Chapter 2

Causality

Shallow men believe in luck, believe in circumstances.
Strong men believe in cause and effect.
—Ralph Waldo Emerson, *The Conduct of Life*

In this chapter, we consider causality, one of the most central concepts of quantitative social science. Much of social science research is concerned with the causal effects of various policies and other societal factors. Do small class sizes raise students' standardized test scores? Would universal health care improve the health and finances of the poor? What makes voters turn out in elections and determines their choice of candidates? To answer these causal questions, one must infer a counterfactual outcome and compare it with what actually happens (i.e., a factual outcome). We show how careful research design and data analysis can shed light on these causal questions that shape important academic and policy debates. We begin with a study of racial discrimination in the labor market. We then introduce various research designs useful for causal inference and apply them to additional studies concerning social pressure and voter turnout, as well as the impact of minimum-wage increases on employment. We also learn how to subset data in different ways and compute basic descriptive statistics in R.

2.1 Racial Discrimination in the Labor Market

Does racial discrimination exist in the labor market? Or, should racial disparities in the unemployment rate be attributed to other factors such as racial gaps in educational attainment? To answer this question, two social scientists conducted the following experiment.[1] In response to newspaper ads, the researchers sent out résumés of fictitious job candidates to potential employers. They varied only the names of job applicants, while leaving the other information in the résumés unchanged.

[1] This section is based on Marianne Bertrand and Sendhil Mullainathan (2004) "Are Emily and Greg more employable than Lakisha and Jamal? A field experiment on labor market discrimination." *American Economic Review*, vol. 94, no. 4, pp. 991–1013.

Table 2.1. Résumé Experiment Data.

Variable	*Description*
`firstname`	first name of the fictitious job applicant
`sex`	sex of applicant (`female` or `male`)
`race`	race of applicant (`black` or `white`)
`call`	whether a callback was made (`1` = yes, `0` = no)

For some candidates, stereotypically African-American-sounding names such as Lakisha Washington or Jamal Jones were used, whereas other résumés contained stereotypically white-sounding names, such as Emily Walsh or Greg Baker. The researchers then compared the callback rates between these two groups and examined whether applicants with stereotypically black names received fewer callbacks than those with stereotypically white names. The positions to which the applications were sent were either in sales, administrative support, clerical, or customer services.

Let's examine the data from this experiment in detail. We begin by loading the CSV data file, `resume.csv`, into R as a data frame object called `resume` using the function `read.csv()`. Table 2.1 presents the names and descriptions of the variables in this data set.

```
resume <- read.csv("resume.csv")
```

Instead of using `read.csv()`, you can also import the data set using the pull-down menu `Tools > Import Dataset > From Text File...` in RStudio.

This data frame object `resume` is an example of *experimental data*. Experimental data are collected from an experimental research design, in which a *treatment variable*, or a causal variable of interest, is manipulated in order to examine its causal effects on an *outcome variable*. In this application, the treatment refers to the race of a fictitious applicant, implied by the name given on the résumé. The outcome variable is whether the applicant receives a callback. We are interested in examining whether or not the résumés with different names yield varying callback rates.

Experimental research examines how a treatment causally affects an outcome by assigning varying values of the treatment variable to different observations, and measuring their corresponding values of the outcome variable.

```
dim(resume)
## [1] 4870    4
```

Using the `dim()` function, we can see that `resume` consists of 4870 observations and 4 variables. Each observation represents a fictitious job applicant. The outcome variable is whether the fictitious applicant received a callback from a prospective employer. The treatment variable is the race and gender of each applicant, though

more precisely the researchers were manipulating how potential employers perceive the gender and race of applicants, rather than directly manipulating those attributes.

Once imported, the data set is displayed in a spreadsheet-like format in an RStudio window. Alternatively, we can look at the first several observations of the data set using the `head()` function.

```
head(resume)
##   firstname    sex  race call
## 1   Allison female white    0
## 2   Kristen female white    0
## 3   Lakisha female black    0
## 4   Latonya female black    0
## 5    Carrie female white    0
## 6       Jay   male white    0
```

For example, the second observation contains a résumé for Kristen, identified as a white female who did not receive a callback. In addition, we can also create a summary of the data frame via the `summary()` function.

```
summary(resume)
##    firstname        sex          race
##  Tamika : 256   female:3746   black:2435
##  Anne   : 242   male  :1124   white:2435
##  Allison: 232
##  Latonya: 230
##  Emily  : 227
##  Latoya : 226
##  (Other):3457
##       call
##  Min.   :0.00000
##  1st Qu.:0.00000
##  Median :0.00000
##  Mean   :0.08049
##  3rd Qu.:0.00000
##  Max.   :1.00000
##
```

The summary indicates the number of résumés for each name, gender, and race as well as the overall proportion of résumés that received a callback. For example, there were 230 résumés whose applicants had the first name of "Latonya." The summary also shows that the data set contains the same number of black and white names, while there are more female than male résumés.

We can now begin to answer whether or not the résumés with African-American-sounding names are less likely to receive callbacks. To do this, we first create a

contingency table (also called a *cross tabulation*) summarizing the relationship between the race of each fictitious job applicant and whether a callback was received. A two-way contingency table contains the number of observations that fall within each category, defined by its corresponding row (`race` variable) and column (`call` variable). Recall that a variable in a data frame can be accessed using the `$` operator (see section 1.3.5). For example, the syntax `resume$race` will extract the `race` variable in the `resume` data frame.

```
race.call.tab <- table(race = resume$race,  call = resume$call)
race.call.tab
##        call
## race       0    1
##   black 2278  157
##   white 2200  235
```

The table shows, for example, that among 2435 (= 2278 + 157) résumés with stereotypically black names, only 157 received a callback. It is convenient to add totals for each row and column by applying the `addmargins()` function to the output of the `table()` function.

```
addmargins(race.call.tab)
##        call
## race       0    1  Sum
##   black 2278  157 2435
##   white 2200  235 2435
##   Sum   4478  392 4870
```

Using this table, we can compute the callback rate, or the proportion of those who received a callback, for the entire sample and then separately for black and white applicants.

```
## overall callback rate: total callbacks divided by the sample size
sum(race.call.tab[, 2]) / nrow(resume)
## [1] 0.08049281

## callback rates for each race
race.call.tab[1, 2] / sum(race.call.tab[1, ]) # black
## [1] 0.06447639

race.call.tab[2, 2] / sum(race.call.tab[2, ]) # white
## [1] 0.09650924
```

Recall that the syntax `race.call.tab[1, ]`, which does not specify the column number, extracts all the elements of the first row of this matrix. Note that in the square brackets, the number before the comma identifies the row of the matrix whereas the number after the comma identifies the column (see section 1.3.5). This can be seen by simply typing the syntax into R.

```
race.call.tab[1, ]  # the first row
##    0    1
## 2278  157

race.call.tab[, 2]  # the second column
## black white
##   157   235
```

From this analysis, we observe that the callback rate for the résumés with African-American-sounding names is 0.032, or 3.2 percentage points, lower than those with white-sounding names. While we do not know whether this is the result of intentional discrimination, the lower callback rate for black applicants suggests the existence of racial discrimination in the labor market. Specifically, our analysis shows that the same résumé with a black-sounding name is substantially less likely to receive a callback than an identical résumé with a white-sounding name.

An easier way to compute callback rates is to exploit the fact that `call` is a *binary variable*, or *dummy variable*, that takes the value 1 if a potential employer makes a callback and 0 otherwise. In general, the sample mean of a binary variable equals the sample proportion of 1s. This means that the callback rate can be conveniently calculated as the *sample mean*, or *sample average*, of this variable using the `mean()` function rather than dividing the counts of 1s by the total number of observations. For example, instead of the slightly more complex syntax we used above, the overall callback rate can be calculated as follows.

```
mean(resume$call)
## [1] 0.08049281
```

What about the callback rate for each race? To compute this using the `mean()` function, we need to first subset the data for each race and then compute the mean of the `call` variable within this subset. The next section shows how to subset data in R.

2.2 Subsetting the Data in R

In this section, we learn how to subset a data set in various ways. We first introduce logical values and operators, which enable us to specify which observations and variables of a data set should be extracted. We also learn about factor variables, which represent categorical variables in R.

2.2.1 LOGICAL VALUES AND OPERATORS

To understand subsetting, we first note that R has a special representation of the two *logical values*, `TRUE` and `FALSE`, which belong to the object class logical (see section 1.3.2).

```
class(TRUE)
## [1] "logical"
```

These logical values can be converted to a binary variable in the integer class using the function `as.integer()`, where `TRUE` is recoded as `1` and `FALSE` becomes `0`.

```
as.integer(TRUE)
## [1] 1

as.integer(FALSE)
## [1] 0
```

In many cases, R will coerce logical values into a binary variable so that performing numerical operations is straightforward. For example, in order to compute the proportion of `TRUE`s in a vector, one can simply use the `mean()` function to compute the sample mean of a logical vector. Similarly, we can use the `sum()` function to sum the elements of this vector in order to compute the total number of `TRUE`s.

```
x <- c(TRUE, FALSE, TRUE) # a vector with logical values
mean(x) # proportion of TRUEs
## [1] 0.6666667

sum(x) # number of TRUEs
## [1] 2
```

The logical values are often produced with the *logical operators* `&` and `|` corresponding to *logical conjunction* ("*AND*") and *logical disjunction* ("*OR*"), respectively. The value of "AND" (`&`) is `TRUE` only when both of the objects have a value of `TRUE`.

```
FALSE & TRUE
## [1] FALSE

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

Table 2.2. Logical Conjunction "AND" and Disjunction "OR".

Statement a	*Statement b*	*a AND b*	*a OR b*
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
FALSE	TRUE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE

The table shows the value of *a* AND *b* and that of *a* OR *b* when statements *a* and *b* are either TRUE or FALSE.

"OR" (|) is used in a similar way. However, unlike "AND", "OR" is true when at least one of the objects has the value TRUE.

```
TRUE | FALSE
## [1] TRUE

FALSE | FALSE
## [1] FALSE
```

We summarize these relationships in table 2.2. For example, if one statement is FALSE and the other is TRUE, then the logical conjunction of the two statements is FALSE but their logical disjunction is TRUE (the second and third rows of the table).

With the same principle in mind, we can also chain multiple comparisons together where all elements must be TRUE in order for the syntax to return TRUE.

```
TRUE & FALSE & TRUE
## [1] FALSE
```

Furthermore, "AND" and "OR" can be used simultaneously, but parentheses should be used to avoid confusion.

```
(TRUE | FALSE) & FALSE # the parentheses evaluate to TRUE
## [1] FALSE

TRUE | (FALSE & FALSE) # the parentheses evaluate to FALSE
## [1] TRUE
```

We can perform the logical operations "AND" and "OR" on the entire vector all at once. In the following syntax example, each element of the TF1 logical vector is compared against the corresponding element of the logical TF2 vector.


```
TF1 <- c(TRUE, FALSE, FALSE)
TF2 <- c(TRUE, FALSE, TRUE)
TF1 | TF2
## [1]  TRUE FALSE  TRUE

TF1 & TF2
## [1]  TRUE FALSE FALSE
```

2.2.2 RELATIONAL OPERATORS

Relational operators evaluate the relationships between two values. They include "greater than" (>), "greater than or equal to" (>=), "less than" (<), "less than or equal to" (<=), "equal to" (==, which is different from =), and "not equal to" (!=). These operators return logical values.

```
4 > 3
## [1] TRUE

"Hello" == "hello"  # R is case sensitive
## [1] FALSE

"Hello" != "hello"
## [1] TRUE
```

Like the logical operators, the relational operators may be applied to vectors all at once. When applied to a vector, the operators evaluate each element of the vector.

```
x <- c(3, 2, 1, -2, -1)
x >= 2
## [1]  TRUE  TRUE FALSE FALSE FALSE

x != 1
## [1]  TRUE  TRUE FALSE  TRUE  TRUE
```

Since the relational operators produce logical values, we can combine their outputs with "AND" (&) and "OR" (|). When there are multiple instances of evaluation, it is good practice to put each evaluation within parentheses for ease of interpretation.

```
## logical conjunction of two vectors with logical values
(x > 0) & (x <= 2)
## [1] FALSE  TRUE  TRUE FALSE FALSE
```

```
## logical disjunction of two vectors with logical values
(x > 2) | (x <= -1)
## [1]  TRUE FALSE FALSE  TRUE  TRUE
```

As we saw earlier, the logical values, TRUE and FALSE, can be coerced into integers (1 and 0 representing TRUE and FALSE, respectively). We can therefore compute the number and proportion of TRUE elements in a vector very easily.

```
x.int <- (x > 0) & (x <= 2) # logical vector
x.int
## [1] FALSE  TRUE  TRUE FALSE FALSE

mean(x.int) # proportion of TRUEs
## [1] 0.4

sum(x.int)  # number of TRUEs
## [1] 2
```

2.2.3 SUBSETTING

In sections 1.3.3 and 1.3.5, we learned how to subset vectors and data frames using indexing. Here, we show how to subset them using logical values, introduced above. At the end of section 2.1, we saw how to calculate the callback rate for the entire sample by applying the `mean()` function to the binary `call` variable. To compute the callback rate among the résumés with black-sounding names, we use the following syntax.

```
## callback rate for black-sounding names
mean(resume$call[resume$race == "black"])
## [1] 0.06447639
```

This command syntax subsets the `call` variable in the `resume` data frame for the observations whose values for the `race` variable are equal to `black`. That is, we can utilize square brackets `[ ]` to index the values in a vector by placing the logical value of each element into a vector of the same length within the square brackets. The elements whose indexing value is TRUE are extracted. The syntax then calculates the sample mean of this subsetted vector using the `mean()` function, which is equal to the proportion of subsetted observations whose values for the `call` variable are equal to 1. It is instructive to print out the logical vector used inside the square brackets for

subsetting. We observe that if the value of the `race` variable equals `black` (`white`) for an observation then its corresponding element of the resulting logical vector is TRUE (FALSE).

```
## race of first 5 observations
resume$race[1:5]
## [1] white white black black white
## Levels: black white

## comparison of first 5 observations
(resume$race == "black")[1:5]
## [1] FALSE FALSE  TRUE  TRUE FALSE
```

Note that `Levels` in the above output represent the values of a *factor* or categorical variable, which will later be explained in detail (see section 2.2.5). The calculation of callback rate for black-sounding names can also be done in two steps. We first subset a data frame object so that it contains only the résumés with black-sounding names and then compute the callback rate.

```
dim(resume) # dimension of original data frame
## [1] 4870    4

## subset blacks only
resumeB <- resume[resume$race == "black", ]
dim(resumeB) # this data.frame has fewer rows than the original data.frame
## [1] 2435    4

mean(resumeB$call) # callback rate for blacks
## [1] 0.06447639
```

Here, the data frame `resumeB` contains only the information about the résumés with black-sounding names. Notice that we used square brackets `[,]` to index the rows of this original data frame. Unlike in the case of indexing vectors, we use a comma to separate row and column indexes. This comma is important and forgetting to include it will lead to an error.

Instead of indexing through the square brackets, we can alternatively use the `subset()` function to construct a data frame that contains just some of the original observations and just some of the original variables. The function's two primary arguments, other than the original data frame object, are the `subset` and `select` arguments. The `subset` argument takes a logical vector that indicates whether each individual row should be kept for the new data frame. The `select` argument takes a character vector that specifies the names of variables to be retained. For example,

the following syntax will extract the `call` and `firstname` variables for the résumés which contain female black-sounding names.

```
## keep "call" and "firstname" variables
## also keep observations with female black-sounding names
resumeBf <- subset(resume, select = c("call", "firstname"),
                   subset = (race == "black" & sex == "female"))
head(resumeBf)
##    call firstname
## 3     0   Lakisha
## 4     0   Latonya
## 8     0     Kenya
## 9     0   Latonya
## 11    0     Aisha
## 13    0     Aisha
```

When using the `subset()` function, we can eliminate the `subset` argument label. For example, `subset(resume, subset = (race == "black" & sex == "female"))` shortens to `subset(resume, race == "black" & sex == "female")`. Note that one could specify the data frame name to which the `race` and `sex` variables belong, i.e., `subset(resume, (resume$race == "black" & resume$sex == "female"))`, but this is unnecessary. By default, the variable names in this argument are assumed to come from the data frame specified in the first argument (`resume` in this case). So we can use simpler syntax: `subset(resume, (race == "black" & sex == "female"))`. It is important to pay close attention to parentheses so that each logical statement is contained within a pair of parentheses.

An identical subsetting result can be obtained using `[,]` rather than the `subset()` function, where the first element of the square brackets specifies the rows to be retained (using a logical vector) and the second element specifies the columns to be kept (using a character or integer vector).

```
## alternative syntax with the same results
resumeBf <- resume[resume$race == "black" & resume$sex == "female",
                   c("call", "firstname")]
```

We can now separately compute the racial gap in callback rate among female and male job applicants. Notice that we do not include a `select` argument to specify which variables to keep. Consequently, all variables will be retained.

```
## black male
resumeBm <- subset(resume, subset = (race == "black") & (sex == "male"))
```



```
## white female
resumeWf <- subset(resume, subset = (race == "white") & (sex == "female"))
## white male
resumeWm <- subset(resume, subset = (race == "white") & (sex == "male"))
## racial gaps
mean(resumeWf$call) - mean(resumeBf$call) # among females
## [1] 0.03264689

mean(resumeWm$call) - mean(resumeBm$call) # among males
## [1] 0.03040786
```

It appears that the racial gap exists but does not vary across gender groups. For both female and male job applicants, the callback rate is higher for whites than blacks by roughly 3 percentage points.

2.2.4 SIMPLE CONDITIONAL STATEMENTS

In many situations, we would like to perform different actions depending on whether a statement is true or false. These "actions" can be as complex or as simple as you need them to be. For example, we may wish to create a new variable based on the values of other variables in a data set. In chapter 4, we will learn more about *conditional statements*, but here we cover simple conditional statements that involve the `ifelse()` function.

The function `ifelse(X, Y, Z)` contains three elements. For each element in `X` that is `TRUE`, the corresponding element in `Y` is returned. In contrast, for each element in `X` that is `FALSE`, the corresponding element in `Z` is returned. For example, suppose that we want to create a new binary variable called `BlackFemale` in the `resume` data frame that equals `1` if the job applicant's name sounds black and female, and `0` otherwise. The following syntax achieves this goal.

```
resume$BlackFemale <- ifelse(resume$race == "black" &
                             resume$sex == "female", 1, 0)
```

We then use a three-way *contingency table* obtained by the `table()` function to confirm the result. As expected, the `BlackFemale` variable equals `1` only when a résumé belongs to a female African-American.

```
table(race = resume$race, sex = resume$sex,
      BlackFemale = resume$BlackFemale)
## , , BlackFemale = 0
##
##         sex
## race    female male
##   black      0  549
##   white   1860  575
```

```
##
## , , BlackFemale = 1
##
##         sex
## race    female male
##   black   1886    0
##   white      0    0
```

In the above output, the `, , BlackFemale = 0` and `, , BlackFemale = 1` headers indicate that the first two dimensions of the three-dimensional table are shown with the third variable, `BlackFemale`, equal to `0` and `1` for the first and second tables, respectively.

2.2.5 FACTOR VARIABLES

Next we show how to create a *factor variable* (or *factorial variable*) in R. A factor variable is another name for a *categorical variable* that takes a finite number of distinct values or levels. Here, we wish to create a factor variable that takes one of the four values, i.e., `BlackFemale`, `BlackMale`, `WhiteFemale`, and `WhiteMale`. To do this, we first create a new variable, `type`, which is filled with missing values `NA`. We then specify each type using the characteristics of the applicants.

```
resume$type <- NA
resume$type[resume$race == "black" & resume$sex == "female"] <- "BlackFemale"
resume$type[resume$race == "black" & resume$sex == "male"] <- "BlackMale"
resume$type[resume$race == "white" & resume$sex == "female"] <- "WhiteFemale"
resume$type[resume$race == "white" & resume$sex == "male"] <- "WhiteMale"
```

It turns out that this new variable is a character vector, and so we use the `as.factor()` function to turn this vector into a factor variable. While a factor variable looks like a character variable, the former actually has numeric values called *levels*, each of which has a character label. By default, the levels are sorted into alphabetical order based on their character labels. The levels of a factor variable can be obtained using the `levels()` function. Moreover, the `table()` function can be applied to obtain the number of observations that fall into each level.

```
## check object class
class(resume$type)
## [1] "character"

## coerce new character variable into a factor variable
resume$type <- as.factor(resume$type)
## list all levels of a factor variable
levels(resume$type)
## [1] "BlackFemale" "BlackMale" "WhiteFemale" "WhiteMale"
```



```
## obtain the number of observations for each level
table(resume$type)
##
## BlackFemale   BlackMale WhiteFemale   WhiteMale
##        1886         549        1860         575
```

The main advantage of factor objects is that R has a number of useful functionalities for them. One such example is the `tapply()` function, which applies a function repeatedly within each level of the factor variable. Suppose, for example, we want to calculate the callback rate for each of the four categories we just created. If we use the `tapply()` function this can be done in one line, rather than computing them one by one. Specifically, we use the function as in `tapply(X, INDEX, FUN)`, which applies the function indicated by argument `FUN` to the object `X` for each of the groups defined by unique values of the vector `INDEX`. Here, we apply the `mean()` function to the `call` variable separately for each category of the `type` variable using the `resume` data frame.

```
tapply(resume$call, resume$type, mean)
## BlackFemale   BlackMale WhiteFemale   WhiteMale
##  0.06627784  0.05828780  0.09892473  0.08869565
```

Recall that the order of arguments in a function matters unless the name of the argument is explicitly specified. The result indicates that black males have the lowest callback rate followed by black females, white males, and white females. We can even go one step further and compute the callback rate for each first name. Using the `sort()` function, we can sort the result into increasing order for ease of presentation.

```
## turn first name into a factor variable
resume$firstname <- as.factor(resume$firstname)
## compute callback rate for each first name
callback.name <- tapply(resume$call, resume$firstname, mean)
## sort the result into increasing order
sort(callback.name)
##      Aisha    Rasheed     Keisha   Tremayne     Kareem
## 0.02222222 0.02985075 0.03825137 0.04347826 0.04687500
##    Darnell     Tyrone      Hakim     Tamika    Lakisha
## 0.04761905 0.05333333 0.05454545 0.05468750 0.05500000
##    Tanisha       Todd      Jamal       Neil      Brett
## 0.05797101 0.05882353 0.06557377 0.06578947 0.06779661
##   Geoffrey    Brendan       Greg      Emily       Anne
## 0.06779661 0.07692308 0.07843137 0.07929515 0.08264463
```

```
##       Jill     Latoya      Kenya    Matthew    Latonya
## 0.08374384 0.08407080 0.08673469 0.08955224 0.09130435
##      Leroy    Allison      Ebony   Jermaine     Laurie
## 0.09375000 0.09482759 0.09615385 0.09615385 0.09743590
##      Sarah   Meredith     Carrie    Kristen        Jay
## 0.09844560 0.10160428 0.13095238 0.13145540 0.13432836
##       Brad
## 0.15873016
```

As expected from the above aggregate result, we find that many typical names for black males and females have low callback rates.

2.3 Causal Effects and the Counterfactual

In the résumé experiment, we are trying to quantify the *causal effects* of applicants' names on their likelihood of receiving a callback from a potential employer. What do we exactly mean by causal effects? How should we think about causality in general? In this section, we discuss a commonly used framework for *causal inference* in quantitative social science research.

The key to understanding causality is to think about the *counterfactual*. Causal inference is a comparison between the factual (i.e., what actually happened) and the counterfactual (i.e., what would have happened if a key condition were different). The very first observation of the résumé experiment data shows that a potential employer received a résumé with a stereotypically white female first name `Allison` but decided not to call back (the value of the `call` variable is `0` for this observation).

```
resume[1, ]
##   firstname    sex  race call BlackFemale        type
## 1   Allison female white    0           0 WhiteFemale
```

The key causal question here is whether the same employer would have called back if the applicant's name were instead a stereotypically African-American name such as `Lakisha`. Unfortunately, we would never observe this counterfactual outcome, because the researchers who conducted this experiment did not send out the same résumé to the same employer using `Lakisha` as the first name (perhaps out of fear that sending two identical résumés with different names would raise suspicion among potential employers).

Consider another example where researchers are interested in figuring out whether raising the minimum wage increases the unemployment rate. Some argue that increasing the minimum wage may not be helpful for the poor, because employers would hire fewer workers if they have to pay higher wages (or hire higher-skilled instead of low-skilled workers). Suppose that one state in a country decided to raise the minimum wage and in this state the unemployment rate increased afterwards. This does not necessarily imply that a higher minimum wage led to the increase in the unemployment rate. In order to know the causal effect of increasing the minimum wage, we would need to observe the unemployment rate that would have resulted if this state had not raised the minimum wage. Clearly, we would never be able to directly survey this counterfactual unemployment rate. Another example concerns the question of whether a job training program increases one's prospect of employment. Even if someone who actually had received job training secured a job afterwards, it does not necessarily follow that it was the job training program which led to the employment. The person may have become employed even in the absence of such a training program.

Table 2.3. Potential Outcome Framework of Causal Inference.

Résumé i	*Black-sounding name* T_i	*Callback* $Y_i(1)$	*Callback* $Y_i(0)$	*Age*	*Education*
1	1	1	?	20	college
2	0	?	0	55	high school
3	0	?	1	40	graduate school
⋮	⋮	⋮	⋮	⋮	⋮
n	1	0	?	62	college

Note: The table illustrates the potential outcome framework of causal inference using the example of the résumé experiment. For each résumé of fictitious job applicant i, either the black-sounding, $T_i = 1$, or white-sounding, $T_i = 0$, name is used. The résumé contains other characteristics such as age and education, which are neither subject to nor affected by the manipulation. For a résumé with a black-sounding name, we can observe whether or not it receives a callback from the potential employer who received it, $Y_i(1)$, but will not be able to know the callback outcome if a white-sounding name was used, $Y_i(0)$. For every résumé, only one of the two potential outcomes is observed and the other is missing (indicated by "?").

These examples illustrate the *fundamental problem of causal inference*, which arises because we cannot observe the counterfactual outcomes. We refer to a key causal variable of interest as a *treatment variable*, even though the variable may have nothing to do with a medical treatment. To determine whether a *treatment* variable of interest T, causes a change in an outcome variable Y, we must consider two *potential outcomes*, i.e., the potential values of Y that would be realized in the presence and absence of the treatment, denoted by $Y(1)$ and $Y(0)$, respectively. In the résumé experiment, T may represent the race of a fictitious applicant ($T = 1$ is a black-sounding name and $T = 0$ is a white-sounding name) while Y denotes whether a potential employer who received the résumé called back. Then, $Y(1)$ and $Y(0)$ represent whether a potential employer calls back when receiving a résumé with stereotypically black and white names, respectively.

All of these variables can be defined for each observation and marked by a corresponding subscript. For example, $Y_i(1)$ represents the potential outcome under the treatment condition for the ith observation, and T_i is the treatment variable for the same observation. Table 2.3 illustrates the potential outcome framework in the context of the résumé experiment. Each row represents an observation for which only one of the two potential outcomes is observed (the missing potential outcome is indicated by "?"). The treatment status T_i determines which potential outcome is observed. Variables such as age and education are neither subject to nor affected by the manipulation of treatment.