R VII

Econ 5/Poli 5D Lecture 18

Announcements

- Third homework
- Final projects
- Any questions?

Today's Application: During Recessions, hours of work falls. How do individuals reallocate their time?

Today's Software: Writing your own functions in R

Today's Question: How do individuals respond to lost work hours during recessions?

Many possibilities:

- Home production: taking care of children, household chores, etc.
- Education: further education may expand job opportunities
- Leisure: watching tv, hobbies, sleeping
- Job search: applying to jobs, polishing resumes, etc.

Getting a handle on how people spend their times necessitates detailed data

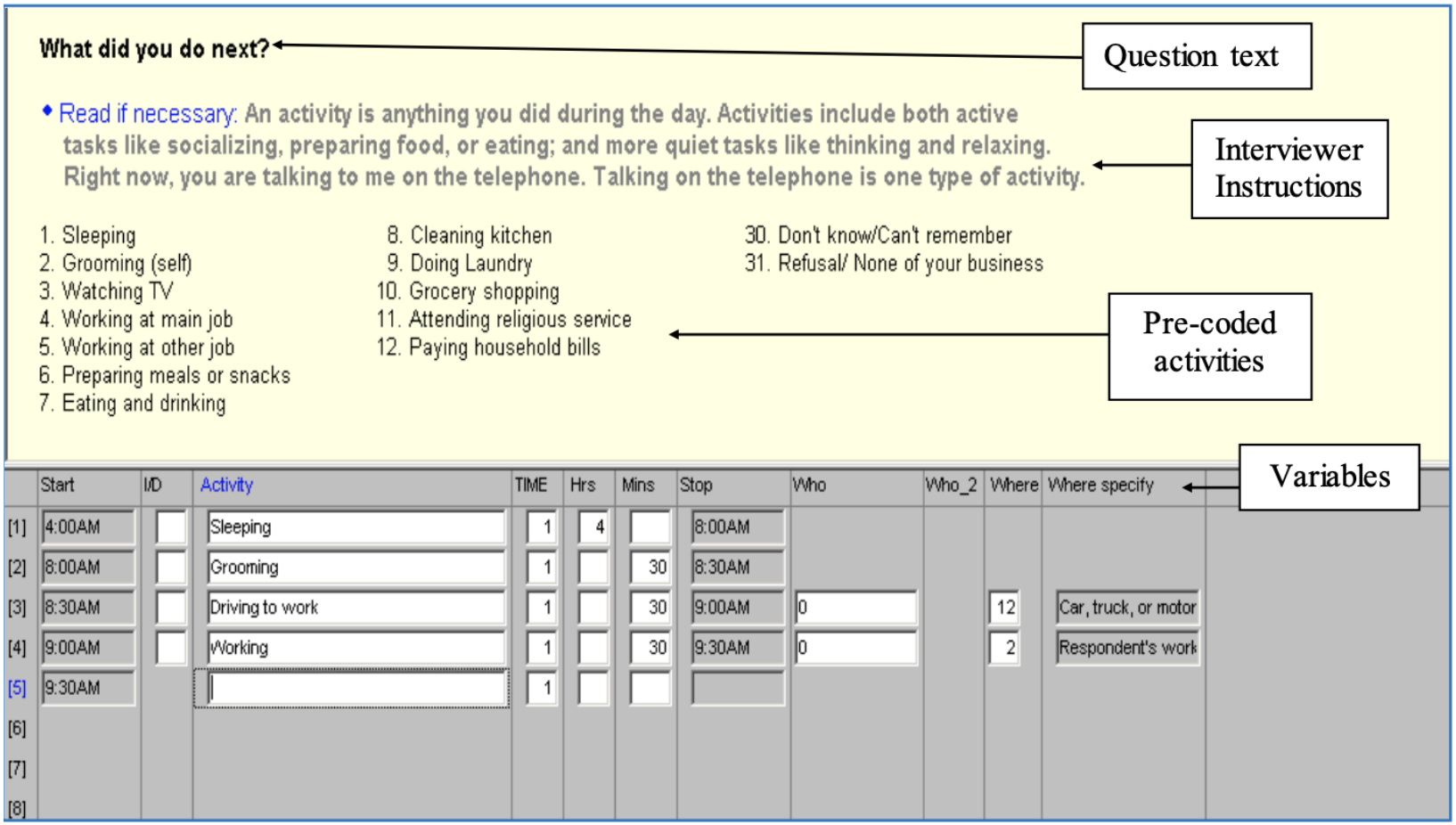
Today we will look at the American Time Use Survey (ATUS)

American Time Use Survey

The American Time Use Survey (ATUS) surveys individuals about their time use (it is administered by the Bureau of Labor Statistics, the same organization that runs the CPS)

The data is based on time-diaries, which are a detailed description of the activities of a given day

For example, if the interview was held today (Wednesday), the individual would report everything they did on Tuesday (from 4 AM on Tuesday to 4 AM on Wednesday)


What did you do next?
Question text
Read if necessary: An activity is anything you did during the day. Activities include both active tasks like socializing, preparing food, or eating; and more quiet tasks like thinking and relaxing. Right now, you are talking to me on the telephone. Talking on the telephone is one type of activity.
Interviewer Instructions
1. Sleeping
2. Grooming (self)
3. Watching TV
4. Working at main job
5. Working at other job
6. Preparing meals or snacks
7. Eating and drinking
8. Cleaning kitchen
9. Doing Laundry
10. Grocery shopping
11. Attending religious service
12. Paying household bills
30. Don't know/Can't remember
31. Refusal/ None of your business
Pre-coded activities
Variables
Start
I/D
Activity
TIME
Hrs
Mins
Stop
Who
Who_2
Where
Where specify
[1] 4:00AM Sleeping 1 4 8:00AM
[2] 8:00AM Grooming 1 30 8:30AM
[3] 8:30AM Driving to work 1 30 9:00AM 0 12 Car, truck, or motor
[4] 9:00AM Working 1 30 9:30AM 0 2 Respondent's work
[5] 9:30AM 1
[6]
[7]
[8]

Today we will look at how leisure (and the components of leisure) change during recessions

Leisure will include many categories including:

- Eating
- Sleeping
- Watching TV
- Socializing with Friends
- Reading

We will also consider how this changed by gender over time

This analysis is inspired by Aguiar, Hurst, and Karabarbounis (2014) who study in detail how individuals reallocated their time during the Great Recession

Before we get to the analysis we will learn how to write new functions

Writing our own functions

So far, in both Stata and R, we have relied on **functions** that have been supplied by the program

For example, to take the mean of a variable in R, we use the `mean()` function

In some cases, it will be helpful to write our own functions

Review: What is a function

A **function** takes in a set of **inputs** and produces a set of **outputs**

Ex: the `mean()` function takes in a variable or vector (the input), and outputs the mean of that variable or vector (the output)

Therefore, in this case, the input is a list of numbers, and the output is a single number (the average of all the numbers)

Understanding Functions

To understand how to write a function in R, let's go through a simple example

We are going to write a function that calculates the area of a circle given a radius

Reminder: $Area = \pi r^2$

Components we need to specify to write a function in R

- **name** : the name of the function
- **body** : what the function does (i.e. take radius and compute area)
- **input** : radius of the cirlce
- **output**: area of the circle

In [1]:

```
circ.area <- function(r) {
    Area <- pi*r^2
    return(Area)
}
```

In [1]:

```r
circ.area <- function(r) {

    Area <- pi*r^2

    return(Area)

}
```

First Line -- gives a name to the function `circ.area` and specifies the input `r`

- Eventually we will run this for a few different instances of `r`
- r is what you are naming the input, but you can name it something different. For example, if you replace every instance of `r` with `radius` the code will work exactly the same

Second Line -- computes the area of a circle with input given by `r`

Third Line -- specifies what the function should return (in this case the area of the circle)

Now in our environment we have a function named `circ.area` that we can use to compute the area of circles with different radii

For example, if we want know the area of a circle with radius equal to 1, we type

Now in our environment we have a function named `circ.area` that we can use to compute the area of circles with different radii

For example, if we want know the area of a circle with radius equal to 1, we type

In [2]:
```
circ.area(1)
```

3.14159265358979

Now in our environment we have a function named `circ.area` that we can use to compute the area of circles with different radii

For example, if we want know the area of a circle with radius equal to 1, we type

In [2]:
```
circ.area(1)
```

3.14159265358979

Area of a circle with radius equal to 2

Now in our environment we have a function named `circ.area` that we can use to compute the area of circles with different radii

For example, if we want know the area of a circle with radius equal to 1, we type

In [2]:
```
circ.area(1)
```

3.14159265358979

Area of a circle with radius equal to 2

In [3]:
```
circ.area(2)
```

12.5663706143592

Multiple Output

So far the output has been a single number

It is just as easy for us to write a function that returns multiple outputs

To see how this works, we will write a function that returns the circumference and area of a circle

The code below (1) computes the area and circumference of a circle with a given radius and (2) saves these results in an output vector

For clarity, the code also includes a `names` vector which indicates what number is the area and what number is the circumference

The code below (1) computes the area and circumference of a circle with a given radius and (2) saves these results in an output vector

For clarity, the code also includes a `names` vector which indicates what number is the area and what number is the circumference

In [4]:

```
circ.area.circumference <- function(r) {

  # construct the statistics we want the function to return
  Area <- pi*r^2
  Circumference <- 2*pi*r

  # We will save two vectors in the output
  # first vector -- names of the statistics
  names <- c("Area","Circumference")

  # second vector -- the values of the statistics
  stats <-c(Area,Circumference)

  # combine the two vectors
  output <- cbind(names,stats)

  return(output)

}
```

Now, whenever we type `circ.area.circumference(r)` we will retrieve the area and circumference of the circle with the given radius

Now, whenever we type `circ.area.circumference(r)` we will retrieve the area and circumference of the circle with the given radius

In [5]:
```
circ.area.circumference(1)
```

A matrix: 2 × 2 of type chr

names	stats
Area	3.14159265358979
Circumference	6.28318530717959

In [6]:
```
circ.area.circumference(2)
```

A matrix: 2 × 2 of type chr

names	stats
Area	12.5663706143592
Circumference	12.5663706143592

Functions with Data

When you find yourself copy-and-pasting the same line of code many times, it might be a good idea to write a function that performs the required procedure (similar to when we wanted to use loops)

This can help clarify and streamline code used for data analysis

Let's load in the American Time Use Survey to get started

In [7]:

```
##Set Working Directory
setwd('/Users/Brian/Dropbox/Grad School/Sixth Year/Econ:Poli 5/Lectures/Week 10/data')
##Load Packages
library("dplyr")
library("ggplot2")
##Load Data
df <- read.csv("ATUS.csv")
```

```
Warning message:
“package ‘dplyr’ was built under R version 3.6.2”

Attaching package: ‘dplyr’


The following objects are masked from ‘package:stats’:

    filter, lag


The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union


Warning message:
“package ‘ggplot2’ was built under R version 3.6.2”
```

Writing your own "mean" function

To begin, let's write a function for which we can check our work

To do so, we will write a function that takes the mean of a variable

We will name this function `my.mean()` and hopefully it will produce the exact same result as using the R function `mean()`

Elements of the mean function

1. Inputs -- Our inputs will be (1) the variable of a dataframe

2. Body -- the body of the function (i.e. what the function does) will take the average of the variable

3. Outputs -- the output will be a single number, the mean of the variable specified in inputs

In [8]:

```
my.mean <- function(variable){

  output <- sum(variable)/length(variable)
  return(output)

}
```

In [8]:

```
my.mean <- function(variable){

  output <- sum(variable)/length(variable)
  return(output)

}
```

The average is equal to

$$Average = \frac{variable_1 + variable_2 + \ldots + variable_n}{n}$$

Where n is the total number of observations, and $variable_1$ is the value of the first observation, $variable_n$ is the value of the n^{th} observation

`sum(variable)` is the numerator

`length(variable)` just counts the number of observations and therefore is equal to the denominator

Now we can check our user-written function `my.mean()` against the R-provided function `mean()`

To test this, let's take the average value of `leisure`

Now we can check our user-written function `my.mean()` against the R-provided function `mean()`

To test this, let's take the average value of `leisure`

In [9]:
```
mean(df$leisure)
```

111.219054644843

Now we can check our user-written function `my.mean()` against the R-provided function `mean()`

To test this, let's take the average value of `leisure`

In [9]:
```
mean(df$leisure)
```

111.219054644843

In [10]:
```
my.mean(df$leisure)
```

111.219054644843

Daily Activities Through The Great Recession

Now let's look at how work and some leisure activities changed over time in the U.S.

We will want to create the same plot over and over for different variables

One way to make this process easier will be to write a function that saves the information for the type of plot we want to produce

To begin, we will make a single plot, and then think about how we can put this into a function

Hours Devoted to Work

Let's make a plot that computes the average number of hours worked per week, by gender, across time

Once we do this, we will try to figure out how to alter the code so that we can put it into a function where the input is a variable and the output is a plot of the average of that variable over time

We can accomplish this by combinging a few things we have learned so far

Step 1: We can use `group_by` from dplyr to create our summary statistics of the average number of hours worked by year and gender

Step 1: We can use `group_by` from dplyr to create our summary statistics of the average number of hours worked by year and gender

In [11]:
```
means <- df %>% group_by(year,male) %>% summarise(mean=mean(work))
```

```
`summarise()` regrouping output by 'year' (override with `.groups` argumen
t)
```

Step 1: We can use `group_by` from dplyr to create our summary statistics of the average number of hours worked by year and gender

In [11]:
```
means <- df %>% group_by(year,male) %>% summarise(mean=mean(work))
```

```
`summarise()` regrouping output by 'year' (override with `.groups` argumen
t)
```

In [12]:
```
head(means)
```

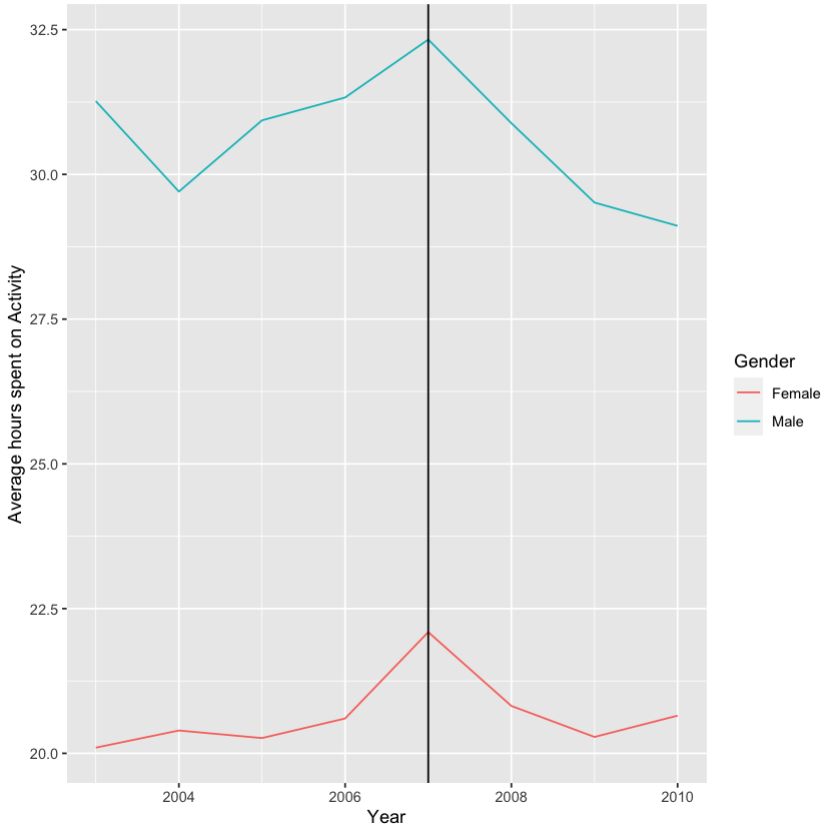
A grouped_df: 6 × 3

year	male	mean
<int>	<int>	<dbl>
2003	0	20.09829
2003	1	31.26603
2004	0	20.39468
2004	1	29.70322
2005	0	20.26397
2005	1	30.93355

Step 2: Create a plot using our `means` dataframe

Step 2: Create a plot using our `means` dataframe

In [13]:
```
ggplot(means, aes(x=year,y=mean, group=as.factor(male), color=as.factor(male))) +
  geom_line() +
  xlab("Year") +
  ylab("Average hours spent on Activity") +
  geom_vline(xintercept=2007) +
  scale_color_discrete(name = "Gender", labels = c("Female", "Male"))
```



Now let's put this code into a function so that we can input a variable and retrieve the average of that variable over time by gender

In our current code, we are doing this just for the variable `work`

So in our function, we want to replace every instance of `work` with whatever we name the input

In [14]:

```
time.plot <- function(var) {

  means <- df %>% group_by(year,male) %>% summarise(mean=mean({{var}}))

  plot <- ggplot(means, aes(x=year,y=mean, group=as.factor(male), color=as.factor(male)),env
    geom_line() +
    xlab("Year") +
    ylab(paste("Hours spent on Given Activity")) +
    geom_vline(xintercept = 2007) +
    scale_color_discrete(name = "Gender", labels = c("Female", "Male"))

  return(plot)
}
```

In [14]:

```
time.plot <- function(var) {

  means <- df %>% group_by(year,male) %>% summarise(mean=mean({{var}}))

  plot <- ggplot(means, aes(x=year,y=mean, group=as.factor(male), color=as.factor(male)),env
    geom_line() +
    xlab("Year") +
    ylab(paste("Hours spent on Given Activity")) +
    geom_vline(xintercept = 2007) +
    scale_color_discrete(name = "Gender", labels = c("Female", "Male"))

  return(plot)
}
```

This is the same as the previous code, except for two places

Instead of

```
means <- df %>% group_by(year,male) %>% summarise(mean=mean(work))
```

We now type

```
means <- df %>% group_by(year,male) %>%
summarise(mean=mean({{var}}))
```

Because we named our input `var`, this line implies that we will provide the variable we want to summarize when we run the function `time.plot()`

The `{{}}` is something that is specific to how `dplyr` can be used inside functions. Just using `var` inside mean seems like it should work, but it doesn't (this is one unfortunate feature of using `dplyr`)

Instead of

```
ggplot(means, aes(x=year,y=mean, group=as.factor(male),
              color=as.factor(male))) +
```

We typed

```
 ggplot(means, aes(x=year,y=mean, group=as.factor(male),
  color=as.factor(male)),environment = environment()) +
```

`environment = environment())` specifies where ggplot should look for the dataframe `means`. In this case `environment()` specifies the current environment, rather than the global environment, which is usually where ggplot looks for data.

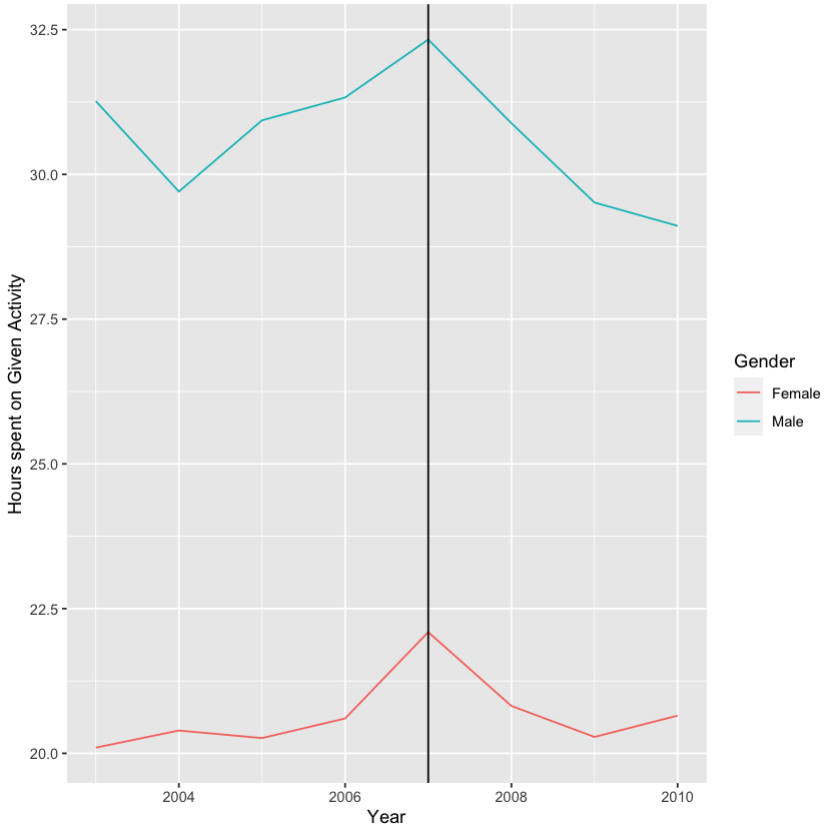
In simple language, it will just tell R that `means` was created inside the function

Now let's use our function to create a number of different plots

Now let's use our function to create a number of different plots

In [15]:
```
plot_work <- time.plot(work)
plot_work
```

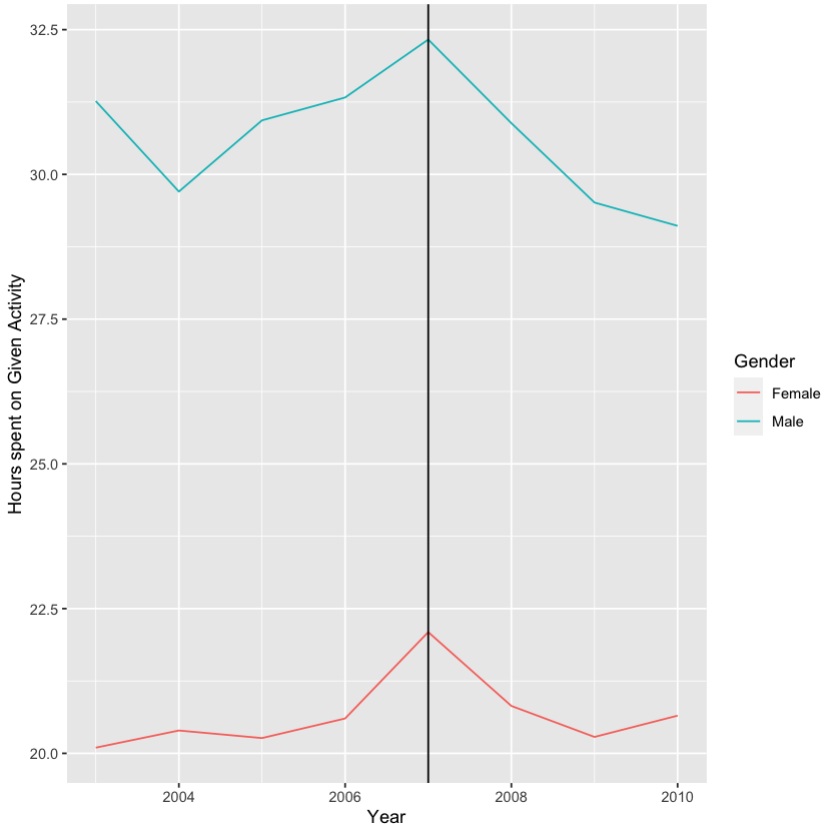
```
`summarise()` regrouping output by 'year' (override with `.groups` argumen
t)
```



Now let's use our function to create a number of different plots

In [15]:
```
plot_work <- time.plot(work)
plot_work
```

```
`summarise()` regrouping output by 'year' (override with `.groups` argumen
t)
```



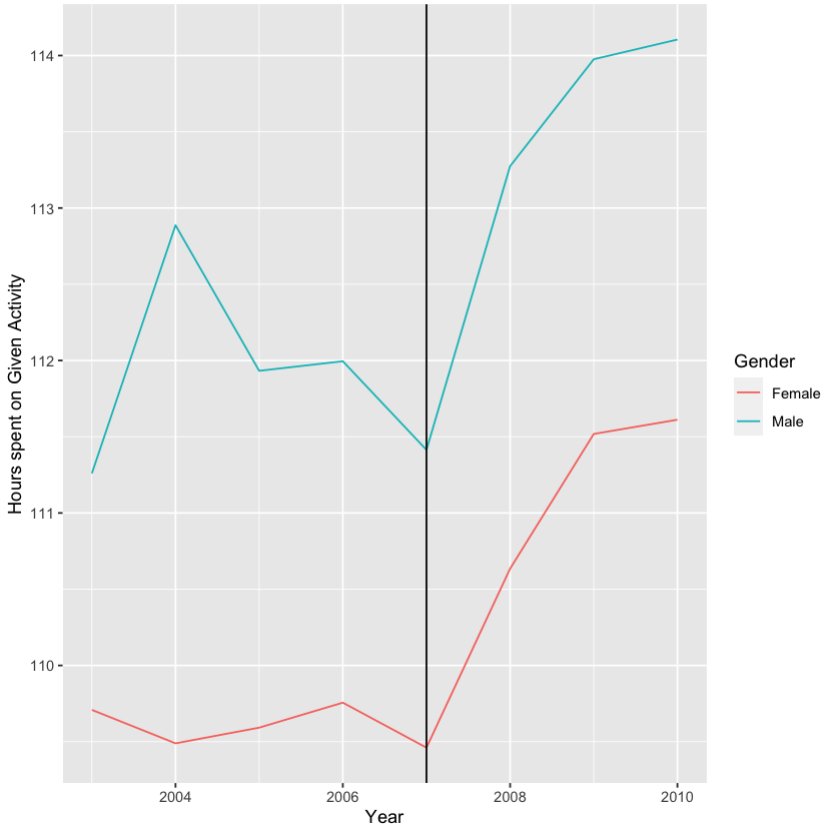
Hours of work for men and women seems to drop after the Great Recession

Now let's consider where these lost work hours may have gone. Let's start with leisure which is composed of many different activities

Now let's consider where these lost work hours may have gone. Let's start with leisure which is composed of many different activities

In [16]:
```
plot_leisure <- time.plot(leisure)
plot_leisure
```

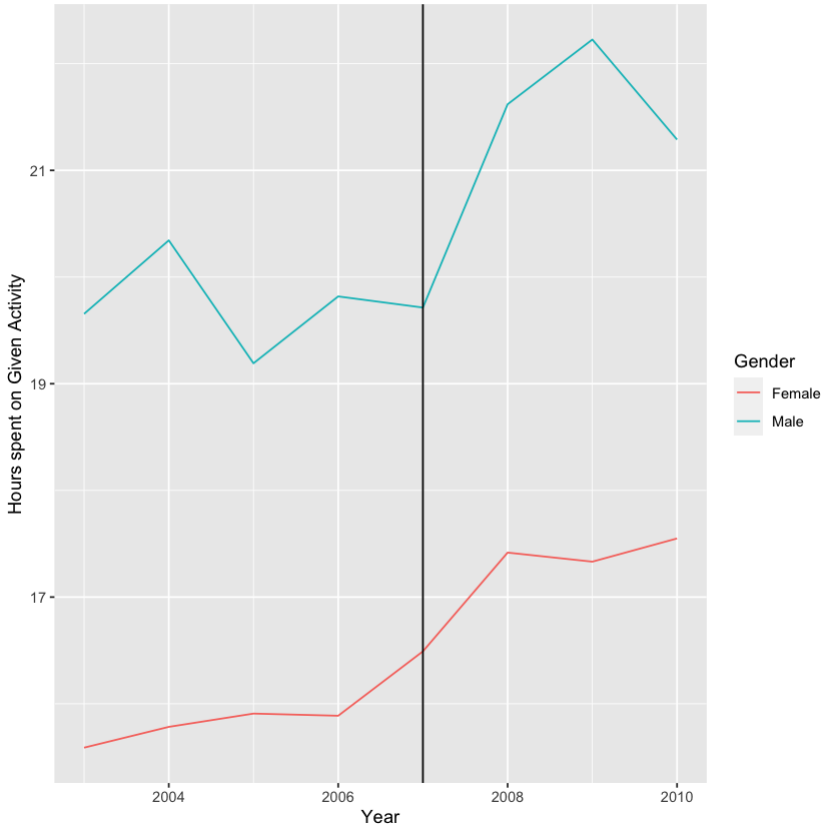
```
`summarise()` regrouping output by 'year' (override with `.groups` argumen
t)
```



In [17]:

```
# TV Plot
plot_tv <- time.plot(tv)
plot_tv
```

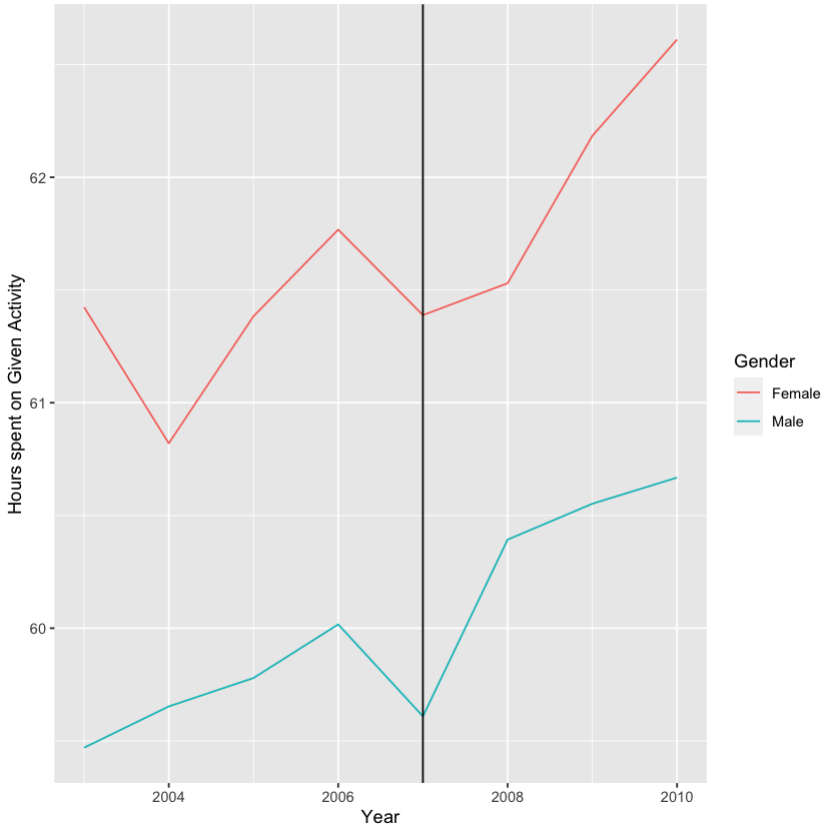
```
`summarise()` regrouping output by 'year' (override with `.groups` argumen
t)
```



In [18]:

```
plot_sleep <- time.plot(sleeping)
plot_sleep
```

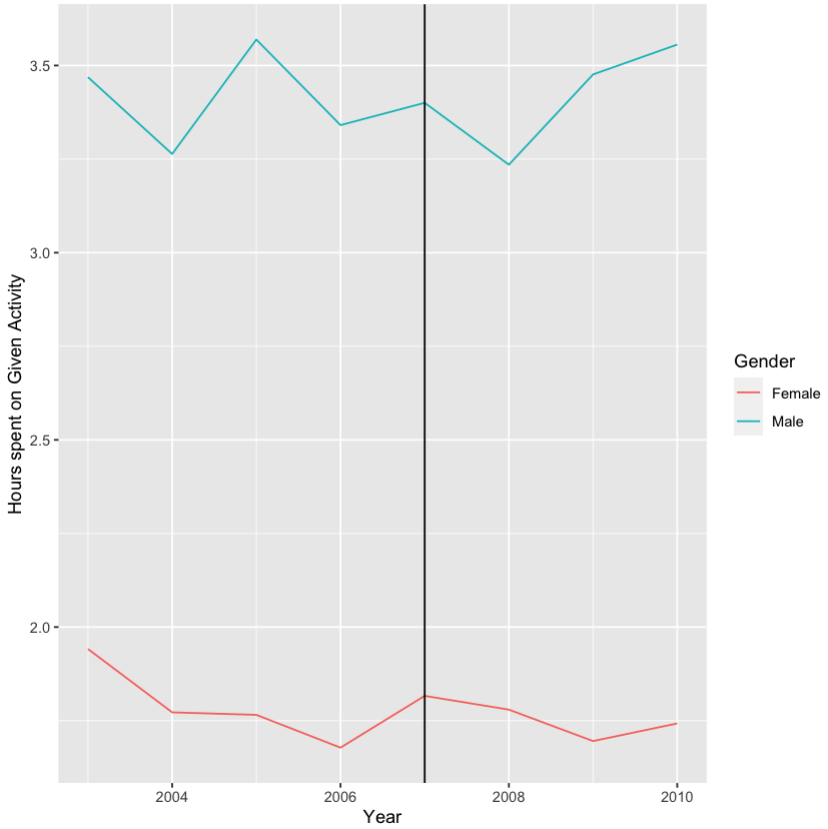
```
`summarise()` regrouping output by 'year' (override with `.groups` argumen
t)
```



In [19]:

```
plot_exercise_sports <- time.plot(exercise_sports)
plot_exercise_sports
```

```
`summarise()` regrouping output by 'year' (override with `.groups` argumen
t)
```



Today we looked at time use during the Great Recession. More in depth analysis by Aguiar et al. finds that when work hours decline, leisure activities increase (mostly sleeping and TV). To do this we learned how to write our own functions.

Conclusion - What did we do this quarter?

Congratulations! You have learned a lot this quarter:

- Introduces to three programs
- Data basics: Units of analsis/observation, data structures, survey/response/selection biases, etc.
- Practiced applying summary statistics to real life data
- Data visualization techniques
- Data cleaning and wrangling
- How to fit linear models and interpret their output
- And much more!

Best practices to keep in mind

- Keep your code, and files, organized and well commented. Write code in scripts and .do files
- Simple is often better
- Keep example codes at the ready, online resources like stackexchange are your friend
    - Whatever problem you are having someone else has probably had before

What can you do next?

- Keep developing your coding skills
    - Learn more R/Stata, learn new languages (Python, SQL, C/C++, Java, Javascript, etc.)
- Learn advanced statistics and econometrics
- Machine learning

What can you do next?

- Keep developing your coding skills
    - Learn more R/Stata, learn new languages (Python, SQL, C/C++, Java, Javascript, etc.)
- Learn advanced statistics and econometrics
- Machine learning

Other Languages

We have learned some basics in two languages (R and Stata), but there are many more that people use

- Python: Data manipulation, machine learning, platform development
- SQL: Structured query language, good for database management
- C/C++, Java, Javascript: Good to learn if you want to work with computer scientists

Further Learning at UCSD

Programming courses:

- Python: COGS 18
- Java: CSE 8 Sequence or CSE 11
- See here: https://cse.ucsd.edu/undergraduate/courses/cse-course-placement-advice

General data science courses:

- DSC 10 (It's a prereq. for everything)

Intro-Level stats courses:

- Econ 120A or Math 183/180 Series or CSE 103

Beyond that advanced topics:

- Machine learning
- GIS (Geographic information systems)
- Advanced topics in Economics (e.g. econometrics courses) and Political Science (e.g. text as data, social network analysis, etc.)

Thanks everybody, hope you had a good quarter! Good luck on the final project!