the ICD10 code for migraine is `G42`, but we won't need it for this example). There are three CCSR columns (`ccsr1x`, `ccsr2x`, `ccsr3x`), and we want to concatenate these into a new column called `all_CCSR` to isolate for migraine. You can find the list of CCSR codes in the AHRQ MEPS site.

```
################################## NOTES ##################################
## Use CLNK file to map condition with events
## CCSR code: https://github.com/HHS-AHRQ/MEPS/blob/master/Quick_Reference_Guides/meps_ccsr_condi
## CCSR code for migraine: NVS010
## ICD10 code for migraine: G43
###########################################################################

# Restrict conditions to Migraine only from the conditions file
### This creates a variable called "all_CCSR" which concatenates the CCSR columns
### Then, filter() only selects rows with the "NVS010" text in the "all_CCSR" column
### Finally, this gets saved into the "migraine" object.
migraine = cond2021x %>%
```

```
  unite("all_CCSR", ccsr1x:ccsr3x, remove = FALSE) %>%
  filter(grepl("NVS010", all_CCSR))


# View freq per diagnosis code type (Pretty nice code from AHRQ)
migraine %>%
  count(icd10cdx, ccsr1x, ccsr2x, ccsr3x)
```

```
## # A tibble: 3 x 5
##   icd10cdx ccsr1x ccsr2x ccsr3x     n
##   <chr>    <chr>  <chr>  <chr>  <int>
## 1 -15      NVS010 -1     -1        26
## 2 G43      NVS010 -1     -1       556
## 3 R51      NVS010 SYM010 -1       202
```

4.2.2 Part 2 - Migraine-specific office-based expenditures

Now that we have our data files set up and prepared, we can begin to identify the migraine-specific office-based expenditures.

First, we want to isolate the office-based visits events from the `clnk2021` file. According to the figure above, `eventype == 1` is for office-based medical provider visit event. We will save these results into a new object called `ob_events`.

```
##########################################
## OUTPATIENT VISITS
##########################################
# Select Office-based events from the CLNK file
ob_events = clnk2021 %>%
  filter(eventype == 1)
```

Next, we want to merge the `ob_events` object with the `migraine` object, which contains the migraine-related conditions. We use an `inner_join`, which will only subset rows that matches between the two objects by `dupersid` and `condidx`. This will be saved as a new object called `migraine_lnk`.

```
# Merge migraine diagnosis conditions file with office-based visit file
migraine_lnk = inner_join(
  migraine, ob_events,
  by = c("dupersid", "condidx")
)
```

Since the `migraine_lnk` object has duplicate `dupersid` and `condidx`, we want to create an updated dataframe (`migraine_ob_distinct`) that contains unique

rows of `dupersid`, `evntidx`, and `eventype`. AHRQ called this process "De-duplicate." We will use this term in our exercise.

```
# Select only DISTINCT office-based events ("De-duplicate")
migraine_ob_distinct = migraine_lnk %>%
  distinct(dupersid, evntidx, eventype)
```

Then we merge the `migraine_ob_distinct` dataframe with the `ob2021x` dataframe using the `inner_join` function because we want to only include the rows that are in both dataframes. We use the `mutate` function to generate a new indicator variable called `migrane_ob_visit =1`. This will yield a dataframe that contains the office-based events specific for migraines.

```
# Merge office-based file with distinct office-based CLNK event file
### Use the inner_join() to merge ob2021x and migraine_lnk_distinct dataframes
### Then create a new indicator variable called "migraine_ob_visit = 1."
ob_migraine = inner_join(
  ob2021x, migraine_ob_distinct) %>%
  mutate(migraine_ob_visit = 1)
```

Once that's done, we can merge the `ob_migraine` dataframe with the Full-Year Consolidated (`hc2021x`) dataframe. We create two indiciator variables: `migraine_ob = 1` to capture the total number of migraine-specific office-based visits, and `fyc = 1` to indicate that this is the Full-Year Consolidated file that was merged.

```
# Merge with full-year consolidated file
### Use the full_join() to keep all the rows from both dataframes.
### Then create a new object called hc2021_ob_migraine
### & a new indicator variable called "migraine_ob = 1"
### & another new indicator variable called "fyc = 1."
### Note: These are not unique rows (repeated dupersid)
hc2021_ob_migraine = full_join(
  ob_migraine %>% mutate(migraine_ob = 1),
  hc2021x %>% mutate(fyc = 1))
```

Since this dataframe (`hc2021_ob_migraine`) contains duplicate `dupersid`, we want to create a dataframe where each row reflects a unique `dupersid`. We can do this by using the `group_by` function on specific variables that we know are unique to an individual (`dupersid`, `varstr`, `varpsu`, and `perwt21f`). These are the unique identifier of the individual and their associated stratum and weight.

We also will summarize the migraine-specific utilization such as office-based visit costs (`obxp21x`) and the total sum of those visits (`migraine_ob_visit`).

```
# Aggregate to person-level
### We have to do the estimations for the person-level at this stage
### We will estimate the total and mean values of the migraine-related expenditures
### The "migraine_ob" will be used to create an indicator for migraine-related expendi
### Need to set NA -> zero.
per_migraine_ob = hc2021_ob_migraine %>%
  group_by(dupersid, varstr, varpsu, perwt21f) %>%
  summarize(
    avgtotexp_tot               = mean(totexp21),  # average total expenditure
    totexp_ob                   = sum(obxp21x),    # sum of migraine-specific office-b
    avgexp_obvexp               = mean(obvexp21),  # average office-based visits per p
    totvisit_ob                 = sum(migraine_ob_visit), # sum of migraine-specific o
    avgvisit_migraine_ob        = mean(migraine_ob_visit), # average migraine-specific
    avgvisit_migraine_visits_ob = mean(migraine_ob), # average migraine-specific offic
    migraine_ob                 = max(migraine_ob)) %>% # indicator for migraine-speci
    replace_na(
      list(totexp_ob = 0, totvisit_ob = 0, totvisit_migraine_ob = 0, avgvisit_migraine
    )
```

The above code chunk seems intimidating, but there are sensible reasons why it is written this way. See the Figure below for a visual explanation.

The migraine-specific office-based costs and visits are expenditures that are at the event level denoted by the event identifier `evntidx`. Costs like the total expenditures and office-based costs are at the individual level denoted by the individual identifier `dupersid`. Hence, you will see that at the event level, the migraine-specific office-based costs `obxp21x` differs by events (`evntidx`. Conversely, the office-based costs are the same by individual (`dupersid`).

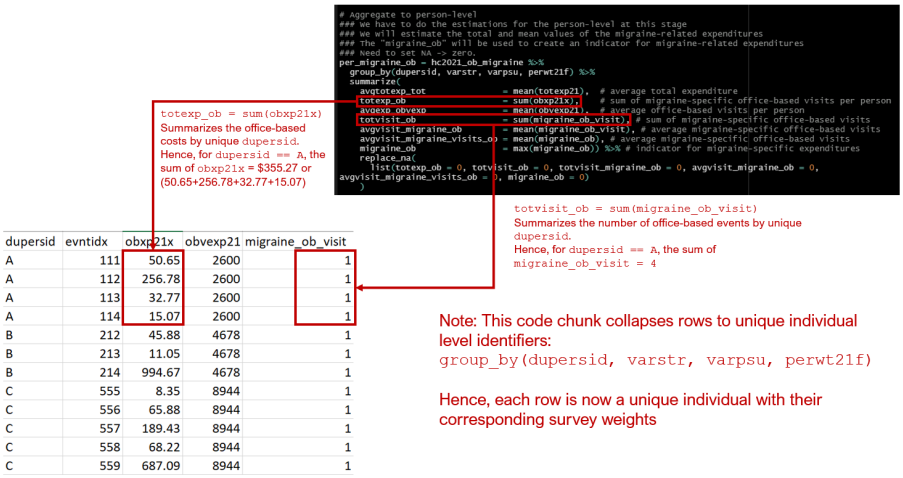
We need to sum or add up the migraine-specific office-based costs for each individual so that we can collapse the rows. We want to do create a new dataframe where each row is a unique individual denoted by their individual identifier (`dupersid`) and corresponding survey weights (`varstr`, `varpsu`, `perwt21f`).

Once this step is completed, you can check to see that the number of rows from the new dataframe (`per_migraine_ob`) is equal to the number of rows from the Full-Year Consolidated file (`hc2021`). They should be equal.

```
### QC: Should have the same number as the full-year consolidated file
nrow(per_migraine_ob) == nrow(hc2021)
```

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

Now that you've merged the migraine-specific office-based expenditures with the Full-Year Consolidated file, we can start on repeating this process for the migraine-specific inpatient expenditures.



dupersid	evntidx	obxp21x	obvexp21	migraine_ob_visit
A	111	50.65	2600	1
A	112	256.78	2600	1
A	113	32.77	2600	1
A	114	15.07	2600	1
B	212	45.88	4678	1
B	213	11.05	4678	1
B	214	994.67	4678	1
C	555	8.35	8944	1
C	556	65.88	8944	1
C	557	189.43	8944	1
C	558	68.22	8944	1
C	559	687.09	8944	1

Figure 4.3: Visual explanation of the code chunk

4.2.3 Part 3 - Migraine-specific inpatient expenditures

The code for the migraine-specific inpatient expenditures is similar to the migraine-specific office-based expenditures so I won't go through them step-by-step. However, there are several important differences.

First, the inpatient event type is `eventype == 4`, which needs to be included when you isolate the event types from the `clnk2021` file.

```
##############################################
## INPATIENT STAYS
##############################################
# Inpatient stays events
inpat_events = clnk2021 %>%
  filter(eventype == 4)

# QC: Should only have EVENTYPE == 4
inpat_events %>%
  count(eventype)
```

```
## # A tibble: 1 x 2
##   eventype                          n
##   <dbl+lbl>                     <int>
## 1 4 [4 INPATIENT HOSPITAL STAY]  3167
```

```r
# Merge migraine diagnosis file with inpatient stays file
migraine_lnk_inpat = inner_join(
  migraine, inpat_events,
  by = c("dupersid", "condidx")
)

# Select only DISTINCT events ("De-duplicate")
migraine_inpat_distinct = migraine_lnk_inpat %>%
  distinct(dupersid, evntidx, eventype)


# Merge inpatient stays file with distinct migraine-specific inpatient stays file
### Use the inner_join() to merge ob2021x and migraine_inpat_distinct dataframes
### Then create a new indicator variable called "migraine_inpat_stays = 1."
inpat_migraine = inner_join(
  inpat2021x, migraine_inpat_distinct) %>%
  mutate(migraine_inpat_stays = 1)


# Merge with full-year consolidated file
### Use the full_join() to keep all the rows from both dataframes.
### Then create a new object called hc2021_inpat_migraine
### & a new indicator variable called "migraine_inpat = 1"
### & another new indicator variable called "fyc = 1."
### Note: These are not unique rows (repeated dupersid)
hc2021_inpat_migraine = full_join(
  inpat_migraine %>% mutate(migraine_inpat = 1),
  hc2021x %>% mutate(fyc = 1)
)


# Aggregate to person-level
### We have to do the estimations for the person-level at this stage
### We will estimate the total and mean values of the migraine-related expenditures
### The "migraine_inpat" will be used to create an indicator for migraine-related expe
### Need to set NA -> zero.
per_migraine_inpat = hc2021_inpat_migraine %>%
  group_by(dupersid, varstr, varpsu, perwt21f) %>%
  summarize(
    totnights_inpat               = sum(numnighx),
    totexp_inpat                  = sum(ipxp21x),
    avgexp_iptexp                 = mean(iptexp21),
    totvisit_inpat                = sum(migraine_inpat_stays),
    avgvisit_inpat                = mean(migraine_inpat_stays),
    avgvisit_migraine_visits_inpat = mean(migraine_inpat),
```

```
    migraine_inpat                  = max(migraine_inpat)) %>%
  replace_na(
    list(totnights_inpat = 0, totexp_inpat = 0, totvisit_inpat = 0, avgvisit_inpat = 0, avgvisit_
  )

### QC: Should have the same number as the full-year consolidated file
nrow(per_migraine_inpat) == nrow(hc2021)
```

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

4.2.4 Part 4 - Combine office-based and inpatient expenditure files

Once you have the migraine-specific inpatient expenditures dataframe completed, you can start the process to combine this with the migraine-specific office-based expenditures dataframe.

First, we want to merge using the `left_join` function the migraine-specific office-based expenditure dataframe with the Full-Year Consolidated file `hc2021x`. We do this because there were a couple of variables in `hc2021x` that we would like to keep such as the sex (`sex`) and race (`racev1x`) variables. We will call this new dataframe `combined_data_ob_part`.

```
# COMBINE OBVISIT AND INPAT - SPECIFIC EXPENDITURE FILES
### Part 1: Combine the office-based visit exp file with the hc2021x file
combined_data_ob_part <- left_join(
  hc2021x,
  per_migraine_ob,
  by = c("dupersid", "varstr", "varpsu", "perwt21f")
)
```

Then, we want to merge the migraine-specific inpatient expenditures `per_migraine_inpat` to the dataframe `combined_data_ob_part` to create a new dataframe called `combined_data_ob_inpat_part`.

```
### Part 2: Combine the inpatient stays file with the "combined_data_ob_part" file
combined_data_ob_inpat_part <- left_join(
  combined_data_ob_part,
  per_migraine_inpat,
  by = c("dupersid", "varstr", "varpsu", "perwt21f")
)
```

Once that part is completed, we will have a dataframe that includes both the migraine-specific office-based visit and inpatient stay expenditures.

Next, we will take this opportunity to create an indicator variable for individuals with a migraine diagnosis or condition. So far, we have identified and isolated office-based and inpatient expenditures that were migraine-specific. This means that some individuals with a migraine diagnosis may not have accrued any migraine-specific office-based or inpatient expenditures. It's possible that they have other expenditures, but those may not be migraine-specific. Hence, it is important that we create an indicator variable for those individuals with migraine in the comprehensive dataframe.

```
### Part 3: Merge an indicator variable for migraine diagnosis.
### Not all patients with migraine has an expenditure.

### a) Create a dataframe with the dupersid and migraine indicator.
migraine_distinct_qc = migraine %>%
  distinct(dupersid) %>%
  mutate(migraine_indicator = 1)

### b) Merge "migraine_distinct_qc" to the larger table.
### This will be a 1 to 1 join.
combined_data_migraine <- left_join(
  combined_data_ob_inpat_part,
  migraine_distinct_qc,
  by = c("dupersid"))

### c) We also need to convert NA to 0 for the "migraine_indicator" variable.
combined_data_migraine = combined_data_migraine %>%
  replace_na(list(migraine_indicator = 0))
```

4.2.5 Part 5 - Descriptive analysis using survey weights

There are many ways to describe the expenditures in the migraine population.

First, we need to invoke the survey design for our data using the `svydesign` function. We will call this the `survey_cohort` design.

```
# Define person-level survey design
survey_cohort = svydesign(
  id = ~varpsu,
  strata = ~varstr,
  weights = ~perwt21f,
  data = combined_data_migraine,
  nest = TRUE)
```

Next, we can provide the average costs and amounts of office-based visit and inpatient stay expenditures for the whole cohort, both individuals with and

without a migraine condition. We will use the `svyby` function to group the findings into those with migraine `migraine_indicator == 1` and those without migraine `migraine_indicator == 0`.

Note: We expect to see zero costs and amount for the non-migraine individuals.

```
### Average migraine-specific office-based visit costs and amount
svyby(~totexp_ob, ~migraine_indicator, survey_cohort, svymean)
```

```
##   migraine_indicator   totexp_ob          se
## 0                  0   0.0000000  0.00000000
## 1                  1 473.7215498 63.42344309
```

```
svyby(~totvisit_ob, ~migraine_indicator, survey_cohort, svymean)
```

```
##   migraine_indicator totvisit_ob           se
## 0                  0 0.000000000 0.0000000000
## 1                  1 1.874864152 0.2637829887
```

```
### Average migraine-specific inpatient costs and nights stayed
svyby(~totexp_inpat, ~migraine_indicator, survey_cohort, svymean)
```

```
##   migraine_indicator totexp_inpat         se
## 0                  0   0.00000000  0.0000000
## 1                  1  69.78947691 37.9709726
```

```
svyby(~totnights_inpat, ~migraine_indicator, survey_cohort, svymean)
```

```
##   migraine_indicator totnights_inpat           se
## 0                  0   0.00000000000 0.0000000000
## 1                  1   0.02538119475 0.0133157815
```

Based on these findings, the average cost for migraine-specific office-based visits was $474 for individuals with a migraine condition. The average number of office-based events was 1.9 events per individual with a migraine condition.

The average migraine-specific inpatient stay expenditure was $70 per individual with a migraine condition. The average migraine-specific inpatient nights was 0.025 nights per individual with a migraine condition.

Among individual without a migraine condition, the averages would have been zero.

Alternatively, we could have done this using a subset of the migraine cohort, but this will require us to create a subset of individuals with a migraine condition, which we will call `migraine_cohort`.

```
# Subgroup of individuals with inpatient stays for migraines
migraine_cohort = subset(survey_cohort, migraine_indicator == 1)
```

Once we have the subset, we can provide the mean migraine-specific office-based and inpatient expenditures. Comparing the results from the subset to the whole cohort, we find that for individuals with the migraine indicator `migraine_indiciator == 1`, the average costs and amount of office-based and inpatient expenditures are the same.

```
### Average migraine-specific office-based visit costs and amount
svyby(~totexp_ob, ~migraine_indicator, migraine_cohort, svymean)
```

```
##   migraine_indicator   totexp_ob          se
## 1                  1 473.7215498 63.42344309
```

```
svyby(~totvisit_ob, ~migraine_indicator, migraine_cohort, svymean)
```

```
##   migraine_indicator totvisit_ob           se
## 1                  1 1.874864152 0.2637829887
```

```
### Average migraine-specific inpatient costs and nights stayed
svyby(~totexp_inpat, ~migraine_indicator, migraine_cohort, svymean)
```

```
##   migraine_indicator totexp_inpat         se
## 1                  1  69.78947691 37.9709726
```

```
svyby(~totnights_inpat, ~migraine_indicator, migraine_cohort, svymean)
```

```
##   migraine_indicator totnights_inpat           se
## 1                  1   0.02538119475 0.0133157815
```

The average values for individuals with a migraine condition should be exactly the same in the subset and the whole cohort.

But what if we are interested in migraine-specific expenditures for only those individuals with a migraine AND non-zero expenditures? This would mean that we will have EXCLUDE individual with a migraine condition and ZERO expenditures.

We can do this with another subset.

The first subset we will do is the migraine-specific office-based visit. We will subset our migraine group from the `per_migraine_ob` dataframe and call it `migraine_ob_nonzero` to reflect the non-zero expenditures of the migraine only cohort. We'll all this subset `migraineOB`.

```
# Define person-level survey design
migraine_ob_nonzero = svydesign(
  id = ~varpsu,
  strata = ~varstr,
  weights = ~perwt21f,
  data = per_migraine_ob,
  nest = TRUE)

# Subgroup of individuals with office-based for migraines
migraineOB = subset(migraine_ob_nonzero, migraine_ob == 1)
```

Once we have the new subset, we can estimate the average expenditures for migraine-specific office-based visits.

```
svymean(~totexp_ob, design = migraineOB) # Mean office-based expenditures
```

```
##                mean        SE
## totexp_ob 988.27575 126.33587
```

```
svymean(~totvisit_ob, design = migraineOB) # Mean migraine office-based visits
```

```
##                   mean     SE
## totvisit_ob 3.9113331 0.5329
```

Based on these findings, among individuals with a migraine condition and non-zero migraine-specific expenditures, the average migraine-specific office-based costs was $988 and the average number of migraine-specific office-based events was 3.9 events. This is very different from the $474 and 1.9 office-based events previously reported for the migraine population.

We can also do this for the inpatient expenditures subset, which we will call `migraineINPAT`.

```
# Define person-level survey design
migraine_inpat_nonzero = svydesign(
  id = ~varpsu,
  strata = ~varstr,
  weights = ~perwt21f,
  data = per_migraine_inpat,
  nest = TRUE)

# Subgroup of individuals with office-based for migraines
migraineINPAT = subset(migraine_inpat_nonzero, migraine_inpat == 1)

svymean(~totexp_inpat, design = migraineINPAT) # Mean inpatient stays expenditures
```

```
##                    mean        SE
## totexp_inpat 7364.0586 1138.0975
```

```
svymean(~totnights_inpat, design = migraineINPAT) # Mean inpatient stays nights
```

```
##                       mean      SE
## totnights_inpat 2.6781775 0.65583
```

Based on these findings, among individuals with a migraine condition and non-zero migraine-specific expenditures, the average migraine-specific inpatient stay costs was $7364 and the average number of nights stayed was 2.7 nights. This is very different from the $70 and 0.025 nights previously reported for the migraine population.

Depending on how you define your cohort, these averages will be different.

There are 756 individuals with a migraine condition, but only 353 of those with a migraine-specific office-based visit expenditure and 7 of those with a migraine-specific inpatient stay expenditure.

A summary of the findings is provided below:

Table 1. Summary of findings

Group	N	Office-based visit costs Mean ($), (SE)	Office-based visits events Mean (events), (SE)	Inpatient stay costs Mean ($), (SE)	Inpatient nights Mean (nights), (SE)
Migraine only	756	$474 (63)	1.9 (0.3)	$70 (38)	0.025 (0.013)
Migraine only and non-zero migraine-specific expenditures (Office-based visits)	353	$988 (126)	3.9 (0.5)	---	---
Migraine only and non-zero migraine-specific expenditures (Inpatient stays)	7	---	---	$7364 (1138)	2.7 (0.7)

Figure 4.4: Summary of findings.

4.3 Conclusions

We can link events with specific conditions to get a more precise estimate of the expenditures. In this motivating example, we linked the migraine condition to its events and estimated the average office-based visit and inpatient stay expenditures. But we also explored the differences in these average when the denominator is restricted further to those with non-zero condition-specific expenditures. This process can be generalized to other conditions and other types of events.

4.4 Acknowledgements

This tutorial would not be possible with the resources provided by AHRQ MEPS GitHub site. The resources are amazing, and the codes are available for Stata, R, and SAS. Each exercise provide a new perspective on how to leverage the MEPS dataset for anyone's research or investigations. I highly encourage people to visit their site.

4.5 Work in progress

This is a work in progress so expect some updates in the future.

4.6 Disclaimers

Any errors or mistakes are those of the author.

This is only for educational purposes.

This was built under R version 4.2.2 "Innocent and Trusting"

Chapter 5

Simple trend analysis with linear models

5.1 Introduction

Analyzing trends can be a tricky matter. You have to consider many things such as the autoregressive correlation between values across the time interval or the seasonal effects that occur and are unrelated to the risk factor. All these issues contribute to the difficulty and challenge of trend analysis. However, there are simple ways to perform rudimentary trend analysis with linear models that might be useful for most stakeholders.

Linear models are useful because they are easy to interpret. There is no re-transformation needed, and the outputs are interpreted in terms of real units. Non-linear models may require re-transformation or some kind of adjustment to get the β coefficients to be interpretable in real units.

Although there are a lot of different models that take into consideration the strength of the correlation between values across time (e.g., generalized estimating equation models) or the random slope and intercepts of subjects and groups, we will use the linear effects model to interpret the trends of healthcare expenditure of a representative sample of the US non-institutionalized patients.

5.2 Motivating example

We will use data from the Agency for Healthcare Research and Quality (AHRQ) Medical Expenditure Panel Survey (MEPS) data from 2016 to 2021. We will use R to perform the simple trend analysis.

5.2.1 Loading the libraries

There are several libraries that we'll need to install and then load.

```
### step 1a: Load the MEPS package
library("MEPS") ## You need to load the library every time you restart R

### Step 1b: Load the other libraries
library("survey")
library("foreign")
library("dplyr")
library("ggplot2")
library("questionr") # remotes::install_github("juba/questionr")
library("lspline")  # devtools::install_github("mbojan/lspline", build_vignettes=TRUE)
library("ggeffects") # remotes::install_github("strengejacke/ggeffects")
library("margins")
library("gtsummary") # remotes::install_github("ddsjoberg/gtsummary")
library("sjPlot") # plot marginal effects ("plot_model" function)
```

5.2.2 Loading data into the R environment

There are two ways to load the data onto the R environment from AHRQ MEPS.

```
# There are two ways to load data from AHRQ MEPS website:
#### Method 1: Load data from AHRQ MEPS website
hc2021 = read_MEPS(file = "h233")
hc2020 = read_MEPS(file = "h224")
hc2019 = read_MEPS(file = "h216")
hc2018 = read_MEPS(file = "h209")
hc2017 = read_MEPS(file = "h201")
hc2016 = read_MEPS(file = "h192")


#### Method 2: Load data from AHRQ MEPS website
hc2021 = read_MEPS(year = 2021, type = "FYC")
hc2020 = read_MEPS(year = 2020, type = "FYC")
hc2019 = read_MEPS(year = 2019, type = "FYC")
hc2018 = read_MEPS(year = 2018, type = "FYC")
hc2017 = read_MEPS(year = 2017, type = "FYC")
hc2016 = read_MEPS(year = 2016, type = "FYC")
```

Once the data have been loaded onto R, we can made some edits. The first edit I make is ensure that all column or variable names are in lower case.

```
## Change column names to lowercase
names(hc2021) <- tolower(names(hc2021))
names(hc2020) <- tolower(names(hc2020))
names(hc2019) <- tolower(names(hc2019))
names(hc2018) <- tolower(names(hc2018))
names(hc2017) <- tolower(names(hc2017))
names(hc2016) <- tolower(names(hc2016))
```

5.2.3 Download the linkage file

Next, we need to download the pooled linkage file. This has the updated primary sampling unit and strata for the individual respondents. This is an important file because when we pool MEPS data from different years, the primary sampling unit and strata will change. We will need to merge this with our pooled data eventually, which will be discussed later in the tutorial.

```
# We need the linkage file with the appropriate stratum of the primary sampling strata (STRA9621)
linkage = read_MEPS(type = "Pooled linkage")
names(linkage) <- tolower(names(linkage)) # change variable name to lower case
```

5.2.4 Creating the pooled data file

Now that we have the Full-Year Consolidated files from 2016 to 2021 and the pooled linkage files loaded onto the R environment, we can being to clean the data and merge them together.

We'll start by keeping only those variables that will be needed for our analysis. These include the following variables:

- `dupersid`: The unique identifier of the individual respondent
- `panel`: The panel when the individual entered the MEPS round
- `varstr`: The individual sampling strata
- `varpsu`: The individual primary sampling unit
- `sex`: The sex variable (1 = MALE, 2 = FEMALE)
- `totexp`: The total healthcare costs per individual per year (Note: This variable name was changed from the year-specific total expenditure (e.g., `totexp17f` to `totexp` because we are pooling from multiple years)
- `perwt`: The person weight for the individual in the sample (Note: This variable name was changed from the year-specific person weight (e.g., `perwt7f`) to `perwt` because we are pooling from multiple years)

We also created an indicator variable `year` to reflect the year the Full-Year Consolidated file was captured.

When we merge all the data, we created a new variable `poolwt` where we divide the `perwt` by the number of years used in the pooled data (e.g., 6). This will generate a new individual person weight with the pooled data.

Note: For more instruction and information about pooling multiple years with MEPS data, please see the 1996-2015 Pooled Linkage Variance Estimation File. This document provides a good summary and documentation on creating a `poolwt` variable and using the pooled strata and primary sampling unit.

```
# Select specific variables

### 2021
hc2021p = hc2021 %>%
  rename(
    perwt = perwt21f,
    totexp = totexp21) %>%
  select(
    dupersid,
    panel,
    varstr,
    varpsu,
    perwt,
    sex,
    totexp)
hc2021p$year <- 2021

### 2020
hc2020p = hc2020 %>%
  rename(
    perwt = perwt20f,
    totexp = totexp20) %>%
  select(
    dupersid,
    panel,
    varstr,
    varpsu,
    perwt,
    sex,
    totexp)
hc2020p$year <- 2020


### 2019
hc2019p = hc2019 %>%
```

```
  rename(
    perwt = perwt19f,
    totexp = totexp19) %>%
  select(
    dupersid,
    panel,
    varstr,
    varpsu,
    perwt,
    sex,
    totexp)
hc2019p$year <- 2019

### 2018
hc2018p = hc2018 %>%
  rename(
    perwt = perwt18f,
    totexp = totexp18) %>%
  select(
    dupersid,
    panel,
    varstr,
    varpsu,
    perwt,
    sex,
    totexp)
hc2018p$year <- 2018

### 2017
hc2017p = hc2017 %>%
  rename(
    perwt = perwt17f,
    totexp = totexp17) %>%
  select(
    dupersid,
    panel,
    varstr,
    varpsu,
    perwt,
    sex,
    totexp)
hc2017p$year <- 2017

### 2016
hc2016p = hc2016 %>%
```

```
  rename(
    perwt = perwt16f,
    totexp = totexp16) %>%
  select(
    dupersid,
    panel,
    varstr,
    varpsu,
    perwt,
    sex,
    totexp)
hc2016p$year <- 2016


# Merge data and adjust the person weight by 6 years
pool_data = bind_rows(hc2021p,
                      hc2020p,
                      hc2019p,
                      hc2018p,
                      hc2017p,
                      hc2016p) %>%
  mutate(poolwt = perwt / 6)
```

5.2.5 Merging with pooled survey primary sampling strata and unit

Next, we need to merge the pooled years file with the updated primary sampling strata and unit from the linkage file. The linkage file is updated every year. Since the latest Full-Year Consolidated Data file that we are using includes 2021, we are using the 2021 linkage file (`HC-229I`).

We first limit the linkage file to only include the following variables:

- `dupersid`: The unique identifier for the individual respondent (Note: We will need this to merge the files later)
- `panel`: The panel when the individual entered the MEPS round (Note: We will need to use this to merge the files later)
- `stra9621`: The pooled sampling strata
- `psu9621`: The pooled primary sampling unit

```
# Reduce the linkage file to only include dupersid, panel, stra9621, psu9621
linkage_file = linkage %>%
  select(dupersid, panel, stra9621, psu9621)

# Merge link file with main data
pool_data_linked = left_join(pool_data,
                             linkage_file,
                             by = c("dupersid", "panel"))
```

5.2.6 Create factor variables

We need to create factor variables so that we can evaluate the marginal effects. Disclaimer: I'm not sure if this is always necessary. However, I run into issues with the `margins` package when I estimated the average marginal effects in R. I get errors about the size of the dimension or vector, which indicates that the dimensions of the dataframe is not correct. I'm not sure why this error occurs, but I seem to have avoided it when I convert my categorical variables into factors. I'm not sure if this is a bug in the `margins` package or something I did with the terms in the regression model, but I'll monitor and update this tutorial if there are any changes or discoveries. For now, we'll convert our categorical variables into factors. Here, I also convert the variable `year` into a factor.

```
# Create factor variables:
### MALE
pool_data_linked$male[pool_data_linked$sex == 1] = 1
pool_data_linked$male[pool_data_linked$sex == 2] = 0
table(pool_data_linked$male)
```

```
##
##     0     1
## 95023 86626
```

```
pool_data_linked$male <- factor(pool_data_linked$male, levels = c(0, 1))
class(pool_data_linked$male)
```

```
## [1] "factor"
```

```
levels(pool_data_linked$male)
```

```
## [1] "0" "1"
```

```
### YEAR
pool_data_linked$year <- factor(pool_data_linked$year, levels = c(2016, 2017, 2018, 201
class(pool_data_linked$year)
```

```
## [1] "factor"
```

```
levels(pool_data_linked$year)
```

```
## [1] "2016" "2017" "2018" "2019" "2020" "2021"
```

5.2.7 Set the survey options

Next, we set the survey options so that we can estimate the survey-weighted values for our population.

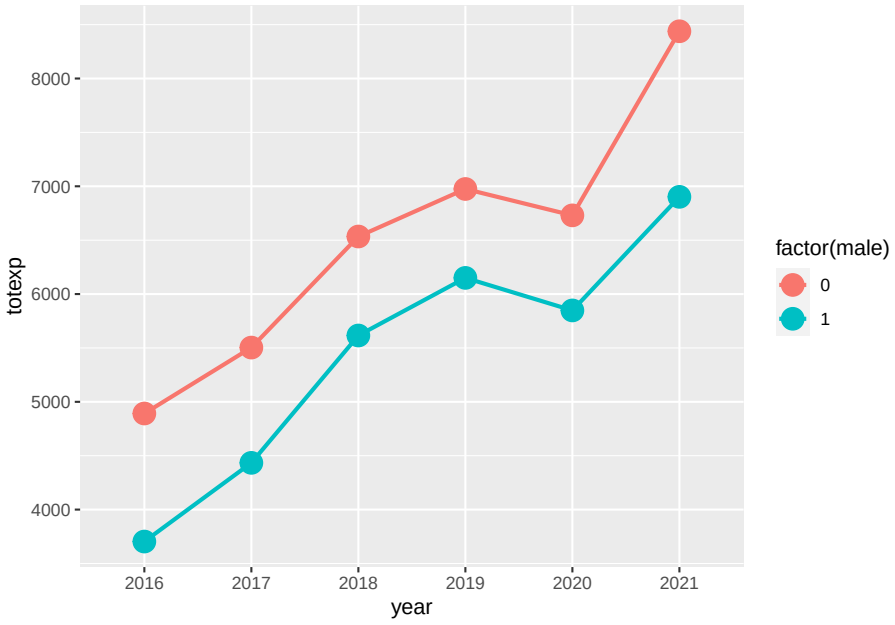
```
################################## SURVEY ###################################

# Set the survey options
options(survey.lonely.psu = "adjust")
options(survey.adjust.domain.lonely = TRUE)


# Define the survey design
survey_design = svydesign(
  id = ~psu9621,
  strata = ~stra9621,
  weights = ~poolwt,
  data = pool_data_linked,
  nest = TRUE)
```

5.2.8 Trend analysis - Plot the total healthcare costs by sex across time

We can look at the total healthcare costs `totexp` by `sex` across time `year` with the `ggsurvey` function, which is part of the `questionr` package.

Visualizing the trends, females have a higher average total healthcare costs compared to males, and these are increasing across time. (Note: This trend was generated with the survey weights.)

```
################################ TREND ANALYSIS ################################

# Plot total expenditures over time (Load the "questionr" package to use "ggsurvey")
ggsurvey(survey_design) +
  aes(x = year, y = totexp, group = factor(male), color = factor(male)) +
  stat_summary(fun = "mean", geom = "point", size = 5) +
  stat_summary(fun = "mean", geom = "line", size = 1)
```



5.2.9 Regression model

We construct a linear regression model using the `svyglm` function in R. We included the terms `sex` and `year`, and we included an interaction term `sex:year`. This interaction term provides the estimates that describe the differences in the healthcare total costs between males and females when the years changes.

```
# Survey-weight GLM model with interaction term
survey_model <- svyglm(totexp ~ factor(male) + factor(year) + factor(male):factor(year), survey_d
```

We can view the estimates with their corresponding 95% confidence intervals (CIs).

```
# Summary table with 95% CI
round(
  cbind(
    summary(survey_model, df.resid = degf(survey_model$survey.design))$coef,
    confint(survey_model, df       = degf(survey_model$survey.design))
  ), 4)
```

```
##                                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)                        5632.6497   153.3156 36.7389   0.0000
## factor(male)1                     -1280.8517   183.1791 -6.9923   0.0000
## factor(year)2017                    164.5324   200.4297  0.8209   0.4121
## factor(year)2018                    981.6884   231.2365  4.2454   0.0000
## factor(year)2019                   1103.9018   220.8327  4.9988   0.0000
## factor(year)2020                   1022.3480   255.6661  3.9988   0.0001
## factor(year)2021                   1964.5098   279.9440  7.0175   0.0000
## factor(male)1:factor(year)2017      276.5807   250.6373  1.1035   0.2704
## factor(male)1:factor(year)2018      154.8177   277.8425  0.5572   0.5776
## factor(male)1:factor(year)2019      291.3151   276.8655  1.0522   0.2932
## factor(male)1:factor(year)2020      487.1318   353.7995  1.3769   0.1692
## factor(male)1:factor(year)2021      -65.6524   403.7881 -0.1626   0.8709
##                                       2.5 %    97.5 %
## (Intercept)                        5331.3906 5933.9088
## factor(male)1                     -1640.7914 -920.9121
## factor(year)2017                   -229.3041  558.3688
## factor(year)2018                    527.3179 1436.0589
## factor(year)2019                    669.9743 1537.8294
## factor(year)2020                    519.9744 1524.7217
## factor(year)2021                   1414.4311 2514.5886
## factor(male)1:factor(year)2017     -215.9115  769.0730
## factor(male)1:factor(year)2018     -391.1317  700.7671
## factor(male)1:factor(year)2019     -252.7145  835.3448
## factor(male)1:factor(year)2020     -208.0701 1182.3338
## factor(male)1:factor(year)2021     -859.0800  727.7751
```

A nicer way of presenting the regression model output is with the `tbl_regression` function from the `gtsummary` package.

```
# Use gtsummary to generate a clean summary table.
survey_model %>%
  tbl_regression(intercept = FALSE,
                 estimate_fun = function(x) style_sigfig(x, digits = 3),
                 pvalue_fun = function(x) style_sigfig(x, digits = 3)) %>%
  modify_caption("Survey-weighted linear regression")
```

From these findings, we can make several interpretations.

Table 5.1: Survey-weighted linear regression

Characteristic	**Beta**	**95% CI**	**p-value**
factor(male)			
0	—	—	
1	-1,281	-1,641, -921	0.000
factor(year)			
2016	—	—	
2017	165	-229, 558	0.412
2018	982	527, 1,436	0.000
2019	1,104	670, 1,538	0.000
2020	1,022	520, 1,525	0.000
2021	1,965	1,414, 2,515	0.000
factor(male) * factor(year)			
1 * 2017	277	-216, 769	0.270
1 * 2018	155	-391, 701	0.578
1 * 2019	291	-253, 835	0.293
1 * 2020	487	-208, 1,182	0.169
1 * 2021	-65.7	-859, 728	0.871

The most important estimates are the interaction terms. These provide the estimation or the change in healthcare total costs between males and females due to an increase in the year. For instance, the interaction term `factor(male)1:factor(year)2017` is interpreted as the average difference in the change in total healthcare costs between males and females when the year increased from 2016 to 2017. This difference in the changes in total healthcare costs between males and females is $276.58 (95% CI: -$215.91, $769.07), which was not statistically significant.

5.2.10 Average marginal effects between males and females at each year

We can generate the average marginal effects or the differences in total healthcare costs between males and females at each year using the `margins` function.

For instance, the difference in total healthcare costs between males and females in 2019 was -$990; 95% CI: -$1,396, -$583, which was statistically significant. This means that males had lower average total healthcare costs compared to females in 2019.

```
# Equivalent to Stata's margin, dydx(year), at(male = 0:1)
margins1 <- margins(survey_model, type = "response", design = survey_design, at = list(year = c(2
summary(margins1)
```

```
##  factor      year        AME       SE       z      p      lower     upper
##   male1 2016.0000 -1280.8517 183.1789 -6.9924 0.0000 -1639.8757 -921.8278
##   male1 2017.0000 -1004.2710 188.0103 -5.3416 0.0000 -1372.7644 -635.7776
##   male1 2018.0000 -1126.0341 208.9060 -5.3901 0.0000 -1535.4823 -716.5859
##   male1 2019.0000  -989.5366 207.6048 -4.7664 0.0000 -1396.4346 -582.6386
##   male1 2020.0000  -793.7199 302.6867 -2.6222 0.0087 -1386.9749 -200.4649
##   male1 2021.0000 -1346.5042 359.8470 -3.7419 0.0002 -2051.7913 -641.2170
```

5.2.11 Average marginal effects between years for males and females

Alternatively, we can estimate the average marginal effects or the changes in total healthcare costs between years for males and females.

For instance, in 2021, males had an average total healthcare cost of $1965; 95% CI: $1,416, $2,513, and females had an average total healthcare cost of $1,899; 95% CI: $1277, $2521.

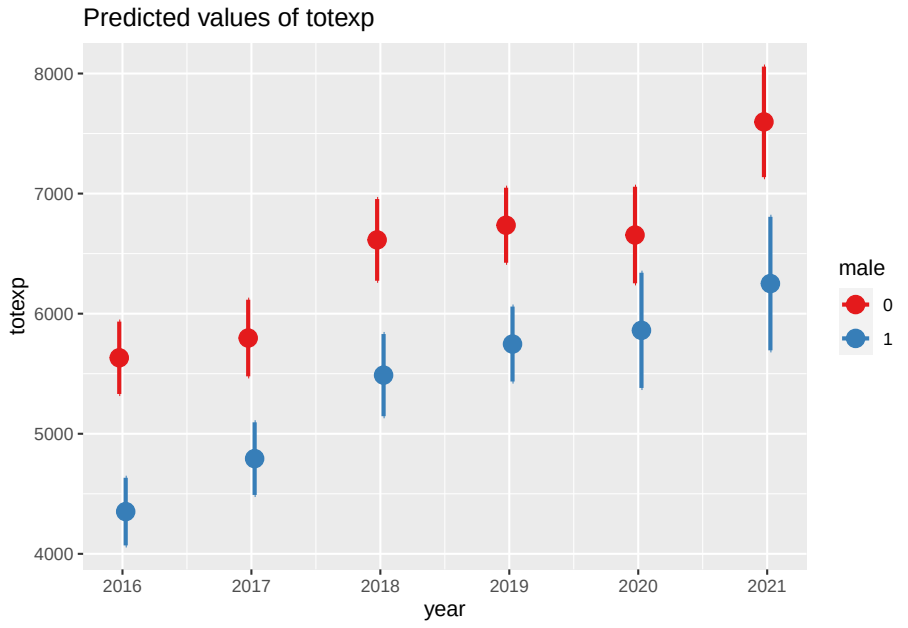
```
# Equivalent to Stata's margin, dydx(year), at(male = 0:1)
margins2 <- margins(survey_model, type = "response", design = survey_design, at = list
summary(margins2)
```

```
##    factor   male       AME       SE      z      p     lower     upper
##  year2017 0.0000  164.5324 200.4294 0.8209 0.4117 -228.3021  557.3669
##  year2017 1.0000  441.1131 190.2654 2.3184 0.0204   68.1998  814.0265
##  year2018 0.0000  981.6884 231.2367 4.2454 0.0000  528.4728 1434.9040
##  year2018 1.0000 1136.5061 225.5083 5.0398 0.0000  694.5180 1578.4942
##  year2019 0.0000 1103.9018 220.8324 4.9988 0.0000  671.0782 1536.7254
##  year2019 1.0000 1395.2170 213.9435 6.5214 0.0000  975.8953 1814.5386
##  year2020 0.0000 1022.3480 255.6657 3.9988 0.0001  521.2525 1523.4436
##  year2020 1.0000 1509.4799 283.0927 5.3321 0.0000  954.6284 2064.3314
##  year2021 0.0000 1964.5098 279.9436 7.0175 0.0000 1415.8305 2513.1891
##  year2021 1.0000 1898.8574 317.1989 5.9863 0.0000 1277.1590 2520.5558
```

5.2.12 Plot the average marginal effects

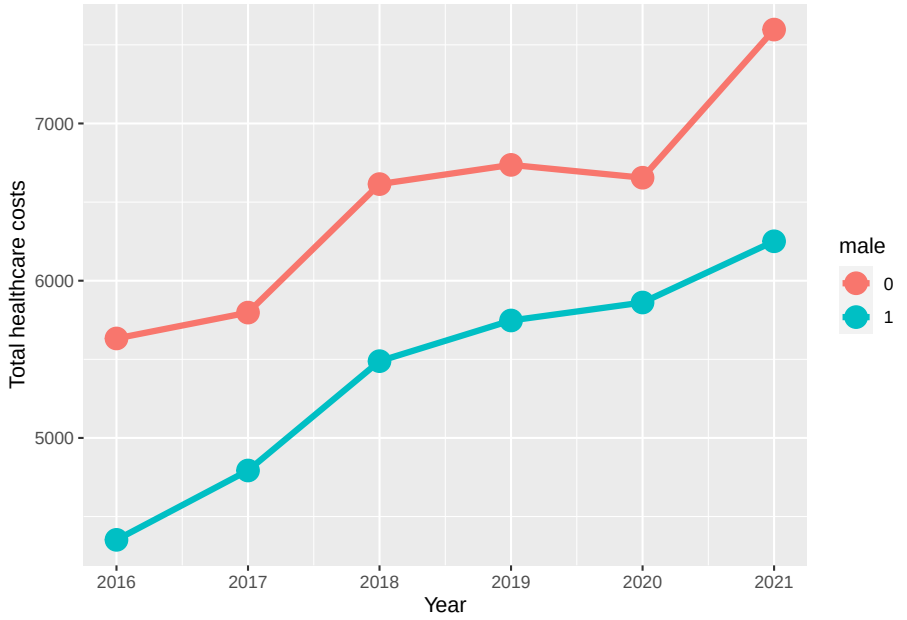
We can plot the average marginal effects between males and females across time `year`. We can visualize the average total healthcare costs for males and females at each year from 2016 to 2021.

```
# Plot the marginal effects
plot_model(survey_model, type = "pred", terms = c("year", "male"), dot.size = 4., line
```



We could also plot the average marginal effects using `ggplot2` with the `ggpredict` package.

```
# Plot the predicted values
fit.dataframe <- ggpredict(survey_model, terms = c("year", "male"), design = survey_design)
ggplot(fit.dataframe, aes(x, predicted, colour = group)) +
  geom_line(size = 1.5) +
  geom_point(size = 5) +
  labs(
    x = "Year",
    y = "Total healthcare costs",
    colour = get_legend_title(fit.dataframe)
  )
```



There are a few important things to plot with these figures.

In the first figure below, we see that the slopes or change in average total healthcare costs across the years are displayed for the males and females. The output from the `margins` function provides us with the within group change in total healthcare costs for males and females across time. For instance, the change from 2016 to 2017 for females (which is also called the slope between 2016 and 2017) was $165. For males, their slope was $441.

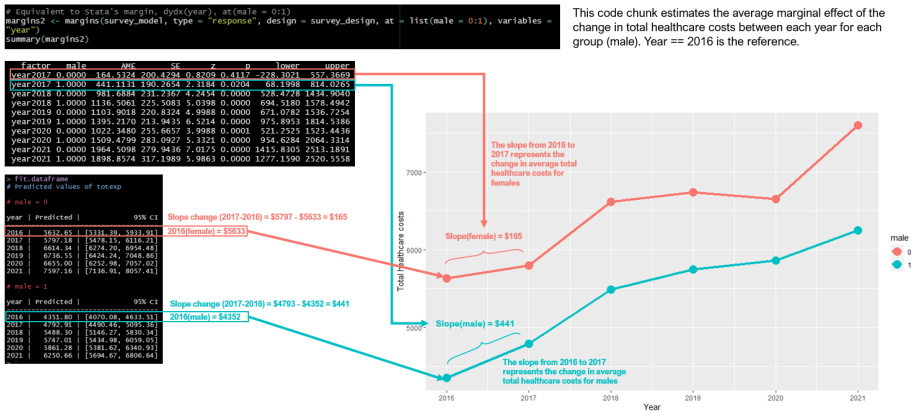


Figure 5.1: Average marginal effects or the Slopes of males and females.

In the second figure, we can use the output from the `margins1` object to estimate

the difference in total healthcare costs at each year between males and females. For instance, we can visualize the difference in total healthcare costs at 2019 between males and females is -$990.

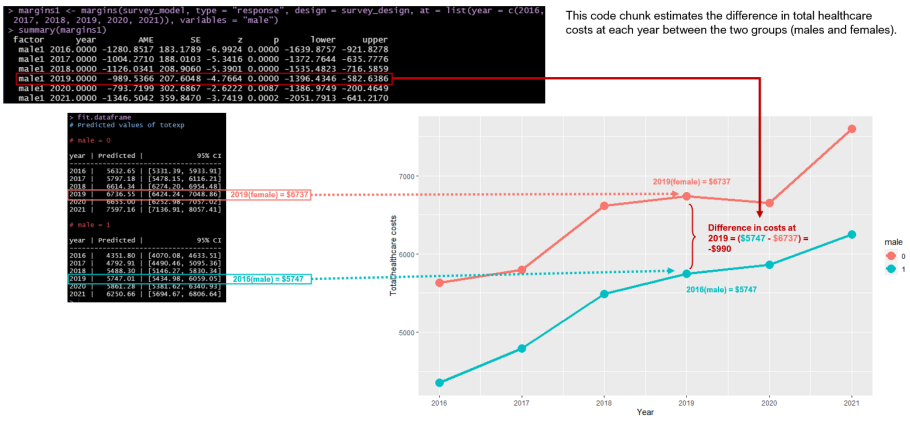


Figure 5.2: Differences in total healthcare costs between males and females in 2019.

The last figure is the most important. This visualizes the interaction term results. We are mostly interested in the interaction terms for trend analysis. The interaction terms tell us the differences in the total healthcare costs between males and females as the year changes (2016 to 2018).

For instance, the slope change for females between 2018 and 2016 was approximately $981. The slope change for males between 2018 and 2016 was $1136. The difference between these two slope changes for males and females is $1136 - $981 = $155. This is our difference in the changes estimation, or we can state that this is the differences in total healthcare costs between males and females as the year changed from 2016 to 2018. But, this was not statistically significant since the 95% CI includes zero (95% CI: -$391, $701).

5.3 Conclusions

Trend analysis can be performed using linear models. The ease of interpretation and simple execution makes this ideal for most stakeholders. Moreover, the `svyglm` function allows us to incorporate the survey weights from the complex survey design of MEPS to generate standard errors that were representative of the general U.S. population. Although the `margins` package is helpful in estimating the average marginal effects, there are other alternatives such as the `marginaleffects` package.

Additionally, more complex models exist that account for autoregressive correlations and more precise estimations of the variance. However, they are harder

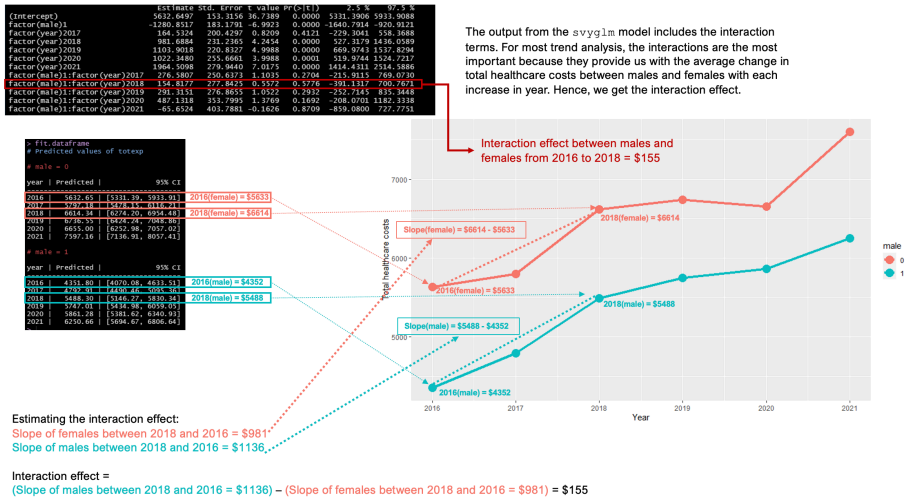


Figure 5.3: Interaction terms - difference in the changes between males and females when year increased from 2016 to 2018.

to implement with complex survey designs. Unfortunately, I have not been able to figure out how to perform some of these more complex longitudinal data analysis with alternative models (e.g., linear mixed effects models) that incorporate the survey weights. For these scenarios, I prefer to use `Stata`, which makes it convenient to apply survey weights in more complex models. Perhaps, I'll create some tutorials using `Stata` for these more complex longitudinal models in the future.

5.4 Acknowledgements

I am grateful to the author of the `margins` package, Thomas J. Leeper. You can find his `margins` package on his `GitHub site`.

Additionally, the `marginaleffects` package by Vincent Arel-Bundock has helped me to compare some of `R`'s marginal effects commands with `Stata`. You can find his GitHub site here.

5.5 Work in progress

This is a work in progress so expect some updates in the future.

5.6 Disclaimers

Any errors or mistakes are those of the author.

This is only for educational purposes.

This was built under R version 4.2.2 “Innocent and Trusting”

Chapter 6

Interrupted time series analysis (ITSA) with R: A short tutorial

6.1 Introduction

Previously, I wrote a tutorial on how to perform an interrupted time series analysis (ITSA) in `Stata`, which is located on my RPubs site (link). This had me thinking of how to perform an ITSA using `R`. So, I decided to create a short tutorial on how to perform an ITSA on `R` using the Agency for Healthcare Research and Quality (AHRQ) Medical Expenditure Panel Survey (MEPS) dataset. I tried several ways to do this, and I think I've found a nice approach that leverages survey weights from the MEPS database. Although I'm writing this ITSA tutorial with this approach, I plan on researching other methods to perform an ITSA in `R` so stay tuned.

This short tutorial will summarize how to perform an ITSA using a linear regression model with survey weights applied from the MEPS database. (Note: I previously wrote a tutorial on how to apply weights from MEPS data (link).

6.2 Motivating example - Total Healthcare Costs from 2016 to 2021 by sex

We'll use the MEPS database to compare the trends in total healthcare costs between sex. We'll assume that a policy or intervention was implemented in

2019, which may have changed the healthcare costs levels and trends between males and females.

I pooled data from 2016 to 2021 and adjusted the weights. I won't go into details on how to create this dataset for this tutorial since I provided a detailed summary in a previous tutorial. For more details on how I created this dataset, please refer to my previous tutorial (link).

```
# To install MEPS package in R, you need to do a couple of things.
### Step 1: Install the "devtools" package.
#install.packages("devtools")

### Step 2: Install the "MEPS" package from the AHRQ MEPS GitHub site.
#devtools::install_github("e-mitchell/meps_r_pkg/MEPS")

### step 3: Load the MEPS package
library("MEPS") ## You need to load the library every time you restart R

### Step 4: Load the other libraries
library("survey")
library("foreign")
library("dplyr")
library("ggplot2")
library("questionr") # remotes::install_github("juba/questionr")
library("lspline")  # devtools::install_github("mbojan/lspline", build_vignettes=TRUE)
library("ggeffects") # remotes::install_github("strengejacke/ggeffects")
library("margins")


# There are two ways to load data from AHRQ MEPS website:
#### Method 1: Load data from AHRQ MEPS website
hc2021 = read_MEPS(file = "h233")
hc2020 = read_MEPS(file = "h224")
hc2019 = read_MEPS(file = "h216")
hc2018 = read_MEPS(file = "h209")
hc2017 = read_MEPS(file = "h201")
hc2016 = read_MEPS(file = "h192")


#### Method 2: Load data from AHRQ MEPS website
hc2021 = read_MEPS(year = 2021, type = "FYC")
hc2020 = read_MEPS(year = 2020, type = "FYC")
hc2019 = read_MEPS(year = 2019, type = "FYC")
hc2018 = read_MEPS(year = 2018, type = "FYC")
hc2017 = read_MEPS(year = 2017, type = "FYC")
hc2016 = read_MEPS(year = 2016, type = "FYC")
```

```
## Change column names to lowercase
names(hc2021) <- tolower(names(hc2021))
names(hc2020) <- tolower(names(hc2020))
names(hc2019) <- tolower(names(hc2019))
names(hc2018) <- tolower(names(hc2018))
names(hc2017) <- tolower(names(hc2017))
names(hc2016) <- tolower(names(hc2016))


# We need the linkage file with the appropriate stratum of the primary sampling unit (STRA9621) c
linkage = read_MEPS(type = "Pooled linkage")
names(linkage) <- tolower(names(linkage)) # change variable name to lower case



# Select specific variables

### 2021
hc2021p = hc2021 %>%
  rename(
    perwt = perwt21f,
    totexp = totexp21,
    ertexp = ertexp21) %>%
  select(
    dupersid,
    panel,
    varstr,
    varpsu,
    perwt,
    sex,
    totexp,
    ertexp)
hc2021p$year <- 2021

### 2020
hc2020p = hc2020 %>%
  rename(
    perwt = perwt20f,
    totexp = totexp20,
    ertexp = ertexp20) %>%
  select(
    dupersid,
    panel,
    varstr,
```

```
    varpsu,
    perwt,
    sex,
    totexp,
    ertexp)
hc2020p$year <- 2020


### 2019
hc2019p = hc2019 %>%
  rename(
    perwt = perwt19f,
    totexp = totexp19,
    ertexp = ertexp19) %>%
  select(
    dupersid,
    panel,
    varstr,
    varpsu,
    perwt,
    sex,
    totexp,
    ertexp)
hc2019p$year <- 2019

### 2018
hc2018p = hc2018 %>%
  rename(
    perwt = perwt18f,
    totexp = totexp18,
    ertexp = ertexp18) %>%
  select(
    dupersid,
    panel,
    varstr,
    varpsu,
    perwt,
    sex,
    totexp,
    ertexp)
hc2018p$year <- 2018

### 2017
hc2017p = hc2017 %>%
  rename(
```

```
    perwt = perwt17f,
    totexp = totexp17,
    ertexp = ertexp17) %>%
  select(
    dupersid,
    panel,
    varstr,
    varpsu,
    perwt,
    sex,
    totexp,
    ertexp)
hc2017p$year <- 2017

### 2016
hc2016p = hc2016 %>%
  rename(
    perwt = perwt16f,
    totexp = totexp16,
    ertexp = ertexp16) %>%
  select(
    dupersid,
    panel,
    varstr,
    varpsu,
    perwt,
    sex,
    totexp,
    ertexp)
hc2016p$year <- 2016


# Merge data and adjust the person weight by 6 years
pool_data = bind_rows(hc2021p,
                      hc2020p,
                      hc2019p,
                      hc2018p,
                      hc2017p,
                      hc2016p) %>%
  mutate(poolwt = perwt / 6)


# Add a new variable for the pre-post index year at 2019
pool_data$period[pool_data$year >= 2016 & pool_data$year < 2019] = 0
pool_data$period[pool_data$year >= 2019 & pool_data$year < 2022] = 1
```

```
head(pool_data)
```

```
## # A tibble: 6 x 11
##   dupersid   panel            varstr varpsu perwt sex          totexp ertexp
##   <chr>      <dbl+lbl>         <dbl>  <dbl> <dbl> <dbl+lbl>     <dbl>  <dbl>
## 1 2320005101 23 [23 PANEL 23]   2079      1 6785. 2 [2 FEMALE]   4908      0
## 2 2320005102 23 [23 PANEL 23]   2079      1 6177. 1 [1 MALE]    21257      0
## 3 2320006101 23 [23 PANEL 23]   2028      1 1599. 2 [2 FEMALE]    827      0
## 4 2320006102 23 [23 PANEL 23]   2028      1 1649. 1 [1 MALE]        0      0
## 5 2320006103 23 [23 PANEL 23]   2028      1 2892. 1 [1 MALE]        0      0
## 6 2320012102 23 [23 PANEL 23]   2069      2 1273. 2 [2 FEMALE]   9813      0
##    year poolwt period
##   <dbl>  <dbl>  <dbl>
## 1  2021  1131.      1
## 2  2021  1029.      1
## 3  2021   267.      1
## 4  2021   275.      1
## 5  2021   482.      1
## 6  2021   212.      1
```

```
# Reduce the linkage file to only include dupersid, panel, stra9621, psu9621
linkage_file = linkage %>%
  select(dupersid, panel, stra9621, psu9621)

# Merge link file with main data
pool_data_linked = left_join(pool_data,
                             linkage_file,
                             by = c("dupersid", "panel"))
```

6.3 Interrupted Time Series Analysis Design

In this ITSA motivating example, we are going to compare the trends in healthcare total costs between males and females before and after the hypothetical policy intervention, which was implemented in 2019. The ITSA design is illustrated in the figure below.

For this example, the ITSA model is denoted by the following:

$Y_i = \beta_0 + \beta_1(T_i) + \beta_2(X_i) + \beta_3(T_i * X_i) + \beta_4(Sex_i) + \beta_5(Sex_i * T_i) + \beta_6(Sex_i * X_i) + \beta_7(Sex_i * T_i * X_i) + \epsilon_i,$

where Sex_i is a new variable that denotes the groups (e.g., sexes), β_4 denotes the difference in the outcomes between the groups (e.g., sexes) at beginning of the study ($T = 0$), β_5 denotes the difference in the slopes between the groups

before the intervention, β_6 denotes the difference in the level changes between the groups immediately after implementation of the intervention X, and β_7 denotes the difference in the slopes after and before the intervention between the groups (e.g., difference-in-differences).

$$Y_i = \beta_0 + \beta_1 (T_i) + \beta_2 (X_i) + \beta_3 (T_i X_i) + \beta_4 (Sex_i) + \beta_5 (Sex_i T_i) + \beta_6 (Sex_i X_i) + \beta_7 (Sex_i T_i X_i) + \epsilon_i$$

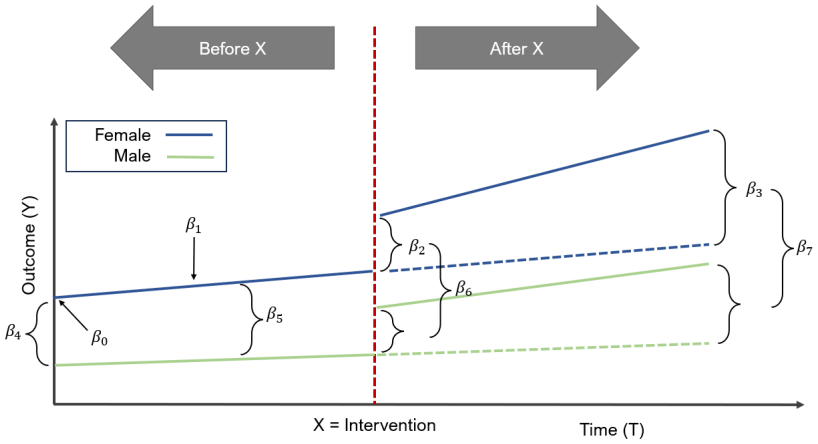


Figure 6.1: ITSA for two groups ('sex') before and after the intervention ('X').

6.4 Set up the survey options

In order to estimate the appropriate costs and standard errors, we will need to set up the survey options and survey design using the weights from MEPS. We will use the `svydesign` function to load the primary sampling unit, individual weights, and strata, which we will then assign to an object called `survey_design`. Please see the code below:

```
# Set the survey options
options(survey.lonely.psu = "adjust")
options(survey.adjust.domain.lonely = TRUE)


# Define the survey design
survey_design = svydesign(
                          id = ~psu9621,
                          strata = ~stra9621,
                          weights = ~poolwt,
                          data = pool_data_linked,
                          nest = TRUE)
```

6.5 Descriptive analysis

Once that's done, we can do some descriptive analysis.

Using the `svytable` function, we can estimate the weighted-sample of males and females in the United States (US). In our MEPS dataset, there are 160,228,773 males and 166,678,038 females, which is representative of the non-institutionalized civilian US population.

```
svytable(~sex, design = survey_design)
```

```
## sex
##           1           2
## 160228772.9 166678038.1
```

Moreover, we can look at the survey-weighted total healthcare costs between males and females. For instance, in 2016, the average annual total healthcare costs for males and females were $4351 and $5633 per person, respectively.

```
df1 <- svyby(~totexp, ~year+sex, survey_design, svymean)
df1
```

```
##        year sex      totexp          se
## 2016.1 2016   1 4351.797969 143.3686740
## 2017.1 2017   1 4792.911089 153.9208333
## 2018.1 2018   1 5488.304037 174.0676798
## 2019.1 2019   1 5747.014931 158.7998588
## 2020.1 2020   1 5861.277852 244.1047619
## 2021.1 2021   1 6250.655387 282.9494481
## 2016.2 2016   2 5632.649717 153.3156133
## 2017.2 2017   2 5797.182100 162.3600394
## 2018.2 2018   2 6614.338104 173.1029815
## 2019.2 2019   2 6736.551546 158.9384299
## 2020.2 2020   2 6654.997756 204.6000543
## 2021.2 2021   2 7597.159563 234.2313172
```

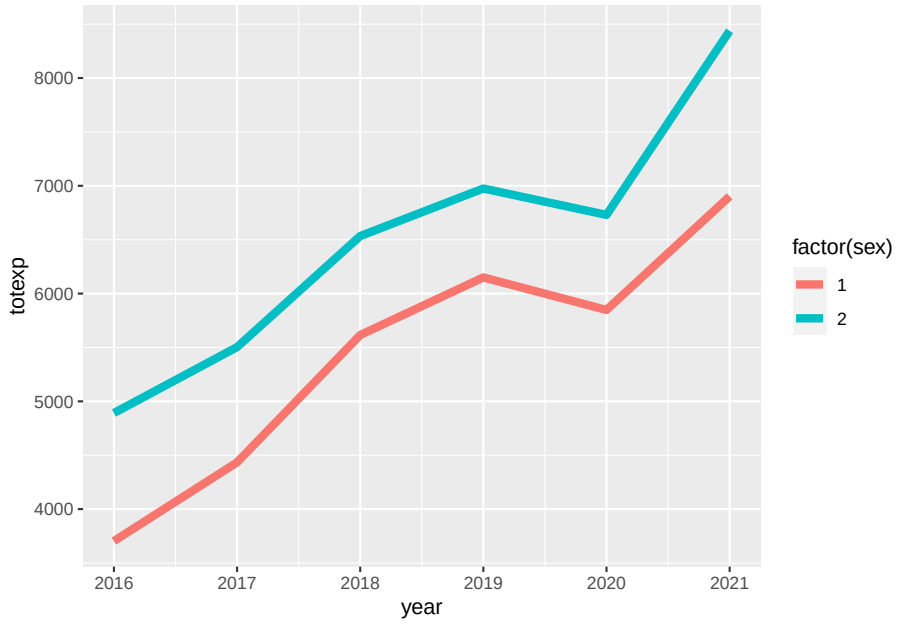
We can also look at the survey-weighted average cost before and after the intervention (before and after 2019). For instance, before the intervention the survey-weighted average total healthcare cost for males and females were $4880 and $6016, respectively. After the intervention, the survey-weighted average total healthcare costs for males and females were $5954 and $6997, respectively.

```
df2 <- svyby(~totexp, ~sex+period, survey_design, svymean)
df2
```

```
##     sex period      totexp          se
## 1.0   1      0 4879.518399 103.6333733
## 2.0   2      0 6016.325527 109.4863008
## 1.1   1      1 5954.493236 166.0967456
## 2.1   2      1 6997.285661 138.6561924
```

We can visualize the trend of the survey-weighted total healthcare costs for males and females from 2016 to 2021 using the `ggsurvey` function. From the plot, we can see that the female group had a higher total healthcare cost trend compared to males.

```
# Plot total expenditures over time (Load the "questionr" package to use "ggsurvey")
ggsurvey(survey_design) +
  aes(x = year, y = totexp, group = factor(sex), color = factor(sex)) +
  stat_summary(fun.y = "mean", geom = "line", size = 2)
```



6.6 Linear regression model

We can now construct our linear regression model using the ITSA framework. Because we have survey-weights that need to be applied, we will

use the `svyglm` function. We will have several interaction terms according to the ITSA formula presented above. The triple interaction term (`factor(period):factor(sex):year`) denotes the difference-in-differences estimation. In other words, it is the difference in total healthcare costs between males and females before and after the intervention was implemented. Think of it as subtracting the difference before and after for males and the difference before and after for females. Thus, we are getting two types of differences or the difference-in-differences.

- `sex` denotes our grouping variable
- `period` denotes the binary variable that denotes when the intervention was implemented (`0 = before`, `1 = after`)
- `year` denotes the time variable in the model

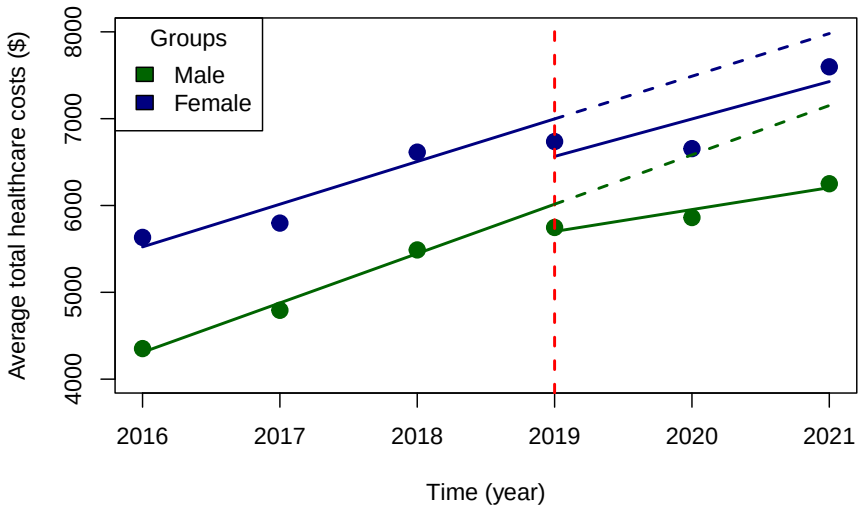
```
# ITSA: Triple interaction approach
itsa1 <- svyglm(totexp ~ factor(sex) + factor(period) + year + factor(period):factor(se
itsa1.confint <- confint(itsa1) ## generate the 95% CI

# "cbind" the coefficient output with 95% CI output
round(
  cbind(
    summary(itsa1, df.resid = degf(itsa1$survey.design))$coef,
    confint(itsa1, df = degf(itsa1$survey.design))
  ), 2)
```

```
##                                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)                       -1141706.66  227313.30   -5.02     0.00
## factor(sex)2                        156608.67  280182.38    0.56     0.58
## factor(period)1                     638235.37  399815.90    1.60     0.11
## year                                   568.46     112.71    5.04     0.00
## factor(sex)2:factor(period)1       -516623.61  486326.52   -1.06     0.29
## factor(sex)2:year                      -77.08     138.92   -0.55     0.58
## factor(sex)1:factor(period)1:year     -316.27     198.07   -1.60     0.11
## factor(sex)2:factor(period)1:year      -60.45     181.29   -0.33     0.74
##                                         2.5 %     97.5 %
## (Intercept)                       -1588358.77 -695054.56
## factor(sex)2                       -393926.88  707144.22
## factor(period)1                    -147370.28 1423841.03
## year                                   347.00     789.92
## factor(sex)2:factor(period)1      -1472215.58  438968.35
## factor(sex)2:year                     -350.04     195.88
## factor(sex)1:factor(period)1:year     -705.47      72.93
## factor(sex)2:factor(period)1:year     -416.68     295.78
```

6.7 Plot

We can plot our ITSA along with the counterfactuals. We create two new objects `itsa1` and `itsa2`, which contain the predicted values from our model. The first object `itsa1` contains the predictive value with all of our terms including the triple interaction. The second object `itsa2` contains only the counterfactual terms.

```
##############
# Plot
##############
### Plot: Part 1 - Generate the predictions
plot.itsa1 <- ggpredict(itsa1, terms = c("year", "sex", "period"), ci.level = 0.95) # Full model
plot.itsa2 <- ggpredict(itsa1, terms = c("year", "sex"), ci.level = 0.95) # Counterfactual


### Plot: Part 2 - Plot the fitted values
# Start with the scatter (average values at each year by group)
colors <- c("darkgreen", "navyblue") # Create color vector

plot(df1$year, df1$totexp, pch = 19, cex = 1.5, lwd = 1,
     col = colors[factor(df1$sex)],
     xlab = "Time (year)",
     ylab = "Average total healthcare costs ($)",
     ylim = c(4000, 8000))
legend("topleft", title = "Groups",
       legend = c("Male", "Female"),
       fill = c("darkgreen", "navyblue"))

# Add the lines for the per-post index periods
with(subset(plot.itsa1, x <= 2019 & group == 1 & facet == 0), lines(x, predicted, col = "darkgree
with(subset(plot.itsa1, x <= 2019 & group == 2 & facet == 0), lines(x, predicted, col = "navyblue
with(subset(plot.itsa1, x >= 2019 & group == 1 & facet == 1), lines(x, predicted, col = "darkgree
with(subset(plot.itsa1, x >= 2019 & group == 2 & facet == 1), lines(x, predicted, col = "navyblue

# Line for the index period
abline(v = 2019, col = "red", lty = 2, lwd = 2)

# Counterfactual lines
with(subset(plot.itsa2, x >= 2019 & group == 1), lines(x, predicted, col = "darkgreen", lty = 2,
with(subset(plot.itsa2, x >= 2019 & group == 2), lines(x, predicted, col = "navyblue", lty = 2, l
```



6.8 Parallel trends assumption

The most critical part of the ITSA results are the parallel trends assumption and the difference-in-differences estimator.

Let's start with the parallel trends assumption. We need to test to see if the trends between males and females are similar before the implementation of the intervention, which we can find in our model output. Based on our findings (see `factor(sex)2:year` coefficient), the slopes between the two groups (males and females) before the intervention were not statistically different (average difference = -$77 (95% CI: -$350, $196). This means that the slopes are not significantly different between males and females. See ITSA Figure below with annotations.

Once we are confident that the trends prior to the implementation of the intervention are not different, any changes we see after the intervention can be attributed to the intervention. This is an important condition for the difference-in-differences design.

6.9 Additional estimations

We can use the `margins` command to estimate the partial derivatives of the total healthcare costs along different parts of the timeline.

For instance, the average difference in total healthcare costs for males and females before and after the intervention were $1020 and $777, respectively. I've summarized these results into a table below.

```
### Average difference total costs in the pre- and post-intervention periods.
margins1 <- margins(itsa1, type = "response", design = survey_design, at = (list(period = 0:1)),
summary(margins1)
```

```
##  factor period       AME       SE      z      p    lower     upper
##    sex2 0.0000 1020.3870 251.0666 4.0642 0.0000 539.0557 1523.2188
##    sex2 1.0000  777.3969 260.7194 2.9817 0.0029 241.4676 1263.4687
```

Comparison	Before intervention, Average difference (95% CI)	After intervention, Average difference (95% CI)
Female versus Male	$1020 (525, 1516)	$777 (306, 1249)

Figure 6.2: Differences in total healthcare costs between males and females.

We could also estimate the slopes or the average annual change in the total healthcare costs for males and females after the intervention using the `margins` commands. For instance, average annual change in total healthcare costs for males and females are $252 and $431, respectively.

```
### Slopes after the intervention
margins3 <- margins(itsa1, type = "response", design = survey_design, at = (list(period = 1, sex
summary(margins3)
```

```
##  factor period    sex      AME       SE      z      p    lower    upper
##    year 1.0000 1.0000 252.1903 151.9225 1.6600 0.0969 -45.5724 549.9530
##    year 1.0000 2.0000 430.9319 134.2015 3.2111 0.0013 167.9019 693.9620
```

6.10 Issues with the current model

However, it is not possible to compare the slopes using the `R` version of the `margins` package. I have tried to search for a way to do this, but the `margins` package for `R` currently can't perform this comparison. You can follow the discussion on the `margins` GitHub page (link), which discusses some potential solution. However, I prefer to use `Stata` for these comparisons. Fortunately, there is a way to make some important conclusions using a different method–the linear spline model.

6.11 Workaround using a linear spline model

We could get the difference in the level change immediately after implementation of the intervention at `year == 2019` using the `lspline` command. We can also get the difference-in-differences estimate with this approach.

```
# ITSA: Linear splines approach
### Create a knot at 2019
spline1 <- svyglm(totexp ~ factor(sex) + factor(period) + lspline(year, c(2019), margi
spline1.confint <- confint(spline1)

# Output with 95% CI
round(
  cbind(
    summary(spline1, df.resid = degf(spline1$survey.design))$coef,
    confint(spline1, df = degf(spline1$survey.design))
  ), 2)
```

```
##                                                              Estimate Std. Error
## (Intercept)                                                -1141706.66  227313.30
## factor(sex)2                                                 156608.67  280182.38
## factor(period)1                                                -313.65     285.65
## lspline(year, c(2019), marginal = TRUE)1                        568.46     112.71
## lspline(year, c(2019), marginal = TRUE)2                       -316.27     198.07
## factor(sex)2:lspline(year, c(2019), marginal = TRUE)1           -77.08     138.92
## factor(sex)2:lspline(year, c(2019), marginal = TRUE)2           255.82     240.93
## factor(sex)2:factor(period)1                                   -118.53     370.05
##                                                             t value Pr(>|t|)
## (Intercept)                                                   -5.02     0.00
## factor(sex)2                                                   0.56     0.58
## factor(period)1                                               -1.10     0.27
## lspline(year, c(2019), marginal = TRUE)1                       5.04     0.00
## lspline(year, c(2019), marginal = TRUE)2                      -1.60     0.11
## factor(sex)2:lspline(year, c(2019), marginal = TRUE)1         -0.55     0.58
## factor(sex)2:lspline(year, c(2019), marginal = TRUE)2          1.06     0.29
## factor(sex)2:factor(period)1                                  -0.32     0.75
##                                                                 2.5 %     97.5 %
## (Intercept)                                                -1588358.77 -695054.56
## factor(sex)2                                                -393926.88  707144.22
## factor(period)1                                                -874.93     247.63
## lspline(year, c(2019), marginal = TRUE)1                        347.00     789.92
## lspline(year, c(2019), marginal = TRUE)2                       -705.47      72.93
## factor(sex)2:lspline(year, c(2019), marginal = TRUE)1          -350.04     195.88
## factor(sex)2:lspline(year, c(2019), marginal = TRUE)2          -217.59     729.23
## factor(sex)2:factor(period)1                                   -845.64     608.58
```

The linear spline model creates `knots` that automatically replaces some of the interactions in our previous model. For example the model element `factor(sex):lspline(year, c(2019)` replaces the triple interaction `factor(period):factor(sex):year` from our previous model.

You'll notice that some of the coefficient values are similar to our first model. For instance, in our first model, the `year` coefficient was equal to $568, and in the linear splines model, the `lspline(year, c(2019), marginal = TRUE)1` coefficient was equal to $568. This denotes the average change in annual total healthcare costs for males before the implementation of the intervention at `year == 2019`.

However, you'll also notice some differences. In the linear splines model, we have a couple of important coefficients that will help to complete our evaluation of the ITSA model.

6.12 Differences in level immediately after implementation of the intervention

One of the most important changes we are interested in is the *immediate* change after implementation of the intervention at `year == 2019`. This is captured with the `factor(sex)2:factor(period)1` coefficient, which has a value equal to -$119 (95% CI: -$846, $609). This tells me that the difference in level change at `year == 2019` was lower for females than males; however, this difference was not statistically significant. See ITSA Figure below with annotations.

6.13 Difference-in-differences estimation

Lastly, we can identify the difference-in-differences estimate from the linear spline model output in the form of the `factor(sex)2:lspline(year, c(2019), marginal = TRUE)2` coefficient, which has a value equal to $256 (95% CI: -$218, $729). This coefficient denotes that average annual change in total healthcare expenditure between males and females before and after implementation of the intervention. In other words, this is a difference in the slopes for males before and after the intervention minus the difference in the slopes for females before and after the intervention. Hence, the term `difference-in-differences`. See ITSA Figure below with annotations.

6.14 Putting it all together

With the ITSA, we will summarize our findings into a table. These include the main coefficients from the ITSA model.

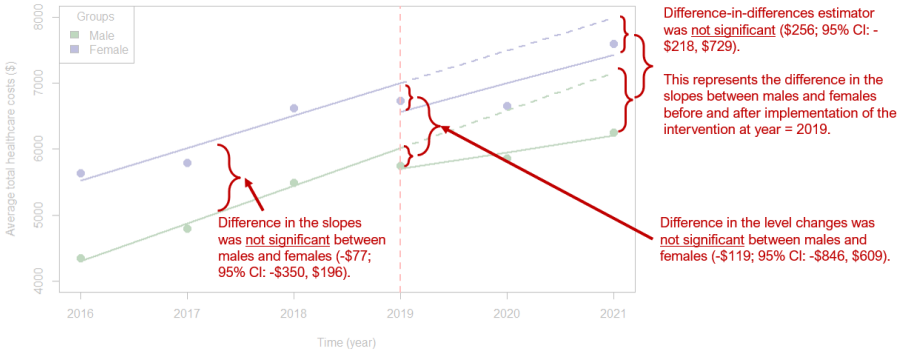


Figure 6.3: ITSA figure with annotations.

Table A. Results from the Interrupted Time Series Analysis.

Comparisons	Average difference (95% CI)	P-value
Slope(males) - Slope(females) before implementation of the interention at Year = 2019	-$77 (-$350, $196)	0.58
Difference in level change between males and females at Year = 2019	-$119 (-$846, $609)	0.75
Difference-in-differences	$256 (-$218, $729)	0.29

Figure 6.4: Summary table for main ITSA coefficients.

6.15 Conclusions

The ITSA model is a very useful design to generate causal interpretation as long as the assumptions hold. For instance, the parallel trends assumptions needs to be assessed. Without it, it will be difficult to argue that the intervention was the only factor that could have impacted the changes in the level and trends between the groups.

In this tutorial, I ran into an issue with the first ITSA model using interaction terms. It didn't provide me with the necessary coefficients to determine the main coefficients for the ITSA. For instance, I had to use a linear spline model to identify the difference-in-differences estimate. Although the ITSA model with the triple interaction was useful in generating the plot, I could have started with the linear splines model and generated the ITSA plot afterwards with the triple interaction model.

At the end of the day, I was able to generate a summary table and a very nice figure of the ITSA. However, I will continue to learn more about other contrasting packages such as `emmeans` and `marginaleffects`.

Although the `margins` package in `R` doesn't have the `pwcompare` feature from `Stata`, there are some potential workarounds that I'll investigate further. Discussion about this can be found in the `margins` GitHub issues forum (link)

6.16 Acknowledgements

As always, I am grateful to the `R` community for producing wonderful tools for novices like me. I've learned so much reading documents, forums, and issues online that it would be impossible to credit everyone. But I would like to highlight a few superstars that I think deserve special recognition.

I am grateful to the author of the `margins` package, Thomas J. Leeper. You can find his `margins` package on his `GitHub site`.

Additionally, the `marginaleffects` package by Vincent Arel-Bundock has helped me to compare some of `R`'s marginal effects commands with `Stata`. You can find his GitHub site here.

The `emmeans` package was developed by Russel V. Length, which I plan on learning more about on his GitHub site (link). Note: I found a nice overview of the `emmeans` package by Ariel Muldoon that I encourage interested users to read (link).

The `ggeffect` package was very useful in helping me develop some very nice plots, which you can learn more about on their GitHub site (link).

Lastly, I found this amazing site on ITSA by Chrissy H. Roberts here. Her site helped me to create the ITSA plot which is in this tutorial. I hope you get a

chance to read her blog on how to conduct an ITSA in R. I think she does a much better job that I did with this tutorial.

6.17 Work in progress

This is a work in progress so expect some updates in the future.

6.18 Disclaimers

Any errors or mistakes are those of the author.

This is only for educational purposes.

This was built under R version 4.3.1 "Beagle Scouts"

Chapter 7

Helpful Notes

7.1 Introduction

There are several things that I've learned while exploring and using the Agency for Health Research and Quality (AHRQ) Medical Expenditure Panel Survey (MEPS) data. Although there are (very good) documentations provided by AHRQ MEPS, I thought I would write a short article about some of the ("hard") lessons I learned.

In this article, I summarize some of these lessons in no particular order. I may expand on this list as time goes on, but for now, I will start off with a few of these.

7.1.1 1. MEPS file names

AHRQ MEPS has a GitHub site with a lot of resources. One of the most useful resource is a list of their MEPS files. This list contains the MEPS file names, which are used to identify the data for import using the `MEPS` package.

For example, we can use the `read_MEPS()` function to import the MEPS file of interest (Note: Make sure you have an internet connection). The file name for the 2021 Full-Year Consolidated File is `"h233"`.

```
### Load MEPS library
library("MEPS")

### Import the 2021 Full Year Consolidated File
hc2021 = read_MEPS(file = "h233") # Full-year consolidated file
```

You can find the list of the MEPS file names here.

Here is an example of some of the file codes:

Preview | Code | Blame | 36 lines (36 loc) · 2.23 KB | Raw

	Year	Panels	PIT	FYC	Conditions	PMED Events	Events	Jobs	PRPL	CLNK	RXLK	Multum	PSAQ	MOS	FS
2	1996	1	h01	h12	h06r	h10a	h10*f1	h07	h24	h10if1	h10if2	h68f1	-	-	-
3	1997	1, 2	h05	h20	h18	h16a	h16*f1	h19	h47f1	h16if1	h16if2	h68f2	-	-	-
4	1998	2, 3	h09	h28	h27	h26a	h26*f1	h25	h47f2	h26if1	h26if2	h68f3	-	-	-
5	1999	3, 4	h13	h38	h37	h33a	h33*	h32	h47f3	h33if1	h33if2	h68f4	-	-	-
6	2000	4, 5	h22	h50	h52	h51a	h51*	h40	h47f4	h51if1	h51if2	h68f5	-	-	-
7	2001	5, 6	h34	h60	h61	h59a	h59*	h56	h57	h59if1	h59if2	h68f6	-	-	-
8	2002	6, 7	h53	h70	h69	h67a	h67*	h63	h66	h67if1	h67if2	h68f7	-	-	-
9	2003	7, 8	h64	h79	h78	h77a	h77*	h74	h76	h77if1	h77if2	h68f8	-	-	-
10	2004	8, 9	h75	h89	h87	h85a	h85*	h83	h88	h85if1	h85if2	h68f9	-	-	-
11	2005	9, 10	h84	h97	h96	h94a	h94*	h91	h95	h94if1	h94if2	h68f10	-	-	-
12	2006	10, 11	h93	h105	h104	h102a	h102*	h100	h103	h102if1	h102if2	h68f11	-	-	-
13	2007	11, 12	h101	h113	h112	h110a	h110*	h108	h111	h110if1	h110if2	h68f12	-	-	-
14	2008	12, 13	h109	h121	h120	h118a	h118*	h116	h119	h118if1	h118if2	h68f13	-	-	-
15	2009	13, 14	h117	h129	h128	h126a	h126*	h124	h127	h126if1	h126if2	h68f14	-	-	-
16	2010	14, 15	h125	h138	h137	h135a	h135*	h133	h136	h135if1	h135if2	h68f15	-	-	-
17	2011	15, 16	h134	h147	h146	h144a	h144*	h142	h145	h144if1	h144if2	h68f16	-	-	-
18	2012	16, 17	h143	h155	h154	h152a	h152*	h150	h153	h152if1	h152if2	h68f17	-	-	-

Figure 7.1: Figure - MEPS file codes example.

7.1.2 2. Expenditure categories

MEPS contains a ton of expenditure categories such as outpatient and inpatient. This is helpful when you want to **deconstruct** the total expenditures into its different components.

Here is a list of the various expenditure categories from the 2021 Full Year Consolidated file.

7.1.2.1 2.1. Example expenditure categories

For example, the 2021 annual costs variable for office-based visits is `OBVEXP21`. Each of these expenditure categories represent different areas of care that an individual can access. I provide a summary of a select few.

- `OBVEXP21` denotes the office-based visits costs for 2021. These include encounters that are provided in the physician's office or clinic setting. It does not include encounters associated with the hospital, nursing home, or patient's home.

Appendix 3

Summary of Utilization and Expenditure Variables by Health Service Category

HEALTH SERVICE CATEGORY	UTILIZATION VARIABLE(S)	EXPENDITURE VARIABLE(S)
All Health Services	--	TOT***21
Total Office Based Visits (Physician + Non-physician + Unknown)	OBTOTV21	OBV***21
Office Based Visits to Physicians	OBDRV21	OBD***21
Total Outpatient Visits (Physician + Non-physician + Unknown)	OPTOTV21	--
Sum of Facility and SBD Expenses	--	OPT***21
Facility Expense	--	OPF***21
SBD Expense	--	OPD***21
Outpatient Visits to Physicians	OPDRV21	--
Facility Expense	--	OPV***21
SBD Expense	--	OPS***21
Total Emergency Room Visits	ERTOT21	--
Sum of Facility and SBD Expenses	--	ERT***21
Facility Expense	--	ERF***21
SBD Expense	--	ERD***21
Total Inpatient Stays	IPDIS21, IPNGTD21	--
Sum of Facility and SBD Expenses	--	IPT***21
Facility Expense	--	IPF***21
SBD Expense	--	IPD***21
Total Prescription Medicines	RXTOT21	RX***21
Total Dental Visits	DVTOT21	DVT***21
Total Home Health Care	HHTOTD21	--
Agency Sponsored	HHAGD21	HHA***21
Paid Independent Providers	HHINDD21	HHN***21
Informal	HHINFD21	--
Vision Aids	--	VIS***21
Other Medical Supplies and Equipment	--	OTH***21

KEY: To complete variable name, replace * with a particular source of payment category as identified in the following tables:**

Source of Payment Category	***
Total payments (sum of all sources)	EXP
Out of Pocket	SLF
Medicare	MCR
Medicaid	MCD
Private Insurance	PRV
Veteran's Administration/CHAMPVA	VA
TRICARE	TRI
Other Federal Sources	OFD
Other State and Local Sources	STL
Workers' Compensation	WCP
Other Unclassified Sources	OSR

Figure 7.2: Figure - MEPS expenditure categories.

- `OBTOTV21` denotes the number of office-based visits in 2021.
- `OPTEXP21` denotes the outpatient visit costs for 2021. These include encounters in hospital's outpatient departments. The total costs for an outpatient visit contains the facility and "SBD" expenses. (Note: "SBD" is short for "separate billing doctor" and refers to physicians who are billed separately for services provided at the hospital or site. They are not included in the facility-only costs.)
- `ERTEXP21` denotes the emergency department visit costs for 2021. Emergency department visit that result in an inpatient stay are "rolled" up into the inpatient costs. You should be aware that "double-counting" can occur as a consequence of this "roll-up." MEPS handles this by assigning $0 to the emergency department facility costs. Physician costs and the number of emergency department visits are not affect by this "roll-up" issue.
- `ERTOT21` denotes the number of emergency room visits in 2021.
- `IPTEXP21` denotes the inpatient visit costs for 2021. Inpatient visit costs include the facility and SBD costs.
- `IPDIS21` denotes the number of hospital discharges.
- `IPNGTD21` denotes the number of nights associated with hospital discharges.
- `RXEXP21` denotes the total medication costs for 2021. These costs include payments by insurers, health plan benefits, public (Medicare and Medicaid), VA, and out-of-pocket costs.
- `RXTOT21` denotes the number of prescription fills and refills for 2021.

7.1.3 3. Sources of payments

Each expenditure is composed of various sources of payments. Some a mostly composed of public or private payers. Some are mostly out-of-pocket payments. MEPS breaks these down into many parts, the sum of which should equal the total expenditures.

7.1.3.1 3.1. Example of sources of payments

In Appendix 3, the `***` that are listed in the expenditure categories as placeholders for the source of payment codes. These codes are listed in the Figure above. For example, the code for total payments is `EXP`, and the code for out-of-pocket payments is `SLF` or self. Thus, the total payments for emergency department

visits in 2021 is `ERTEXP21`, and the out-of-pocket payments for total prescription medicines in 2021 is `RXSLF21`.

Below is an illustration on how to match expenditure variable codes with the source of payment codes.

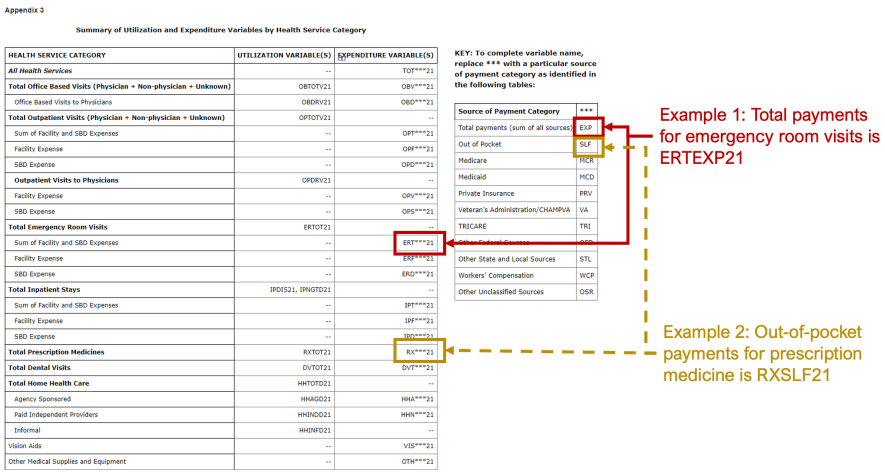


Figure 7.3: Figure - MEPS expenditure categories.

7.2 References

Most of the information can be found on the AHRQ MEPS website. I encourage the learner to read through the documentations for each data file. It's rich with useful information.

Another great resource is the AHRQ MEPS GitHub page. There are example codes on how to use the MEPS data based on R, Stata, and SAS.

7.3 Disclaimer

I plan to update this as I learn more about MEPS data, so stay tuned.

This is for educational purposes only.