Lab 4

Public Health 241: Statistical Analysis of Categorical Data

YOUR NAME / YOUR STUDENT ID HERE

TODAY'S DATE

In this lab, we'll cover tools you'll need to complete Homework 5, including:

- Hypothesis testing
- Confidence intervals
- Obtaining critical values and p-values from Gaussian and χ^2

1 Hypothesis tests, confidence intervals, and χ^2

We'll be using the WCGS dataset, as in lab Load it into R. In this dataset, each person in the study is represented by a row. Each row contains, among other variables, the values of two binary (0/1 indicator) variables called chd69 (Coronary Heart Disease) and arcus0 (Arcus senilis, a ring in the corneal margin or around the iris that can indicate CHD).

1.1 The Normal approximation for proportions, exact confidence intervals, and Fisher's exact test

Using the Normal curve for inferences about proportions is actually an approximation. For confidence intervals for a proportion, the number of observed successes and failures should be at least 10. For hypothesis tests about a proportion, the expected numbers of successes and failures should be at least 10. For two-sample problems with binary outcomes, where the data are summarized in a 2 x 2 table, the criteria are a bit different – many authors say that the observed counts in each cell should be greater than 5 to use the Normal approximation, and for the two-sample test for proportion or the equivalent χ^2 test, the expected counts should all be greater than 5.

What if our data don't satisfy the assumptions for the Normal or χ^2 distributions? For these situations, there are exact tests based on the binomial distribution and its relatives. Using `epi.2by2` from the `epiR` package, R allows us to request exact p-values and construct "exact" confidence intervals.

1.1.1 Creating a 2 x 2 table

We begin by creating a 2 x 2 table, using the `table` function:

Generic: `table(data$dependentvariable, data$independent_variable)`

```
chd.arc <- table(wcgs$chd69, wcgs$arcus0)
chd.arc
```

```
##
##        0    1
##   0 2058  839
##   1  153  102
```

This creates a basic 2 x 2 table for our data. However, in order to pass this to `epi.2by2` to do the calculations, we need to have the disease/exposed count on the upper-lefthand corner and the non-disease/non-exposed count on the lower right-hand corner.

1.1.2 Modifying the table to fit `epi.2by2`

Since our data is in the wrong order, we need to reorder our table before we can pass it to `epi.2by2`. To do this, we can use `relevel`.

Note: you only need to do this if your data is in the wrong order

```
chd.arc <- table(relevel(as.factor(wcgs$chd69), '1'), relevel(as.factor(wcgs$arcus0), '1'))
chd.arc
```

```
##
##        1    0
##   1  102  153
##   0  839 2058
```

1.1.3 Using `epi.2by2`

We can now pass our table to `epi.2by2` to tabulate our data.

Generic: `epi.2by2(table)`

```
epi.2by2(chd.arc)
```

```
##              Outcome +    Outcome -      Total        Inc risk *
## Exposed +          102          153        255              40.0
## Exposed -          839         2058       2897              29.0
## Total              941         2211       3152              29.9
##                   Odds
## Exposed +        0.667
## Exposed -        0.408
## Total            0.426
##
## Point estimates and 95 % CIs:
## -------------------------------------------------------------------
## Inc risk ratio                                 1.38 (1.18, 1.62)
## Odds ratio                                     1.64 (1.26, 2.13)
## Attrib risk *                                  11.04 (4.80, 17.27)
## Attrib risk in population *                    0.89 (-1.40, 3.19)
## Attrib fraction in exposed (%)                 27.60 (14.97, 38.35)
## Attrib fraction in population (%)              2.99 (1.25, 4.70)
## -------------------------------------------------------------------
##  X2 test statistic: 13.638 p-value: < 0.001
##  Wald confidence limits
##  * Outcomes per 100 population units
```

1.1.4 Specifying the confidence interval

The default confidence level for `epi.2by2` is 95%. To override this default, reassign the `conf.level` parameter. The confidence level must be between 0 and 1.

Generic: `epi.2by2(table, conf.level = 0.99)`

```
epi.2by2(chd.arc, conf.level = 0.99)
```

```
##              Outcome +    Outcome -      Total        Inc risk *
## Exposed +          102          153        255              40.0
```

```
## Exposed -          839         2058       2897              29.0
## Total              941         2211       3152              29.9
##                   Odds
## Exposed +        0.667
## Exposed -        0.408
## Total            0.426
##
## Point estimates and 99 % CIs:
## -------------------------------------------------------------------
## Inc risk ratio                                 1.38 (1.12, 1.71)
## Odds ratio                                     1.64 (1.16, 2.31)
## Attrib risk *                                  11.04 (2.84, 19.23)
## Attrib risk in population *                    0.89 (-2.13, 3.91)
## Attrib fraction in exposed (%)                 27.60 (10.56, 41.39)
## Attrib fraction in population (%)              2.99 (0.70, 5.23)
## -------------------------------------------------------------------
##  X2 test statistic: 13.638 p-value: < 0.001
##  Wald confidence limits
##  * Outcomes per 100 population units
```

1.1.5 Specifying the test method

`epi.2by2` can also calculate appropriate measures of association and measures of effect in the exposed and total population based on the study design upon which the data was collected. By default, `epi.2by2` uses cohort-count. To override this default, reassign the `method` parameter. Options are cohort.count, cohort.time, case.control, or cross.sectional

Generic: `epi.2by2(table, method = "method")`

```
epi.2by2(chd.arc, method = "case.control")
```

```
##              Outcome +    Outcome -      Total        Prevalence *
## Exposed +          102          153        255                40.0
## Exposed -          839         2058       2897                29.0
## Total              941         2211       3152                29.9
##                   Odds
## Exposed +        0.667
## Exposed -        0.408
## Total            0.426
##
## Point estimates and 95 % CIs:
## -------------------------------------------------------------------
## Odds ratio (W)                                 1.64 (1.26, 2.13)
## Attrib prevalence *                            11.04 (4.80, 17.27)
## Attrib prevalence in population *              0.89 (-1.40, 3.19)
## Attrib fraction (est) in exposed  (%)          38.84 (19.59, 53.33)
## Attrib fraction (est) in population (%)        4.21 (1.79, 6.58)
## -------------------------------------------------------------------
##  X2 test statistic: 13.638 p-value: < 0.001
##  Wald confidence limits
##  * Outcomes per 100 population units
```

1.2 Obtaining critical values and confidence levels

We learned previously how to find the p-value associated with a particular χ^2 test statistic by using `pchisq`: *Remember this value is the area in the right hand tail of the χ^2 distribution with a specified number of degrees of freedom.*

```
# 1 is the number of degrees of freedom and 13.6382 is our
# test statistic from the WCGS dataset example above
1 - pchisq(13.6382, df = 1)
```

```
## [1] 0.0002216298
```

```
# or, more elegantly:
pchisq(13.6382, df = 1, lower.tail = FALSE)      # recommended
```

```
## [1] 0.0002216298
```

This is because all cumulative probability functions in R compute left tail probabilities by default. To find the right tail probability, set `lower.tail = FALSE`. This is recommended over taking $1 - p$.

We can similarly find the p-value associated with a particular test statistic (z-value) based on the Standard Normal distribution. For a 2-sided test, this would look like:

```
2 * (1 - pnorm(1.96))
```

```
## [1] 0.04999579
```

1.2.1 Inverses

The `pchisq`, and `pnorm` functions have inverses `qchisq` and `qnorm` respectively. These functions take a probability as input and will give you its associated critical value. Be careful about making adjustments for 2-tailed tests if you want a particular confidence level. For example, for a 2-tailed test at the 95% confidence level (where $\alpha = 0.05$), the critical z-value is found as follows:

```
qnorm(.975)
```

```
## [1] 1.959964
```

Looking up two tails of the χ^2 distribution for a hypothesis test is only appropriate when con- structing a confidence interval. Recall that the χ^2 distribution is not symmetric, so it is necessary to to look up both tails separately.

```
qchisq(.025, 1)           # where 1 is the degree of freedom
```

```
## [1] 0.0009820691
```

```
qchisq(.975, 1)
```

```
## [1] 5.023886
```

2 Creating and modifying variables with R

Objects are created in R using the assignment operator `<-`. For example, to create the `population.size` object that stores `1000`, type the following:

```
# assigns 1000 to population.size
population.size <- 1000
```

```
# to print the value of a object, simply enter the object name:
population.size
```

```
## [1] 1000
```

To change an object value, just reassign it with the same operator. However, note that this will clear the previous value completely.

```
# reassign population.size to 2000
population.size <- 2000
population.size
```

```
## [1] 2000
```

To represent a variable in R, we use a data vector, which is list of values of the same type. Think of this as a column in a dataset. To create a vector, use the command `c`.

Generic: 'c(value1, value2, value3 . . .)

For example, let's create a vector to represent the variable age for a cohort of 5 people ages 25, 36, 35, 65, and 8:

```
age <- c(25, 36, 35, 65, 8)
```

2.1 Conditioned variables

Perhaps the age data in your dataset is from 2014 and you now need their ages in 2019. How do we adjust the age vector? It's simple—R handles vector addition very nicely. If you add two vectors of the same length the elements will add element-wise; if you add an integer to a vector, that integer will be added to each of the elements in the vector as follows:

```
age.2019 <- age + 4
age.2019
```

```
## [1] 29 40 39 69 12
```

If you want a new variable `adult` to take the value 0 when `age` is under 21 and 1 when `age` is 21+, you can do the following using the `ifelse` function. The `ifelse()` function will evaluate the `condition` parameter and if it evaluates to `TRUE`, will return the `yes` value. Otherwise, the `condition` evaluates to `FALSE` and the function returns the `no` value.

Generic: `ifelse(condition, yes, no)`

```
adult <- ifelse(age >= 21, 1, 0)
adult
```

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

Instead of a number, say you want ages under 21 to be recoded as "child" and 21+ to be recoded as "adult". You can do so again using the `ifelse` function, but supplying strings instead of integers for the `yes` and `no` values:

```
age.category <- ifelse(age >= 21, "adult", "child")
age.category
```

```
## [1] "adult" "adult" "adult" "adult" "child"
```

For your reference: a complete list of comparison operators *(these commands won't work in the current environment as the variables a and b don't exist):*

```
a > b     # greater than
a < b     # less than
a >= b    # greater than or equal to
a <= b    # less than or equal to
a == b    # equal to
a != b    # not equal to
```

IMPORTANT: Note that a single equal sign = is similar to <- and will work as an object assignment, whereas the double equals == will compute a comparison.

2.2 Boolean expressions

However, if we are just coding binary values (0/1), we don't have to use `ifelse()` at all. In fact, we can just assign the new variable based on a boolean expression, an expression that evaluates to a `TRUE` or `FALSE` value.

```
# option 1:
adult1 <- age >= 21
adult1
```

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

```
# option 2:
adult2 <- !(age < 21)      # ! is the negation operator
adult2
```

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

As you can see, there are two ways to code this, but option 1 is preferred because it allows our code to be very intuitive: the `adult` variable takes a `TRUE` (1) value when age is >= 21.

In R, `TRUE` is represented numerically as 1 and `FALSE` as 0. Conversely, 0 evaluates to `FALSE` and all other integers evaluate to `TRUE`. We can see this if we add 0 to `adult`, which effectively turns a `boolean` type to a `numeric`.

```
adult <- adult + 0
adult
```

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

```
age <- c("infant", "infant", "adult", "adult", "adult", "infant")
heart.rate <- c(132, 96, 42, 64, 86, 210)
```

2.3 AND, OR, NOT

These boolean expressions can be very powerful with the introduction of AND, OR and NOT. For example, consider a dataset with variables `age` and `heart.rate`. To encode a new variable `healthy.hr` that is dependent on both `age` and `heart.rate`, you can type the following:

```
healthy.hr <- (age == "infant" & heart.rate > 80 & heart.rate < 160) |
              (age == "adult" & heart.rate > 60 & heart.rate < 100)
healthy.hr
```

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