Online Supplement

Calculating risk and prevalence ratios and differences in R: Developing intuition with a hands-on tutorial and code

Rachel R. Yorlets, Youjin Lee, Jason R. Gantenberg

Table of contents

For ease of reference, the following sections and their section numbers in this supplementary file indicate the section of the main paper to which they correspond. We also have suppressed supporting citations for material already discussed and cited in the paper.

We provide code written in the R language (R Core Team 2023) because of the software's flexibility, open-source nature, and increasing popularity among epidemiologists. R users need not pay fees, maintain an institutional affiliation, nor obtain a license to install or update the software, which works on any machine. Researchers who use R are therefore able to share (and teach) code that can be easily downloaded and run by others, improving reproducibility and facilitating real-time collaborations (Kopp 2021).

1 Applied example using clinical data to calculate a crude risk ratio and risk difference

```r
# install the pacman package if not already installed
if (! "pacman" %in% installed.packages()) {
  install.packages("pacman")
}

# load necessary packages
pacman::p_load(
  magrittr,     # pipes
  sandwich,     # sandwich estimator
  finalfit,     # missing_plot()
  logbin,       # log-binomial regression
  broom,        # exponentiate coefficients
  boot,         # bootstrapping
  ggplot2,      # plotting
  sessioninfo   # formatted session information
)

# set document output options
knitr::opts_chunk$set(cache = TRUE)

# read in NHEFS dataset
nhefs <- read.csv2("nhefs.csv", sep = ",")
```

As mentioned in the main manuscript, we are interested in estimating the associational ("crude") risk of death in 1992 relative to taking (or not taking) medication for a weak heart in 1971 among NHEFS participants who completed a baseline medical history between 1971–1975

($n = 1629$). We can first use R to explore if we have any missing data for our exposure (heart medication) or outcome (death).

```
missing_plot(
  nhefs,
  dependent = "death",
  explanatory = c("weakheart", "pregnancies"),
  plot_opts = theme(
    text = element_text(size = 20),
    axis.title.x = element_text(margin = margin(t = 15, unit ="pt"))
  )
)
```

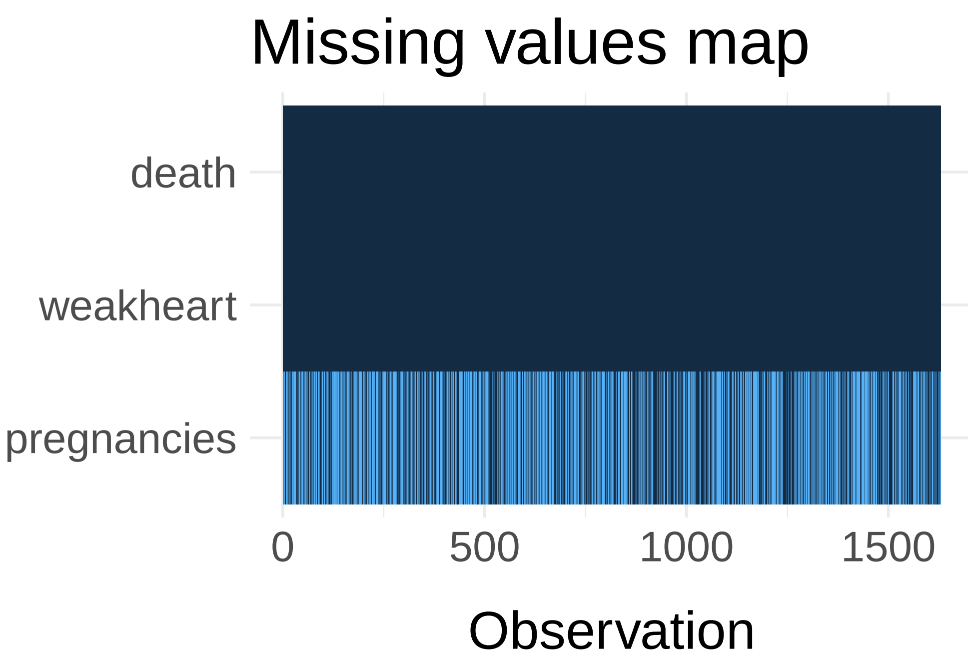


Figure 1: In this example we are not interested in the `pregnancies` variable. We include it only to depict output for a variable that contains missing values and compare it to the output for the `weakheart` and `death` variables, which do not contain missing values.

Noting that we have no missingness in our analytic variables, we can proceed.

2 Direct estimation of a risk ratio and risk difference by hand

2.1 Calculate the risk ratio

We can use R to calculate the number of participants who did or did not die by 1992, and stratify them by whether they took heart medication in 1971.

```
# 17 individuals took heart medication in 1971 died in 1992
length(which(nhefs$weakheart == 1 & nhefs$death == 1))
```

```
[1] 17
```

```
# 19 individuals took heart medication in 1971 did not die in 1992
length(which(nhefs$weakheart == 1 & nhefs$death == 0))
```

```
[1] 19
```

```
# 301 individuals did not take heart medication in 1971 and died in 1992
length(which(nhefs$weakheart == 0 & nhefs$death == 1))
```

```
[1] 301
```

```
# 1292 individuals did not take heart medication in 1971 and did not die in 1992
length(which(nhefs$weakheart == 0 & nhefs$death == 0))
```

```
[1] 1292
```

We use these numbers to populate our two-by-two table to see the risk of death as of 1992 by heart medication use in 1971 among NHEFS participants who completed a baseline medical survey:

We can now calculate the risk of death in each exposure group (here, we arbitrarily define taking heart medication as exposure, but we could also choose not taking heart medication as an exposure), and divide the risks to yield their ratio:

$$\text{Risk ratio} = \frac{\text{Risk among exposed}}{\text{Risk among unexposed}} = \frac{\left(\frac{17}{36}\right)}{\left(\frac{301}{1593}\right)} = 2.50$$

We can also use R as a calculator for this equation:

Table 1: Risk of death by medication history (n = 1,629)

		Heart medication, 1971	
		Yes	No
Death, 1992	Yes	17	301
	No	19	1292
Total		36	1593

```
rr <- (17/36) / (301/1593)
# 2.499169
rr
```

```
[1] 2.499
```

2.2 Calculate the standard error for the risk ratio

Referencing our two-by-two table, we use the number (n) of participants in each "cell" to calculate the standard error around the risk ratio. For clarity and brevity, we refer to participants who died as "cases", and those who did not as "non-cases"; here, again, exposed participants took heart medication while unexposed participants did not, indicating exposure and "case" status, where "case" can refer to any outcome.

$$
\begin{aligned}
\mathrm{SE}(\ln(\mathrm{RR})) = \\
&= \sqrt{\frac{1}{n_{\text{exp case}}} + \frac{1}{n_{\text{unexp case}}} - \frac{1}{n_{\text{exp case}} + n_{\text{exp noncase}}} - \frac{1}{n_{\text{unexp case}} + n_{\text{unexp noncase}}}} \\
&= \sqrt{\frac{1}{17} + \frac{1}{301} - \frac{1}{17+19} - \frac{1}{301+1292}} \\
&= 0.1836852 \\
&= 0.18
\end{aligned}
\tag{1}
$$

Again, we could have used R as a calculator:

```
rrse <- sqrt((1/17) + (1/301) - (1/(17+19)) - (1/(301+1292)))
# 0.1836852
rrse
```

```
[1] 0.1837
```

2.3 Calculate the 95% confidence interval for the risk ratio

We now use our standard error and risk ratio to calculate the confidence interval around the risk ratio:

$$
\begin{aligned}
\text{CI(RR)} &= e^{ln(RR) \pm z \times SE(RR)} \\
95\% \text{ CI(RR)} &= e^{ln(RR) \pm 1.96 \times SE(RR)} \\
&= e^{ln(2.499169) \pm (1.96 \times 0.1836852)} \\
&= (1.74 - 3.58)
\end{aligned} \tag{2}
$$

Note that when we use R to calculate the confidence interval, we will use the `log()` function, which R interprets as a **natural log** (not log base-ten):

```
# Calculate the upper bound
rrupp <- exp(log(rr) + 1.96 * rrse)
# 3.582215

# Calculate the lower bound
rrlow <- exp(log(rr) - 1.96 * rrse)
# 1.743571

c(RR = rr, RRse = rrse, ll95 = rrlow, ul95 = rrupp) |>
  round(digits = 3)
```

```
   RR  RRse  ll95  ul95
2.499 0.184 1.744 3.582
```

2.4 Calculate the risk difference

We can now use the same quantities to calculate the risk difference:

$$
\begin{aligned}
\text{Risk difference} &= \text{Risk among exposed} - \text{Risk among unexposed} \\
&= \left(\frac{17}{36}\right) - \left(\frac{301}{1593}\right) \\
&= 0.2832706 \\
&= 0.28
\end{aligned} \tag{3}
$$

```r
rd <- (17/36) - (301/1593)
# 0.2832706

rd
```

```
[1] 0.2833
```

2.5 Calculate the standard error for the risk difference

We can now use the above quantities of the risk among the exposed (R_1), risk among the unexposed (R_0), and the number of participants in each exposure group to calculate the standard error:

$$
\begin{aligned}
\text{SE(RD)} &= \sqrt{\frac{\text{Risk}_{\text{exposed}}(1-\text{Risk}_{\text{exposed}})}{n_{\text{exposed}}} + \frac{\text{Risk}_{\text{unexposed}}(1-\text{Risk}_{\text{unexposed}})}{n_{\text{unexposed}}}} \\
&= \sqrt{\frac{(17/36) \times [1-(17/36)]}{36} + \frac{(301/1593) \times [1-(301/1593)]}{1593}} \\
&= 0.08378074 \\
&= 0.08
\end{aligned} \tag{4}
$$

Once again, we could use R as a calculator:

```r
rdse <- sqrt((((17/36) * (1 - (17/36))) / 36) +
             ((301/1593) * (1 - (301/1593)) / 1593))
# 0.08378074
rdse
```

```
[1] 0.08378
```

2.6 Calculate the confidence interval for the risk difference

Lastly, we use the standard error to calculate a confidence interval around the risk difference:

$$
\begin{aligned}
\text{CI(RD)} &= RD \pm z \times SE \\
95\%\ \text{CI(RD)} &= RD \pm 1.96 \times SE \\
&= 0.2832706 \pm (1.96 \times 0.08378074) \\
&= (0.12, 0.45)
\end{aligned} \tag{5}
$$

```
# Calculate the upper bound
rdupp <- rd + (1.96 * rdse)
# 0.4474809

# Calculate the lower bound
rdlow <- rd - (1.96 * rdse)
# 0.1190603

c(RD = rd, RDse = rdse, ll95 = rdlow, ul95 = rdupp) |>
  round(digits = 3)
```

```
   RD  RDse  ll95  ul95
0.283 0.084 0.119 0.447
```

3 Direct estimation of a risk difference using linear probability models

When we have a binary outcome, as we do in our example (death), we can estimate the risk difference directly using linear probability models. We can fit a generalized linear model using a binomial distribution and an identity link, in which case the outcome distribution may be correctly specified, and we do not have to adjust the standard error estimates used to calculate confidence limits. (Moving forward, we will make the assumption that all observations are independent and identically distributed, justifying the assertion that the outcome follows a binomial distribution.)

In some cases, we may wish to use alternative models. The OLS model and the GLM with a Gaussian distribution are equivalent. Both of these models, plus the GLM using a Poisson distribution, require use of a "robust" sandwich estimator to estimate the standard error for the risk difference. This sandwich estimator accounts for misspecification of the probability distribution, *i.e.*, the fact that we are fitting models that assume Gaussian or Poisson distributions to an outcome that follows a binomial distribution. (See Section 5.2 for further explication in the context of the risk ratio).

3.1 Calculate a risk difference by fitting a generalized linear model, specifying an identity link and binomial probability distribution

Here, we provide code to estimate a risk difference using a **GLM with an identity link and a binomial probability distribution**. Given that in our example the outcome death is a binary variable and has a binomial distribution, the model is correctly specified; hence, no correction is needed for the standard error:

```r
glm_lin_bin_fit <- glm(death ~ weakheart,
                       data = nhefs,
                       family = binomial(link = "identity"))

# extract the coefficient for weakheart
glm_lin_bin_rd <- unname(coef(glm_lin_bin_fit)[2])

# note we use the standard variance-covariance matrix extractor
# no need for sandwich estimator under the correct probability distribution
glm_lin_bin_robse <- sqrt(diag(vcov(glm_lin_bin_fit))[2])
glm_lin_bin_robse <- unname(glm_lin_bin_robse)

c(RD = glm_lin_bin_rd,
  SE = glm_lin_bin_robse,
  `95% CL, Lower` = glm_lin_bin_rd - qnorm(0.975) * glm_lin_bin_robse,
  `95% CL, Upper` = glm_lin_bin_rd + qnorm(0.975) * glm_lin_bin_robse)
```

```
           RD            SE 95% CL, Lower 95% CL, Upper
      0.28327       0.08378       0.11906       0.44748
```

3.2 Calculate a risk difference by fitting a linear model using ordinary least squares estimation

Here, we provide code to estimate a risk difference using **OLS**. Because this model is incorrectly specified (assumes a Gaussian distribution when we are working with a binomially distributed outcome), we must estimate "robust" standard errors:

```r
lm_fit <- lm(death ~ weakheart, data = nhefs)

# extract the coefficient for weakheart
lm_fit_rd <- unname(coef(lm_fit)[2])

# extract the variance for the weakheart coeffient and
# take the square root to get the standard error
lm_fit_robse <- sqrt(diag(vcovHC(lm_fit, type = "HC0"))[2])
lm_fit_robse <- unname(lm_fit_robse)

c(RD = lm_fit_rd,
  RobustSE = lm_fit_robse,
  `95% CL, Lower` = lm_fit_rd - qnorm(0.975) * lm_fit_robse,
  `95% CL, Upper` = lm_fit_rd + qnorm(0.975) * lm_fit_robse)
```

```
     RD      RobustSE 95% CL, Lower 95% CL, Upper
0.28327       0.08378       0.11906       0.44748
```

3.3 Calculate a risk difference by fitting a generalized linear model, specifying an identity link and a Gaussian probability distribution

Here, we provide code to estimate a risk difference using a **GLM with an identity link and a Gaussian probability distribution**. Because this model is incorrectly specified (assumes a Gaussian distribution when we are working with a binomially distributed outcome), we must estimate "robust" standard errors:

```
glm_lin_gauss_fit <- glm(death ~ weakheart,
                     data = nhefs,
                     family = gaussian(link = "identity"))

glm_lin_gauss_rd <- unname(coef(glm_lin_gauss_fit)[2])
glm_lin_gauss_robse <- sqrt(diag(vcovHC(glm_lin_gauss_fit, type = "HC0"))[2])
glm_lin_gauss_robse <- unname(glm_lin_gauss_robse)

c(RD = glm_lin_gauss_rd,
  RobustSE = glm_lin_gauss_robse,
  `95% CL, Lower` = glm_lin_gauss_rd - qnorm(0.975) * glm_lin_gauss_robse,
  `95% CL, Upper` = glm_lin_gauss_rd + qnorm(0.975) * glm_lin_gauss_robse)
```

```
     RD      RobustSE 95% CL, Lower 95% CL, Upper
0.28327       0.08378       0.11906       0.44748
```

3.4 Calculate a risk difference by fitting a generalized linear model, specifying an identity link and a Poisson probability distribution

Here, we provide code to estimate a risk difference using a **GLM with an identity link and a Poisson probability distribution**. Because this model is incorrectly specified (assumes a Poisson distribution when we are working with a binomially distributed outcome), we must estimate "robust" standard errors:

```
glm_lin_pois_fit <- glm(death ~ weakheart,
                        data = nhefs,
                        family = poisson(link = "identity"))

glm_lin_pois_rd <- unname(coef(glm_lin_pois_fit)[2])
```

```
glm_lin_pois_robse <- sqrt(diag(vcovHC(glm_lin_pois_fit, type = "HC0"))[2])
glm_lin_pois_robse <- unname(glm_lin_pois_robse)

c(RD = glm_lin_pois_rd,
  RobustSE = glm_lin_pois_robse,
  `95% CL, Lower` = glm_lin_pois_rd - qnorm(0.975) * glm_lin_pois_robse,
  `95% CL, Upper` = glm_lin_pois_rd + qnorm(0.975) * glm_lin_pois_robse)
```

```
           RD      RobustSE 95% CL, Lower 95% CL, Upper
      0.28327       0.08378       0.11906       0.44748
```

4 Direct estimation of a risk ratio using a log-binomial regression model

4.1 Fit a log-binomial regression for the relationship between heart medication and mortality

We can fit a log-binomial regression model in R by using the `glm()` function from the `stats` package by specifying a binomial distribution and a (natural) log link, which is passed to the `family` argument of `glm()` as shown below. We extract β_1 and use `tidy()` from the `broom` package to exponentiate the coefficient, yielding a risk ratio, and include the 95% confidence interval in our model output.

```
lbin_fit <- glm(death ~ weakheart,
                data = nhefs,
                family = binomial(link = "log"))

tidy(lbin_fit, exponentiate = TRUE, conf.int = TRUE)
```

```
# A tibble: 2 x 7
  term        estimate std.error statistic   p.value conf.low conf.high
  <chr>          <dbl>     <dbl>     <dbl>     <dbl>    <dbl>     <dbl>
1 (Intercept)    0.189    0.0519    -32.1  4.42e-226    0.170     0.209
2 weakheart      2.50     0.184       4.99 6.15e-  7    1.65      3.42
```

We could also use the `logbin` package, which implements several subroutines that may avoid commonly encountered convergence issues with the log-binomial model:

```r
logbin_fit <- logbin(death ~ weakheart,
                     data = nhefs,
                     method = "glm")

summary(logbin_fit)
```

```
Call:
logbin(formula = death ~ weakheart, data = nhefs, method = "glm")

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-1.131  -0.647  -0.647  -0.647   1.825

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.6663     0.0519  -32.10  < 2e-16 ***
weakheart     0.9160     0.1837    4.99  6.1e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

    Null deviance: 1608.5  on 1628  degrees of freedom
Residual deviance: 1594.0  on 1627  degrees of freedom

  AIC: 1598
AIC_c: 1598

Number of iterations: 6
```

5 Direct estimation of a risk ratio using a modified Poisson model

5.1 Fit a Poisson regression for the relationship between heart medication and mortality

We can fit a Poisson regression model in R by using the `glm()` function from the `stats` package by specifying a Poisson distribution and a (natural) log link, which is passed to the `family` argument of `glm()` as shown below. We extract β_1 and use `tidy()` from the `broom` package to expontentiate the coefficient, yielding a risk ratio, and include the 95% confidence interval in our model output.

```r
poisson_fit <- glm(death ~ weakheart,
                   data = nhefs,
                   family = poisson(link = log))

tidy(poisson_fit, exponentiate = TRUE, conf.int = TRUE)
```

```
# A tibble: 2 x 7
  term        estimate std.error statistic   p.value conf.low conf.high
  <chr>          <dbl>     <dbl>     <dbl>     <dbl>    <dbl>     <dbl>
1 (Intercept)    0.189    0.0576    -28.9  9.31e-184    0.168     0.211
2 weakheart      2.50     0.249       3.67 2.39e-  4    1.47      3.94
```

5.2 Calculate "robust" standard errors using the sandwich estimator

Because we are using Poisson regression to model a binary outcome, and because binary outcomes do not follow a Poisson distribution, the Poisson model is misspecified in the sense that it will not produce valid standard errors for coefficient estimates. The sandwich estimator "corrects" these standard errors to account for the misspecification.

We begin by estimating the covariance matrix using the `vcov()` function from the `sandwich` package.

```r
# Calculate the covariance matrix using 'HC0' (refers to the sandwich estimator)
covmat <- vcovHC(poisson_fit, type = "HC0")
covmat
```

```
            (Intercept) weakheart
(Intercept)    0.002695 -0.002695
weakheart     -0.002695  0.033740
```

The diagonal of this covariance matrix contains the estimated variances for each coefficient—in this case, the intercept (β_0) and `weakheart` (β_1). Therefore, to calculate the standard errors for each coefficient, we extract the diagonal using the `diag()` function and take the square root.

The code below carries out these additional steps and uses the robust standard errors to calculate 95% confidence intervals for the coefficients on the (natural) log scale.

```r
#Calculate the standard error
se <- sqrt(diag(covmat))

# Bind together model output
#  1. exponentiated coefficients
#  2. robust standard errors
#  3. 95% confidence intervals
# Note that qnorm(0.975) approximately equals 1.96
model_output <- cbind(
  Estimate = exp(coef(poisson_fit)),
  `Robust SE` = se,
  Lower = exp(coef(poisson_fit) - qnorm(0.975) * se),
  Upper = exp(coef(poisson_fit) + qnorm(0.975) * se)
)

# Coerce model_output into a data frame
# Return second row to focus on the weakheart variable
model_output <- as.data.frame(model_output)
knitr::kable(model_output[2, ], digits = 4)
```

	Estimate	Robust SE	Lower	Upper
weakheart	2.499	0.1837	1.744	3.582

5.3 Estimate 95% confidence limits for the risk ratio via non-parametric bootstrapping.

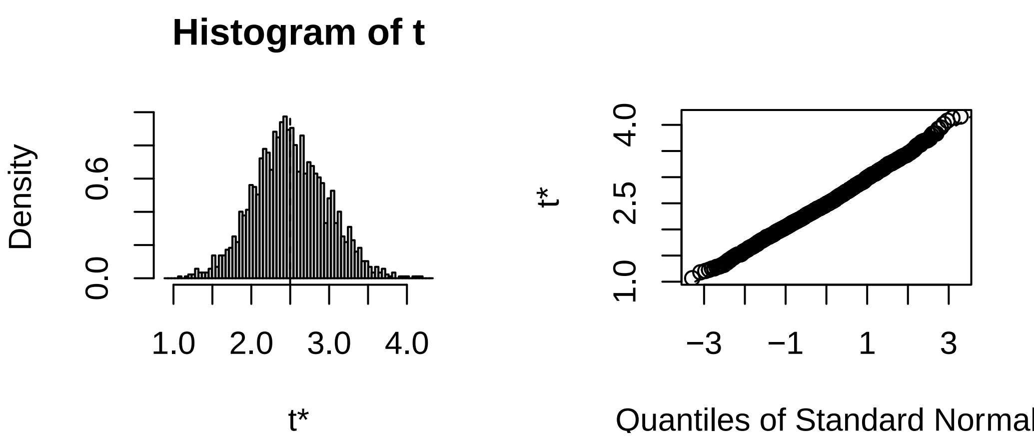
```r
# First, write a function that
#  1. takes the data
#  2. indexes the data to create a bootstrap replicate
#  3. fits a Poisson model to the indexed data
#  4. returns the estimated risk ratio
bootpois = function(dat, indices){
  fit <- glm(death ~ weakheart,
             family = poisson(link = log),
             data = dat[indices, ])
  return(exp(coef(fit))[2])
}

# Set seed
```

```r
set.seed(2999)

# Use the boot() function combined with the bootpois() function we wrote
boot_estimate <- boot(
  data = nhefs,
  statistic = bootpois,
  R = 1999,
  parallel = "multicore",
  ncpus = 6
)

plot(boot_estimate)
```




```r
# Calculate bootstrapped confidence intervals
bci <- boot.ci(boot.out = boot_estimate,
               type = c('perc', 'bca'))

print(bci)
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1999 bootstrap replicates

CALL :
boot.ci(boot.out = boot_estimate, type = c("perc", "bca"))

Intervals :
Level     Percentile            BCa
95%   ( 1.605,  3.422 )   ( 1.651,  3.470 )
Calculations and Intervals on Original Scale
```

6 Indirect estimation of a risk ratio from a logistic regression model

We can fit a logistic regression model in R by using the `glm()` function from the `stats` package by specifying a binomial distribution, as shown below. The `binomial()` function uses a logit link by default.

6.1 Fit a logistic regression for the relationship between heart medication and mortality

```
fit1 <- glm(death ~ weakheart,
            data = nhefs,
            family = binomial())
```

6.2 Predict the expected probability of death for each weakheart group.

Calculate the risk ratio's numerator as $Pr(death = 1|weakheart = 1)$:

```
prd1_w1 <- predict(fit1,
                   newdata = data.frame(weakheart = 1),
                   type = "response")
prd1_w1
```

```
     1
0.4722
```

Calculate the risk ratio's denominator as $Pr(death = 1|weakheart = 0)$:

```
prd1_w0 <- predict(fit1,
                   newdata = data.frame(weakheart = 0),
                   type = "response")
prd1_w0
```

```
    1
0.189
```

Calculate the risk ratio:

```
rr_indirect <- prd1_w1 / prd1_w0
rr_indirect
```

```
    1
2.499
```

Note that we can also calculate the risk difference using the predicted outcome probabilities for each exposure group:

```
rd_indirect <- prd1_w1 - prd1_w0
rd_indirect
```

```
     1
0.2833
```

Here are the results of each step in the procedure thus far:

```
sumlogit_ests <-c("Crude Pr(death = 1)" = mean(nhefs$death),
                  "Predicted Pr(death = 1 | weakheart = 1)" = unname(prd1_w1),
                  "Predicted Pr(death = 1 | weakheart = 0)" = unname(prd1_w0),
                  "Risk ratio" = unname(rr_indirect),
                  "Risk difference" = unname(rd_indirect),
                  "Odds ratio" = unname(exp(coef(fit1)[2])))


data.frame(Measure = names(sumlogit_ests),
           Estimate = round(unname(sumlogit_ests), 3)) |>
  kableExtra::kbl(booktabs = T, linesep = "") |>
  kableExtra::kable_styling(latex_options = "HOLD_position")
```

Measure	Estimate
Crude Pr(death = 1)	0.195
Predicted Pr(death = 1 | weakheart = 1)	0.472
Predicted Pr(death = 1 | weakheart = 0)	0.189
Risk ratio	2.499
Risk difference	0.283
Odds ratio	3.841

6.3 Estimate 95% confidence limits for the risk ratio via non-parametric boostrapping.

The basic aim of bootstrapping is to approximate the hypothetical sampling distribution upon which frequentist statistics are based (Efron and Tibshirani 1994). The general procedure involves the following steps:

1) Draw B samples from your dataset with replacement.
   - Each element of B is referred to as a *bootstrap replicate*
   - We usually set B to a large number (*e.g.*, at least 999). The specific choice must be dictated by specific features of your dataset and analysis.
   - Note that because we are sampling the original dataset with replacement, individuals from our original dataset might appear in a single bootstrap replicate 0, 1, or more than 1 time.
2) Estimate your statistic(s) of interest within each bootstrap replicate.
   - Essentially, we rerun our entire data analysis within each bootstrap replicate to get a distribution of estimates.
   - In our case, we will build bootstrapped distributions of B risk ratio estimates and B risk difference estimates.
3) Calculate standard errors, confidence intervals, and other statistical measures using the bootstrapped distributions of estimates.
   - In the simplest case, we can extract the lower and upper bounds for a 95% confidence interval by retrieving the 2.5% and 97.5% quantiles of the bootstrap distribution for our estimate.

We could write a program to carry out the procedure above, but thankfully, the `boot` package in R implements these procedures gracefully and with the added benefit of allowing us to use parallel processing to speed up computation. The `boot` package also implements several methods for obtaining bootstrapped confidence limits via simple arguments to its primary function.

The following subsections describe how to get 95% confidence intervals for our indirect estimates of the risk ratio and risk difference using non-parametric bootstrapping.

6.4 Write a function to evaluate repeatedly (*i.e.*, within each bootstrap replicate).

We begin by writing a function called `estimate_risk_measures()` that:

1) fits a logistic regression

2) extracts the predicted mortality probabilities for each `weakheart` group, and
3) returns the estimated risk ratio and risk difference.

We will use this function throughout the rest of the example.

```
estimate_risk_measures <- function(dat, indices) {
  # 1. fit logistic model
  fit <- glm(death ~ weakheart,
             data = dat[indices, ],
             family = binomial())

  # 2. get predicted probabilities for each weakheart group

  ## exposed
  pred_w1 <- predict(fit,
                     newdata = data.frame(weakheart = 1),
                     type = "response")

  ## unexposed
  pred_w0 <- predict(fit,
                     newdata = data.frame(weakheart = 0),
                     type = "response")

  # 3. calculate risk ratio and risk difference
  rr_est <- pred_w1 / pred_w0
  rd_est <- pred_w1 - pred_w0

  # 4. return the desired statistics
  output <- c(RR = rr_est, RD = rd_est)
  output
}
```

In order to play nicely with the `boot` package, our function must take two arguments: the first argument must take our base dataset as its input, while the second must take a vector of numeric indices indicating the sampled observations within a given bootstrap replicate. Note that the `boot()` function will conduct the resampling procedure itself, without our having to do it manually.

6.5 Run analysis within each bootstrap replicate.

Here we put it all together and run `estimate_risk_measures()` within each of the 1,999 bootstrap replicates we direct the `boot()` function to generate for us. Note, too, that we ask `boot::boot()` to split the process up into multiple "jobs" and run these jobs in parallel vis the `ncpus` argument. (You will typically want to set `ncpus` to one or two fewer than the total number of cores available on a personal machine, so as not to exhaust your computer's resources.)

```
# set a random number seed for reproducibility
set.seed(31415)

# Subset the data to those variables we're interested in.
# Not necessary here, but with very large datasets, could help to
# avoid memory issues, particularly when using parallel processing.
nhefs_sub <- nhefs[, c("seqn", "weakheart", "death")]

# run the bootstrap procedure
indirect_boot <- boot::boot(
  data = nhefs_sub,
  statistic = estimate_risk_measures,
  R = 1999,
  parallel = "multicore",
  ncpus = 4
)

indirect_boot
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot::boot(data = nhefs_sub, statistic = estimate_risk_measures,
    R = 1999, parallel = "multicore", ncpus = 4)


Bootstrap Statistics :
    original    bias    std. error
t1*   2.4992 0.013875      0.4686
t2*   0.2833 0.001998      0.0852
```

The output above gives us the specification of our bootstrap job (that is, the *Call*) along with bootstrapped estimates of bias and standard error for our risk ratio (row 1) and risk difference (row 2). Be mindful of the order in which we exported our statistics within `estimate_risk_measures()`.

We can plot a histogram to summarize the bootstrapped distribution of risk ratios and risk differences.

```r
plot(indirect_boot, index = 1)
```

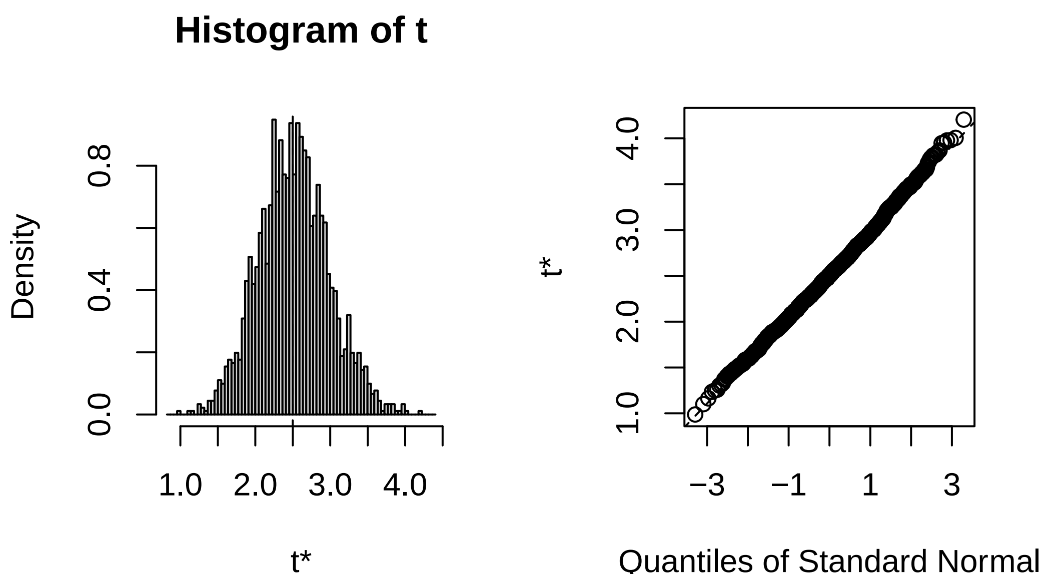


Figure 2: Bootstrapped distribution of risk ratios

```r
plot(indirect_boot, index = 2)
```

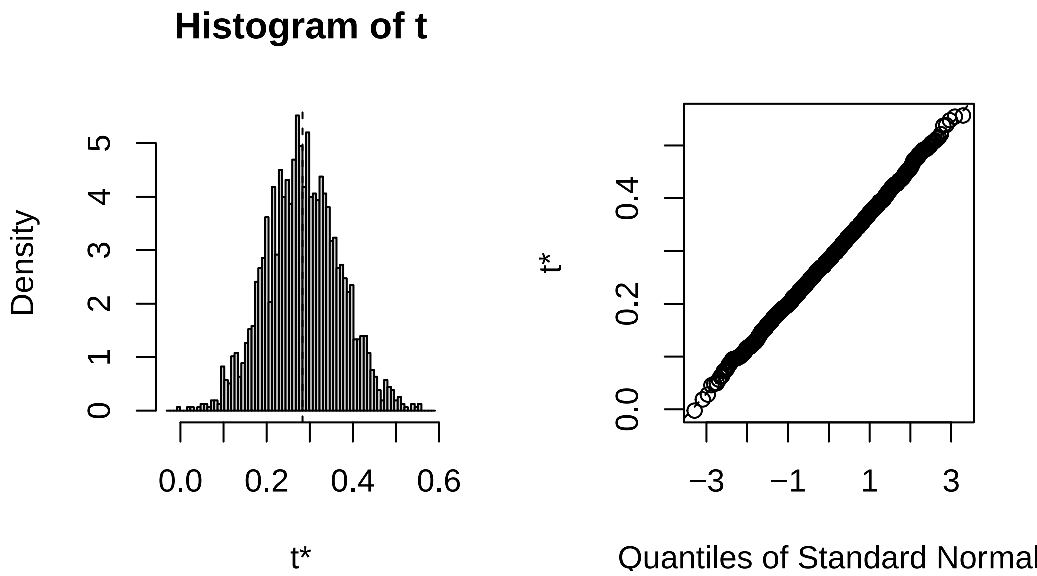


Figure 3: Bootstrapped distribution of risk differences

6.6 Calculate bootstrapped 95% confidence intervals

In the code below, we ask `boot()` for both the standard percentile and bias-corrected and adjusted (BCa) confidence intervals via the `type` argument. These methods have strengths and drawbacks that depend on the statistic of interest (Chernick and Labudde 2009), though BCa intervals may be preferable for general purpose estimation of common point estimates in epidemiology (Carpenter and Bithell 2000).

Bootstrapped confidence intervals for the risk ratio:

```
indirect_boot_ci_rr <- boot::boot.ci(
  indirect_boot,
  index = 1,  # risk ratio
  conf = 0.95,
  type = c("perc", "bca")
)

indirect_boot_ci_rr
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1999 bootstrap replicates

CALL :
```

```
boot::boot.ci(boot.out = indirect_boot, conf = 0.95, type = c("perc",
    "bca"), index = 1)

Intervals :
Level     Percentile            BCa
95%   ( 1.608,  3.470 )   ( 1.598,  3.462 )
Calculations and Intervals on Original Scale
```

Bootstrapped confidence intervals for the risk difference:

```r
indirect_boot_ci_rd <- boot::boot.ci(
  indirect_boot,
  index = 2,  # risk difference
  conf = 0.95,
  type = c("perc", "bca")
)

indirect_boot_ci_rd
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1999 bootstrap replicates

CALL :
boot::boot.ci(boot.out = indirect_boot, conf = 0.95, type = c("perc",
    "bca"), index = 2)

Intervals :
Level     Percentile            BCa
95%   ( 0.1182,  0.4536 )   ( 0.1189,  0.4554 )
Calculations and Intervals on Original Scale
```

6.7 Estimating a risk ratio using the `logisticRR` package

The `logisticRR` package will estimate marginal and conditional risk ratios using logistic regression via a simple interface. Below is an example of a simple specification.

```r
rr_lrr <- logisticRR::logisticRR(death ~ weakheart,
                                 data = nhefs,
                                 boot = TRUE,
                                 n.boot = 1999)
```

The `logisticRR()` function will conduct bootstrapping and calculate an estimate of the risk ratio's variance using the Delta method.

We can extract the appropriate quantiles for a 95% confidence interval from the bootstrap distribution produced by `logisticRR` as follows.

```
quantile(rr_lrr$boot.rr, c(0.025, 0.975))
```

```
 2.5% 97.5%
1.628 3.468
```

We could also construct 95% confidence intervals using the estimate of the variance based on the Delta method:

```
c(log(rr_lrr$RR) - 1.96 * rr_lrr$delta.var,
  log(rr_lrr$RR) + 1.96 * rr_lrr$delta.var) |> exp()
```

```
    1     1
1.654 3.777
```

7 References

Carpenter, J, and J Bithell. 2000. "Bootstrap Confidence Intervals: When, Which, What? A Practical Guide for Medical Statisticians." *Statistics in Medicine* 19 (9): 1141–64.

Chernick, Michael R, and Robert A Labudde. 2009. "Revisiting Qualms about Bootstrap Confidence Intervals." *American Journal of Mathematical and Management Sciences* 29 (3-4): 437–56. https://doi.org/10.1080/01966324.2009.10737767.

Efron, Bradley, and R J Tibshirani. 1994. *An Introduction to the Bootstrap*. CRC Press.

Kopp, Katherin. 2021. "COVID Vaccine Distribution Shiny App Walkthrough (Mock Data)." Posit. May 13, 2021. https://posit.co/resources/videos/covid-vaccine-distribution-shiny-app-walkthrough/.

R Core Team. 2023. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

8 Session Information

```r
sessioninfo::session_info()
```

```
- Session info ---------------------------------------------------------------
 setting  value
 version  R version 4.3.3 (2024-02-29)
 os       Ubuntu 22.04.4 LTS
 system   x86_64, linux-gnu
 ui       X11
 language
 collate  en_US.UTF-8
 ctype    en_US.UTF-8
 tz       America/New_York
 date     2024-05-29
 pandoc   2.9.2.1 @ /usr/bin/ (via rmarkdown)

- Packages -------------------------------------------------------------------
 package     * version    date (UTC) lib source
 backports     1.4.1      2021-12-13 [1] CRAN (R 4.3.3)
 boot        * 1.3-29     2024-02-19 [2] CRAN (R 4.3.3)
 broom       * 1.0.5      2023-06-09 [1] CRAN (R 4.3.1)
 cli           3.6.2      2023-12-11 [1] CRAN (R 4.3.2)
 codetools     0.2-19     2023-02-01 [2] CRAN (R 4.3.3)
 colorspace    2.1-0      2023-01-23 [1] CRAN (R 4.3.1)
 data.table  * 1.15.4     2024-03-30 [1] CRAN (R 4.3.3)
 digest        0.6.35     2024-03-11 [1] CRAN (R 4.3.3)
 dplyr         1.1.4      2023-11-17 [1] CRAN (R 4.3.2)
 evaluate      0.23       2023-11-01 [1] CRAN (R 4.3.2)
 fansi         1.0.6      2023-12-08 [1] CRAN (R 4.3.2)
 fastmap       1.1.1      2023-02-24 [1] CRAN (R 4.3.2)
 finalfit    * 1.0.7      2023-11-16 [1] CRAN (R 4.3.2)
 forcats       1.0.0      2023-01-29 [1] CRAN (R 4.3.1)
 foreach       1.5.2      2022-02-02 [1] CRAN (R 4.3.3)
 generics      0.1.3      2022-07-05 [1] CRAN (R 4.3.3)
 ggplot2     * 3.5.1      2024-04-23 [1] CRAN (R 4.3.3)
 glm2          1.2.1      2018-08-11 [1] CRAN (R 4.3.2)
 glmnet        4.1-8      2023-08-22 [1] CRAN (R 4.3.3)
 glue          1.7.0      2024-01-09 [1] CRAN (R 4.3.2)
 gtable        0.3.5      2024-04-22 [1] CRAN (R 4.3.3)
 htmltools     0.5.7      2023-11-03 [1] CRAN (R 4.3.2)
```

```
iterators     1.0.14    2022-02-05 [1] CRAN (R 4.3.3)
itertools2    0.1.1     2014-08-08 [1] CRAN (R 4.3.2)
jomo          2.7-6     2023-04-15 [1] CRAN (R 4.3.2)
jsonlite      1.8.8     2023-12-04 [1] CRAN (R 4.3.2)
kableExtra  * 1.4.0     2024-01-24 [1] CRAN (R 4.3.3)
knitr       * 1.46      2024-04-06 [1] CRAN (R 4.3.3)
lattice       0.22-5    2023-10-24 [2] CRAN (R 4.3.3)
lifecycle     1.0.4     2023-11-07 [1] CRAN (R 4.3.1)
lme4          1.1-35.1  2023-11-05 [1] CRAN (R 4.3.2)
logbin      * 2.0.5     2021-08-09 [1] CRAN (R 4.3.2)
logisticRR    0.3.0     2020-04-03 [1] CRAN (R 4.3.3)
magrittr    * 2.0.3     2022-03-30 [1] CRAN (R 4.3.3)
MASS          7.3-60.0.1 2024-01-13 [2] CRAN (R 4.3.3)
Matrix        1.6-5     2024-01-11 [2] CRAN (R 4.3.3)
mice          3.16.0    2023-06-05 [1] CRAN (R 4.3.2)
minqa         1.2.6     2023-09-11 [1] CRAN (R 4.3.2)
mitml         0.4-5     2023-03-08 [1] CRAN (R 4.3.2)
munsell       0.5.1     2024-04-01 [1] CRAN (R 4.3.3)
nlme          3.1-164   2023-11-27 [2] CRAN (R 4.3.3)
nloptr        2.0.3     2022-05-26 [1] CRAN (R 4.3.2)
nnet          7.3-19    2023-05-03 [2] CRAN (R 4.3.3)
pacman        0.5.3     2023-11-15 [1] Github (trinker/pacman@ace0936)
pan           1.9       2023-12-07 [1] CRAN (R 4.3.2)
pillar        1.9.0     2023-03-22 [1] CRAN (R 4.3.1)
pkgconfig     2.0.3     2019-09-22 [1] CRAN (R 4.3.3)
purrr         1.0.2     2023-08-10 [1] CRAN (R 4.3.1)
R6            2.5.1     2021-08-19 [1] CRAN (R 4.3.3)
Rcpp          1.0.12    2024-01-09 [1] CRAN (R 4.3.2)
rlang         1.1.3     2024-01-10 [1] CRAN (R 4.3.2)
rmarkdown     2.25      2023-09-18 [1] CRAN (R 4.3.2)
rpart         4.1.23    2023-12-05 [2] CRAN (R 4.3.3)
rstudioapi    0.15.0    2023-07-07 [1] CRAN (R 4.3.2)
sandwich    * 3.1-0     2023-12-11 [1] CRAN (R 4.3.2)
scales        1.3.0     2023-11-28 [1] CRAN (R 4.3.2)
sessioninfo * 1.2.2     2021-12-06 [1] CRAN (R 4.3.3)
shape         1.4.6.1   2024-02-23 [1] CRAN (R 4.3.3)
stringi       1.8.3     2023-12-11 [1] CRAN (R 4.3.2)
stringr       1.5.1     2023-11-14 [1] CRAN (R 4.3.3)
survival      3.5-8     2024-02-14 [2] CRAN (R 4.3.3)
svglite       2.1.2     2023-10-11 [1] CRAN (R 4.3.2)
systemfonts   1.0.6     2024-03-07 [1] CRAN (R 4.3.3)
tibble        3.2.1     2023-03-20 [1] CRAN (R 4.3.1)
tidyr         1.3.1     2024-01-24 [1] CRAN (R 4.3.2)
```

```
 tidyselect    1.2.1      2024-03-11 [1] CRAN (R 4.3.3)
 utf8          1.2.4      2023-10-22 [1] CRAN (R 4.3.1)
 vctrs         0.6.5      2023-12-01 [1] CRAN (R 4.3.2)
 viridisLite   0.4.2      2023-05-02 [1] CRAN (R 4.3.1)
 withr         3.0.0      2024-01-16 [1] CRAN (R 4.3.2)
 xfun          0.43       2024-03-25 [1] CRAN (R 4.3.3)
 xml2          1.3.6      2023-12-04 [1] CRAN (R 4.3.2)
 yaml          2.3.8      2023-12-11 [1] CRAN (R 4.3.3)
 zoo           1.8-12     2023-04-13 [1] CRAN (R 4.3.2)

 [1] /home/jrgant/R/x86_64-pc-linux-gnu-library/4.3
 [2] /usr/local/lib/R/library

------------------------------------------------------------------------------
```