**Nominal Categorical Response in Regression-Based Analysis**

Benjamin Bui

October 10, 2024

Categorical variables provide an inherent problem when attempting to describe relationships mathematically. Many analyses we run involve regression analyses which inherently require that **ALL** variables be treated numerically. Categorical variables are normally converted into numerical variables in one of two ways, either via a direct scale mapping or by converting to {0, 1} dummy variables. This memo argues why this method, when applied to non-binary dependent variables, is statistically problematic and offers alternative solutions to the issue.

This memo is divided into three main sections. The first section describes categorical variables, both statistically and how they are handled in R. This is followed by a statistical argument against the use of multiple dummy variables as a proxy for a nominal categorical variable when dealing with a multinomial nominal categorical response variable. Finally, we go through solutions for how to handle this statistical problem in R.

[1. Categorical variables, in general and in R 1](#_Toc179854202)

[1.1. What is a categorical and dummy variable? 1](#_Toc179854203)

[1.2. Variable types as R classes 3](#_Toc179854204)

[2. The potential problems with dummy response variables 6](#_Toc179854205)

[2.1. Multiple comparisons with dummy variables 7](#_Toc179854206)

[3. Avoiding dummy response variables 8](#_Toc179854207)

[3.1. The simple test 9](#_Toc179854208)

[3.2. Multinomial logistic regression 11](#_Toc179854209)

[3.2.1. -Computation, SUR, and Adjusted Permutation 15](#_Toc179854210)

[3.3. Linear mixed-effects model 16](#_Toc179854211)

[3.4. Linear mixed-effects model with random intercept level covariates 21](#_Toc179854212)

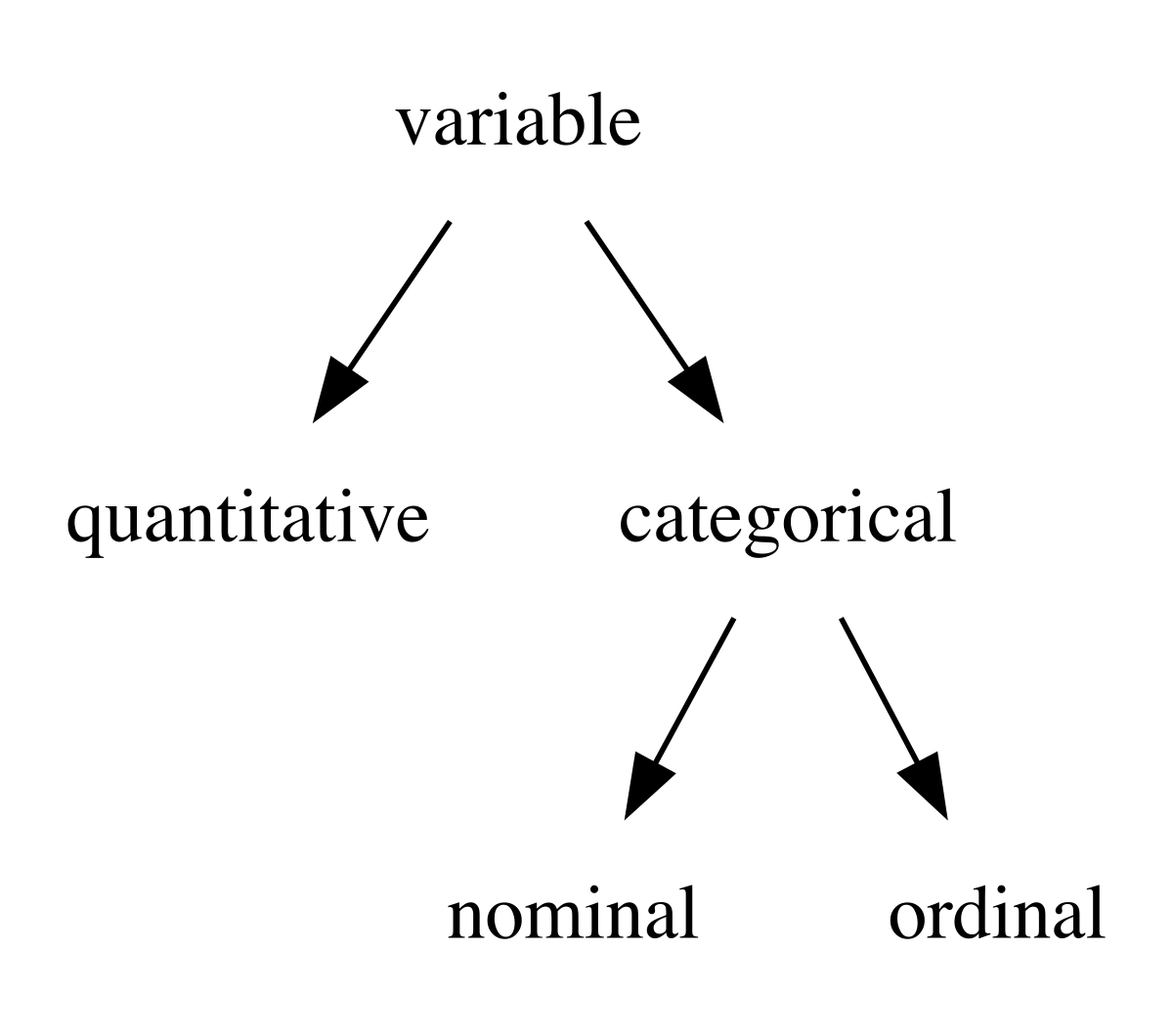
[3.4.1. Alternative methods for mixed-effect models 24](#_Toc179854213)

[4. References 25](#_Toc179854214)

# Categorical variables, in general and in R

## What is a categorical and dummy variable?

For our purposes, we can characterise variables as one of three types: quantitative, nominal categorical, and ordinal categorical (Sinharay, 2010). There is also a meaningful distinction between categorical variables with just two levels and those with more than two levels that we will also go into shortly.



A variable is quantitative if it is inherently ordered, and has an internally consistent, scalable definition of difference. For example, consider a variable representing age. Given any two values of age we can state that one age value is greater than or the same as the other age value. Moreover, this relationship is transitive as can be seen with the following.

12 years of age is greater than 8 years of age. 8 years of age is greater than 4 years of age. Thus 12 years of age is necessarily also greater than 4 years of age.

The difference property can be seen with how we can describe age with arbitrary precision. We can understand what is meant when we say something is 5 years of age, but we can also understand what is meant when we say something is 4.999999999 years of age. Moreover the distance between 4.999999999 and 5 years of age is the same as 40.999999999 and 50 years of age. Although these properties seem trivial, indeed they are just a subset of the properties of the real numbers, categorical variables break with some of these assumptions.

First, we will break with the scalability property. Suppose we have a variable with the possible values: {“Less than high school”, “High school”, “Bachelor’s Degree”, “Greater than Bachelor’s Degree”}. We can order this such that the ordering represents the average amount of time for completion: {“Less than high school” < “High school” < “Bachelor’s Degree” < “Greater than Bachelor’s Degree”}. The variable as it is represented doesn’t allow for any further precision because there are no in-between values. This type of variable is known as an ordered categorical variable, or an ordinal variable.

Now we will break with the ordering property. Here we’ll use a fruit variable with the possible values of {“apple”, “orange”, “grape”}. There is no agreed upon way to order these values so we will refer to this as a nominal categorical variable, or a nominal variable for short.

It is important to consider that many variables can be considered quantitative, ordinal, or nominal depending on your interpretation and goals. Age was what we used as an example of a quantitative variable. We can effectively transform this into an ordinal variable by breaking up the age into the possible values: {“Younger than 3”, “3 to 18 years”, “Older than 18”}. We can also transform this into a nominal variable by making it into the variable {“3 to 18 years”, “Not 3 to 18 years”}. All three of these variables describe age despite the differences in their representation.

The nominal variable we made also qualifies as a dummy variable, or a binary variable. Binary and dummy variables are, for most intents and purposes, the same thing just from the perspective of a programmer and a statistician, respectively. The only difference between them is that dummy variables are technically always coded as a {0, 1} variable and that they are used to indicate the presence of some categorical trait. Dummy variables break up a single categorical variable into multiple different variables, each representing a specific possible value of the original categorical. Dummy variables are coded with a value space of {0, 1} because it allows for convenient mathematical manipulation and interpretation. It is possible to create dummy variables from a categorical variable with possible levels; however, dummy variables encode the same amount of information. For example, consider our fruit categorical variable {“apple”, “orange”, “grape”}. We can separate this into an apple dummy variable that is 1 if the fruit is “apple” and 0 otherwise and an orange dummy variable that is 1 if the fruit is “orange” and 0 otherwise. If both apple and orange is 0, then we know that the fruit must be “grape” which makes creating an explicit grape variable redundant and can introduce issues of perfect multicollinearity.

## Variable types as R classes

These three types of variables: quantitative, nominal, and ordered are represented in R as different classes of vector. Quantitative variables should be represented by the numeric class, nominal variables by the factor class, and ordinal variable by a combination of the ordered and factor classes.

$ has 25 different types of vectors at its lowest level, most relevant here are integer, double, and character (*Advanced R, Second Edition*, 2019). You may notice that factor is not included here, that is because factor isn’t actually implemented as a low level vector type, rather it is implemented via a integer vector and character vector. A factor stores its data as an integer vector but also includes an attribute called levels This levels attribute is a character vector representing the sample space of your categorical variable. Unless the factor is ordered, the integers do **NOT** store or intend any meaning as a number. They are only stored as integer because it is very space efficient to do so. Adding ordered as a class to a factor, usually by setting ordered = TRUE when defining a factor just adds ordered as a class but doesn’t do anything else to the underlying implementation.

Let’s take at this using iris$Species.

# Taking a subset for readability  
iris\_subset <- iris[sample(seq\_len(nrow(iris)), 20), ]  
iris\_subset$Species

## [1] versicolor setosa virginica versicolor virginica setosa   
## [7] versicolor setosa versicolor setosa virginica virginica   
## [13] virginica versicolor virginica versicolor setosa versicolor  
## [19] virginica versicolor  
## Levels: setosa versicolor virginica

The actual values of Species are stored as the integer vector:

typeof(iris\_subset$Species)

## [1] "integer"

unclass(iris\_subset$Species)

## [1] 2 1 3 2 3 1 2 1 2 1 3 3 3 2 3 2 1 2 3 2  
## attr(,"levels")  
## [1] "setosa" "versicolor" "virginica"

And the levels are stored as the character vector:

levels(iris$Species)

## [1] "setosa" "versicolor" "virginica"

This mapping means that all values of 1 correspond to “setosa”, 2 corresponds to “versicolor”, and 3 corresponds to “virginica”. Converting this into an ordered factor is simple (although I recommend being more explicit with your ordering).

as.ordered(iris\_subset$Species)

## [1] versicolor setosa virginica versicolor virginica setosa   
## [7] versicolor setosa versicolor setosa virginica virginica   
## [13] virginica versicolor virginica versicolor setosa versicolor  
## [19] virginica versicolor  
## Levels: setosa < versicolor < virginica

Here, we see the only difference in presentation is that the levels now use < instead of , as the delimiter which represents which levels are greater than which. R the language doesn’t treat these variables differently, again the only difference is the addition of the ordered class.

attributes(iris\_subset$Species)

## $levels  
## [1] "setosa" "versicolor" "virginica"   
##   
## $class  
## [1] "factor"

attributes(as.ordered(iris\_subset$Species))

## $levels  
## [1] "setosa" "versicolor" "virginica"   
##   
## $class  
## [1] "ordered" "factor"

This additional class is used by modelling packages to differentiate between nominal and ordered categorical variables and to apply different methods to these variables. Consider the most common method for fitting linear models: stats::lm. When given a plain factor as a covariate, stats::lm creates dummy variables and runs the model treating these dummy variable as if they were quantitative. However, if you pass an ordered factor as a covariate, then stats::lm will treat the variable as a polynomial model with degrees. A polynomial model assumes that the levels of the ordered factor are equally spaced and that the space is continuous. The results it presents evaluate how well the relationship between the ordered factor and response variable can be described as linear, quadratic, cubic, etc. This is a choice by the authors of the stats package on how to handle ordered factor covariates and the exact way that any modelling package handles these variables can differ. Treating the categorical variable as a polynomial model isn’t better or worse than treating it as dummies, it is just important that you select the appropriate method for your data and intention.

It can be confusing to think of a plain factor as unordered because the order that the levels attribute is defined as can still have a practical meaning even if it doesn’t have a statistical or mathematical one. For example, suppose we want to sort a dataset or plot such that certain values of a categorical variable appear first. We aren’t implying that any level is greater or less than the other, we just want our data to appear a certain way. A simple way to do this would be to adjust the levels such that the values we want to see first are first in the actual levels attribute character vector. The first value in the levels attribute is also used when calculating contrasts as the “baseline”. For example, consider our fruit variable: {“apple”, “orange”, “grape”}. If we used this variable in a model and requested contrasts on fruit, then our results would, by default, present the impact of “orange” compared to “apple” and the impact of “grape” compared to “apple”. This doesn’t have any impact in the computation of the model itself but it affects the direct interpretability of the coefficients. However, because we shouldn’t make a variable an ordered factor unless we explicitly want it to be treated as an ordinal, we can instead just change the level attribute. A simple way of doing this would be using stats::relevel or forcats::fct\_relevel

iris\_subset$Species

## [1] versicolor setosa virginica versicolor virginica setosa   
## [7] versicolor setosa versicolor setosa virginica virginica   
## [13] virginica versicolor virginica versicolor setosa versicolor  
## [19] virginica versicolor  
## Levels: setosa versicolor virginica

# relevel is great to just set 1 new level at the start  
relevel(iris\_subset$Species, "virginica")

## [1] versicolor setosa virginica versicolor virginica setosa   
## [7] versicolor setosa versicolor setosa virginica virginica   
## [13] virginica versicolor virginica versicolor setosa versicolor  
## [19] virginica versicolor  
## Levels: virginica setosa versicolor

# fct\_relevel is useful for more complex re-ordering operations  
forcats::fct\_relevel(iris\_subset$Species, "virginica", "versicolor", "setosa")

## [1] versicolor setosa virginica versicolor virginica setosa   
## [7] versicolor setosa versicolor setosa virginica virginica   
## [13] virginica versicolor virginica versicolor setosa versicolor  
## [19] virginica versicolor  
## Levels: virginica versicolor setosa

# The potential problems with dummy response variables

This section will cover why we do not want to substitute nominal categorical variables with dummy variables.

Transforming nominal categorical variables into dummy variables is accepted practice when being done on predictor variables for a mode. As stated previously, this is how most statistical modelling packages in R automatically handle factor variables. It intuitively makes sense then that we apply the same to nominal categorical response variables; to just run models for each dummy variable. However, consider the interpretability of our results if only 1 of these analyses provides a statistically significant p-value. Does that mean our explanatory variable only effectively predicts for one possible value of our categoricals? That conclusion is difficult to reconcile with our knowledge that our dummy variables are collinear and mutually exclusive with each other. For this eason, a single, shared statistical test would be preferred when evaluating a categorical variable. This would allow us to better evaluate the impact of the explanatory variable on the categorical response.

Another important consideration pertains to multiple hypothesis testing. Given some impact analysis, we have a null hypothesis and an alternate hypothesis . Although the specifics will differ, these two hypotheses can be generally summarised as:

The true population treatment and control groups are the same.

The true population treatment and control groups are not the same.

The p-value result you get from running a statistical test tells you the likelihood that, given the assumptions particular to your choice of test, that you would have achieved the result found given . A p-value of 0.05 thus means that there is a 5% chance that, given the treatment haa no effect, you would have oberved the differences found. Although 5% is a fairly small number, it is not 0 and it becomes practically inevitable that we encounter an erroneously find a statistically significant result the more tests we run. This issue is commonly referred to as the multiple comparisons problem and the use of dummied outcome variables can inflate the number of outcomes such that this can become a much more significant issue.

## Multiple comparisons with dummy variables

Here I will be creating a randomly generated dataset to demonstrate the potential pitfalls of using dummy variables as a proxy for a nominal categorical variable in the context of a response variable. This dataset will consist of 1000 observations with the following variables:

* Environment - Random sampling with choices: {“Urban”, “Suburban”, “Rural”}
* Education - Random sampling with choices: {“Bachelors”, “Graduate”}
* Month - Random sampling with choices: {“Jan”, “Feb”, “Mar”, “Apr”, “May”, “Jun”, “Jul”, “Aug”, “Sep”, “Oct”, “Nov”, “Dec”}
* State - Random sampling with choices: {“NY”, “CA”, “DC”}
* Letter - Random sampling from all available lowercase letters in the English alphabet

suppressPackageStartupMessages(library(tidyverse))  
# Generating dataset with 5 nominal categorical variables drawn from a uniform  
# distribution  
test\_categoricals <- data.frame(  
 Environment = sample(factor(c("Urban", "Suburban", "Rural")), 1000, replace = TRUE),  
 Education = sample(factor(c("Bachelors", "Gradauate")), 1000, replace = TRUE),  
 Month = sample(factor(month.abb), 1000, replace = TRUE),  
 State = sample(factor(c("NY", "DC", "CA")), 1000, replace = TRUE),  
 Letter = sample(factor(letters), 1000, replace = TRUE),  
 # The variable we are using as our independent  
 Intervention = sample(factor(c("Control", "Intervention")), 1000, replace = TRUE)  
)

# Creating dummied version of dataset  
test\_dummies <- fastDummies::dummy\_cols(  
 test\_categoricals,  
 c("Environment", "Education", "Month", "State", "Letter"),  
 remove\_selected\_columns = TRUE  
)

Since our variables are created randomly, there should be no relationship between the intervention group and any of the nominal categorical generated. However, running a simple linear model and looking at the results suggests otherwise.

# Running linear model comparisons  
test\_comparisons <- mdrcAnalysis::lm\_extract(test\_dummies,  
 .dependents = setdiff(  
 names(test\_dummies),  
 "Intervention"  
 ),  
 .treatment = "Intervention",  
 .inc\_sample = FALSE,  
 .inc\_trail = FALSE  
)

In lieu of printing the results of each test, let’s look at how the significance levels are distributed. Remember, all these variables were randomly and independently generated so there is no true relationship between any of them.

| p<X | n |
| --- | --- |
| 0.05 | 3 |
| 0.1 | 1 |
| >=0.1 | 42 |

We see no statistical significance in 42 of our 46 tests using a threshold of 0.1; however, there are four tests that display some significance, or about 8.7% of the results. Recall that we are really trying to determine the relationship between our treatment variable and five response variables, not 46. Although presenting 8.7% of these results as statistically significant isn’t too problematic in and of itself, it would be disingenuous to present the statistically significant results without either the context of the other, non-significant results, or with a more formalized multiple comparisons test.

# Avoiding dummy response variables

We ideally want a test that provides a singular result on our categorical response variable. Although we did discuss ordinal variables previously, that largely to avoid the accidental conflation of nominal and ordinal variables. The solutions presented henceforth should only be applied to nominal response variables.

The simplest solution to the issue is to use a quantitative variable instead. In cases where a categorical variable is generated from a quantitative variable, say when creating age blocks when exact birth dates are known, using the quantitative variable as the response instead can avoid much of this headache. Moreover, the choice of category boundaries itself, intentionally or not, can generate false significance (Stefan & Schönbrodt, 2023). There are many cases where we must use a categorical response variable: the funder wants results in terms of a specific categorical variable, the quantitative basis variable is heavily skewed, or the quantitative basis variable is unavailable or doesn’t exist. If this is the case, then continue on to one of the methods below.

## The simple test

Although a bit of an oversimplification, a test serves an analogous function to categorical variables as a -test is to quantitative variables. This test is useful when determining the relationship between two categorical variables, e.g. race and your treatment variable but it is not useful when attempting to account for other covariates. These tests are often used at MDRC for early investigations of data regarding baseline equivalence and non-response bias, both cases where there is a relatively clear categorical vs categorical relationship we are trying to ascertain.

Programming a test in R is simple. Here we are using the mdrcAnalysis::sim\_data\_robust\_reg dataset and comparing Employment (employed\_01) and treatment group (treatment).

data(sim\_data\_robust\_reg, package = "mdrcAnalysis")  
chisq.test(  
 sim\_data\_robust\_reg$employed\_01,  
 sim\_data\_robust\_reg$treatment, ,  
 correct = FALSE  
)

##   
## Pearson's Chi-squared test  
##   
## data: sim\_data\_robust\_reg$employed\_01 and sim\_data\_robust\_reg$treatment  
## X-squared = 62.502, df = 1, p-value = 2.661e-15

The broom package provided useful tools for extracting the results of statistical models, say for tabling or outputting into Excel. Please read ?stats::chisq.test for more details on what each column is referring to.

library(broom)  
tidy(chisq.test(  
 sim\_data\_robust\_reg$employed\_01,  
 sim\_data\_robust\_reg$treatment,  
))

| statistic | p.value | parameter | method |
| --- | --- | --- | --- |
| 60.5275 | 7.255772e-15 | 1 | Pearson's Chi-squared test with Yates' continuity correction |

Because these two variables are dummy variables, we can also treat them quantitatively and pass them to a paired, two-sided -test to compare our results.

tidy(t.test(  
 sim\_data\_robust\_reg$employed\_01,  
 sim\_data\_robust\_reg$treatment,  
 paired = TRUE  
))

| estimate | statistic | p.value | parameter | conf.low | conf.high | method | alternative |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0.2291022 | 8.207132 | 5.511062e-15 | 322 | 0.1741833 | 0.284021 | Paired t-test | two.sided |

This was done largely to demonstrate how a difference in interpretation of the variables can cause a real statistical impact. Both tests show very small p-values, much lower than the typical threshold of 0.05; however, the actual p-value does differ between the two as is highlighted below.

bind\_rows(  
 tidy(chisq.test(  
 sim\_data\_robust\_reg$employed\_01,  
 sim\_data\_robust\_reg$treatment,  
 )),  
 # Running without Yates correction  
 tidy(chisq.test(  
 sim\_data\_robust\_reg$employed\_01,  
 sim\_data\_robust\_reg$treatment,  
 correct = FALSE  
 )),  
 tidy(t.test(  
 sim\_data\_robust\_reg$employed\_01,  
 sim\_data\_robust\_reg$treatment,  
 paired = TRUE  
 ))  
) |>  
 select(statistic, p.value, method)

| statistic | p.value | method |
| --- | --- | --- |
| 60.527500 | 7.255772e-15 | Pearson's Chi-squared test with Yates' continuity correction |
| 62.502207 | 2.661463e-15 | Pearson's Chi-squared test |
| 8.207132 | 5.511062e-15 | Paired t-test |

Using a non-binary variable doesn’t introduce any complications. Here we replace Employment with Favorite Animal (favAnimals), a variable with three possible values. The only practical difference is that the Yates continuity correction is only applicable to binary-on-binary comparisons.

tidy(  
 chisq.test(sim\_data\_robust\_reg$favAnimals, sim\_data\_robust\_reg$treatment)  
)

| statistic | p.value | parameter | method |
| --- | --- | --- | --- |
| 1.674965 | 0.4327987 | 2 | Pearson's Chi-squared test |

## Multinomial logistic regression

A test offers a very quick and simple comparison of a singular categorical variable against another categorical variable. However, if we want to perform more complex analyses, say if we want to account for any covariates in the relationship or if we want to determine the effect of a quantitative variable on our categorical response, then tests are insufficient.

A logistic regression (logit) model is the most commonly used of the so-called binary regression models. A logistic model binds the output between 0 and 1 which makes it better suited to the probability of the presence of a characteristic compared to an unbounded linear model. Technically, a logistic model is considered proper when you want to model binary outcomes with covariates. However, a linear model offer “qualitatively the same” results with higher immediate interpretability and practical consistency between running and analyzing quantitative and categorical outcomes (Porter et al., 2024). Please refer to the [QMG memo](https://mdrc365.sharepoint.com/sites/QuantMethods/Shared%20Documents/Guidance/QMG-Memo-on-Binary-Outcomes%20(January%202024).pdf) for their testing and conclusions on the topic.

A multinomial logistic (mlogit) model generalizes the logit model to categorical response variables with arbitrarily many levels. For this example, we will be determining the effect of treatment on favAnimals while including cities, eduLevel and pre\_income\_raw as covariates, drawing from the mdrcAnalysis::sim\_data\_robust\_reg dataset.

# Loading dataset  
data(sim\_data\_robust\_reg, package = "mdrcAnalysis")  
# Making categoricals explicit and with desired reference levels for treatment  
sim\_data\_robust\_reg <- sim\_data\_robust\_reg %>%  
 mutate(  
 favAnimals = factor(favAnimals, levels = c("cat", "croc", "dog")),  
 treatment = factor(treatment, levels = c(0, 1)),  
 cities = factor(cities)  
 )  
sim\_data\_robust\_reg %>%  
 select(treatment, favAnimals, cities, eduLevel, pre\_income\_raw) %>%  
 head()

| treatment | favAnimals | cities | eduLevel | pre\_income\_raw |
| --- | --- | --- | --- | --- |
| 0 | dog | 1 | 2 | 7774.086 |
| 1 | cat | 1 | 5 | 29456.055 |
| 1 | cat | 1 | 3 | 21378.514 |
| 0 | croc | 1 | 2 | 7887.680 |
| 1 | cat | 1 | 3 | 31355.188 |
| 1 | dog | 1 | 1 | 11980.657 |

Here we will be using the nnet package to perform our analyses. This package is designed for neural network/machine learning applications and logistic regression functions as the backbone for many of these sorts of applications.

Running the model operates similarly to running a linear model with stats::lm. Here, I am scaling the quantitative predictor variable pre\_income\_raw for performance and convergence issues. It can also help with coefficient interpretability and comparison with other coefficients (Baguley, 2009).

library(nnet)  
mlogit\_model <- multinom(  
 favAnimals ~ cities + eduLevel + scale(pre\_income\_raw) + treatment,  
 data = sim\_data\_robust\_reg  
)

## # weights: 27 (16 variable)  
## initial value 354.851769   
## iter 10 value 347.147846  
## iter 20 value 346.622167  
## final value 346.622122   
## converged

Multinomial logistic fits a model via an iterative estimation process. If you want to hide the automatic printing of the optimization process, then you can set trace = FALSE:

mlogit\_model <- multinom(  
 favAnimals ~ cities + eduLevel + scale(pre\_income\_raw) + treatment,  
 data = sim\_data\_robust\_reg,  
 trace = FALSE  
)

We can view clean results by using the broom package. The p-value listed here is the result of a Wald z-test which is not actually calculated by nnet but rather by broom.

# Using broom and filtering for the treatment coefficient  
tidy(mlogit\_model) %>%  
 filter(term == "treatment1")

| y.level | term | estimate | std.error | statistic | p.value |
| --- | --- | --- | --- | --- | --- |
| croc | treatment1 | 0.1875968 | 0.2798265 | 0.670404 | 0.5026003 |
| dog | treatment1 | 0.4497142 | 0.2807859 | 1.601627 | 0.1092382 |

Logistic model coefficients can not be interpreted the same way you can interpret the linear model coefficients. Again, a significant reason why logistic models are avoided is because of the difficulty in interpreting results beyond direction and relative strength of effect. The coefficient of a logistic model describes “the estimated increase in the log[arithmic] odds of the outcome per unit increase in the value of the” predictor (Szumilas, 2010). The odds of an event are similar to but not equal to the probability of an event. Given the probability of an event , the odds are calculated as:

Because of how difficult it is to intuit the meaning of logistic model coefficients; they are frequently converted into “odds ratios”. This is done by exponentiating the coefficients, changing them from log odds into direct odds. broom makes this simple to do via the exponentiate argument.

tidy(mlogit\_model, exponentiate = TRUE) %>%  
 filter(term == "treatment1")

| y.level | term | estimate | std.error | statistic | p.value |
| --- | --- | --- | --- | --- | --- |
| croc | treatment1 | 1.206347 | 0.2798265 | 0.670404 | 0.5026003 |
| dog | treatment1 | 1.567864 | 0.2807859 | 1.601627 | 0.1092382 |

The estimate column here now lists the “odds ratio” instead of the direct coefficient. The odds ratio here are interpreted as:

* Given every other covariate in the model, being in the treatment group increases the odds of having “croc” as a favorite animal as opposed to the baseline “cat” by 1.21.
* Given every other covariate in the model, being in the treatment group increases the odds of having “dog” as a favorite animal as opposed to the baseline “cat” by 1.57.
* The p-value indicates the probability that, given all other coefficients are present, that the specified coefficient is equal to 0.

To find an p-value determining overall model fit, we will need to perform a commparison against a null model, a type of omnibus test. The way this conceptually works is that by comparing the fit quality between model with the treatment against a model without the treatment variable, we can determine if the improvement in fit quality is statistically significant. If our model is improved by a statistically significant degree, then treatment serves as an effective predictor of the categorical outcome overall. I will first show the more manual process of performing this calculation before following it up with a function that performs the work for us.

This manual process serves to better showcase how this sort of model comparison works and is applicable to a method described later in this paper. To perform this comparison, we need to create this “null” model by copying the original model code but removing treatment.

mlogit\_nullmodel <- multinom(  
 favAnimals ~ cities + eduLevel + scale(pre\_income\_raw),  
 data = sim\_data\_robust\_reg,  
 trace = FALSE  
)

We then pass both models into the anova function which, despite its name, will perform a model comparison test.

anova(mlogit\_model, mlogit\_nullmodel)

| Model | Resid. df | Resid. Dev | Test | Df | LR stat. | Pr(Chi) |
| --- | --- | --- | --- | --- | --- | --- |
| cities + eduLevel + scale(pre\_income\_raw) | 632 | 695.8591 |  | NA | NA | NA |
| cities + eduLevel + scale(pre\_income\_raw) + treatment | 630 | 693.2442 | 1 vs 2 | 2 | 2.614853 | 0.2705154 |

Here we have an overall p-value of 0.2 indicating that treatment is not a particularly effective predictor of favAnimals.

The car package provides the Anova function which performs this sort of null/effect model comparison automatically.

library(car, warn.conflicts = FALSE, quietly = TRUE)  
tidy(Anova(mlogit\_model))

| term | statistic | df | p.value |
| --- | --- | --- | --- |
| cities | 8.6880135 | 8 | 0.3692944 |
| eduLevel | 5.3485530 | 2 | 0.0689567 |
| scale(pre\_income\_raw) | 0.3130017 | 2 | 0.8551308 |
| treatment | 2.6148527 | 2 | 0.2705154 |

Anova presents the statistic and corresponding p-values for each coefficient in the model. We can once again use the broom package to filter for and tabulate the specific value we are interested in.

Anova(mlogit\_model) %>%  
 tidy() %>%  
 filter(term == "treatment")

| term | statistic | df | p.value |
| --- | --- | --- | --- |
| treatment | 2.614853 | 2 | 0.2705154 |

We see the exact same statistic and p-value using car::Anova as we did with our manual null model comparison.

### -Computation, SUR, and Adjusted Permutation

The [QMG memo on categorical methods](https://mdrc365.sharepoint.com/:b:/r/sites/QuantMethods/Shared%20Documents/Guidance/QMG%20Categorical%20Outcomes%20Memo%20(December%202013).pdf?csf=1&web=1&e=wqRkvO) recommends supplementing mlogit models with -computation to calculate p-values and estimates on p-values. This memo was written with a SAS workflow in mind and has not yet been updated with the equivalent R mdethods. Notably, this method uses a bootstrap estimate of standard error when calculating its Wald statistic which can dramatically increase the runtime of the calculation as the model specification and data increases in complexity. Until this method is written and tested in R, I would recommend consulting the DSB/QMG for assistance in setting this up.

The QMG has also tested a variety of other methods for dealing with categorical response variables. Their overall recommendation was to use either seemingly unrelated regression (SUR), mlogit with G-computation (which offers the most power), or adjusted permutation testing (Porter et al., 2013). These methods have not yet been rigorously implemented and tested in R by the QMG or DSB yet but I offer brief explanations and avenues for further exploration of their implementation.

Seemingly unrelated regression (SUR) which was driven in SAS by the SYSLIN procedure fits linear models with dummy response variables and then performs an F-test that that the treatment coefficients are zero. Here I use the systemfit package to perform this analysis. First we create a dummy variable off our categorical response variable.

sim\_data\_robust\_reg\_dummied <- sim\_data\_robust\_reg %>%  
 fastDummies::dummy\_cols("favAnimals", remove\_selected\_columns = TRUE)

We then create a list of linear model formulas and pass them to systemfit

suppressPackageStartupMessages(library("systemfit"))  
sur\_model <- c("favAnimals\_croc", "favAnimals\_dog") %>%  
 set\_names(c("croc", "dog")) %>%  
 map(function(.response\_variable) {  
 model\_formula <- as.formula(  
 paste(.response\_variable, "~ treatment + cities + eduLevel + scale(pre\_income\_raw)"),  
 env = rlang::env(!!!sim\_data\_robust\_reg\_dummied)  
 )  
 }) %>%  
 systemfit(method = "SUR")

Finally, we use car::linearHypothesis to perform an F-test on the coefficients on all treatment indicators is equal to 0.

# Setting up restriction "treatment coefficeint = 0"  
tidy(linearHypothesis(sur\_model, c("croc\_treatment1 = 0", "dog\_treatment1 = 0")))

| term | df.residual | df | statistic | p.value |
| --- | --- | --- | --- | --- |
| restricted model | 632 | NA | NA | NA |
| sur\_model | 630 | 2 | 1.290065 | 0.2759789 |

Adjusted permutation testing works by permuting the treatment indicator, estimating a custom, QMG developed test statistic for each permutation and then calculating a p-value based off of this custom test statistic. Packages like lmPerm simplify some aspects of permutation testing although it is likely to produce the permutation loop manually given the custom test statistic. This method is fairly computationally intensive given the 2000 recommended iterations and is also more difficult to explain compared to more commonly used tests but does provide flexibility around functional form.

The test statistic is defined:

The p-value is defined:

## Linear mixed-effects model

A mixed effect model (MEM), which for our purposes is identical to a hierarchical level model (HLM) and mixed level model (MLM), are used at MDRC to model data with a natural “nesting” structure, e.g. children within classrooms within schools or time periods nested in children for discrete time series analysis. A simple test is inappropriate for nested data because these observations are not independent. While mathematically possible and explored, we have not yet found an R package that works with multinomial logistic MEMs. Here we cover methods for working with categorical response variables without MNL but I do include some possible areas of further investigation at the end of the memo. The methods covered here do use dummied response variables but shows how to handle them in aggregate, thereby avoiding the previously listed issues with multiple dummy variable analysis.

Here we reference an omnibus test method that was proposed by Amy Taub and Marie-Andree Somers for the VIQI projects. Note that there exists a mixed extract in the mdrcAnalysis package but, I am not using it here for increased consistency between the code used to demonstrate quantitative and categorical response analysis.

For these examples, we will be using and modifying the merTools::hsb dataset which is a subset of the “1982 High School and Beyond survey” for its pre-existing nesting structure. We will be using the following variables in our example model:

* schid - School id
* mathach - Performance on standardized math assessment
* mathcat - A categorical variable indicating quantile in mathach
* schtype - Dummy variable indicating if the school is private
* minority - Dummy variable indicating if student is non-white or white
* ses - Student socio-economic status

data("hsb", package = "merTools")  
hsb\_cat <- hsb %>%  
 # Only keeping data with our desired outcome variable  
 filter(!is.na(mathach)) %>%  
 mutate(  
 # Creating a multinomial categorical variable  
 # by breaking up into 25% quantiles  
 mathcat = factor(cut(mathach, 4,  
 labels = c("First", "Second", "Third", "Fourth")  
 )),  
 # Making other categoricals explicit factors  
 across(c(schid, schtype), as.factor)  
 )  
glimpse(hsb\_cat)

## Rows: 7,185  
## Columns: 9  
## $ schid <fct> 1224, 1224, 1224, 1224, 1224, 1224, 1224, 1224, 1224, 1224, 1…  
## $ minority <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0…  
## $ female <dbl> 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1…  
## $ ses <dbl> -1.528, -0.588, -0.528, -0.668, -0.158, 0.022, -0.618, -0.998…  
## $ mathach <dbl> 5.876, 19.708, 20.349, 8.781, 17.898, 4.583, -2.832, 0.523, 1…  
## $ size <dbl> 842, 842, 842, 842, 842, 842, 842, 842, 842, 842, 842, 842, 8…  
## $ schtype <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ meanses <dbl> -0.428, -0.428, -0.428, -0.428, -0.428, -0.428, -0.428, -0.42…  
## $ mathcat <fct> Second, Fourth, Fourth, Second, Third, Second, First, First, …

If we want to calculate the effect of minority on mathach, a quantitative variable using ses as a covariate and schid as a random intercept effect, we can simplify specify the model as such using the lmerTest package. I am also calling the emmeans package for adjusted means calculations and broom.mixed to extend the tidying methods of broom onto the output of lme4

# Calling necessary packages  
library(lmerTest, warn.conflicts = FALSE, quietly = TRUE)  
library(emmeans, warn.conflicts = FALSE, quietly = TRUE)

## Welcome to emmeans.  
## Caution: You lose important information if you filter this package's results.  
## See '? untidy'

# Setting a higher limit given the amount of rows in our dataset  
emm\_options(pbkrtest.limit = 8000)  
library(broom.mixed, warn.conflicts = FALSE, quietly = TRUE)

# Specifying the model  
lmer(mathach ~ minority + ses + (1 | schid), data = hsb\_cat) %>%  
 # Getting adjusted means by treatment but using satterthwaite degres of freedom  
 emmeans("minority", options = get\_emm\_option("satterthwaite")) %>%  
 # Showing schtype contrasts  
 pairs()

## contrast estimate SE df t.ratio p.value  
## minority0 - minority1 2.94 0.207 5101 14.172 <.0001  
##   
## Degrees-of-freedom method: kenward-roger

Suppose then we wanted to identify the relationship between mathcat and minority while still accounting for the school level nesting. The following method will work because we do not have any school-level predictors. If you do have predictor variables that are only vary across your nesting variable, then you

This method separates the calculation of contrasts and the overall p-value. The contrasts are calculated using dummy response variables as a proxy for the categorical response variable, treating these dummies as quantitative in the same way we dealt with the quantitative response variable above. The p-value is calculated using a similar null/effect model omnibus comparison as we performed for the mlogit section.

First to calculate the contrasts, we create dummy variables of the original mathcat variable.

hsb\_dummied <- hsb\_cat %>%  
 # Using fastDummies to simplify construction  
 fastDummies::dummy\_cols("mathcat", remove\_selected\_columns = TRUE)  
glimpse(hsb\_dummied)

## Rows: 7,185  
## Columns: 12  
## $ schid <fct> 1224, 1224, 1224, 1224, 1224, 1224, 1224, 1224, 1224, 1…  
## $ minority <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0…  
## $ female <dbl> 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1…  
## $ ses <dbl> -1.528, -0.588, -0.528, -0.668, -0.158, 0.022, -0.618, …  
## $ mathach <dbl> 5.876, 19.708, 20.349, 8.781, 17.898, 4.583, -2.832, 0.…  
## $ size <dbl> 842, 842, 842, 842, 842, 842, 842, 842, 842, 842, 842, …  
## $ schtype <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…  
## $ meanses <dbl> -0.428, -0.428, -0.428, -0.428, -0.428, -0.428, -0.428,…  
## $ mathcat\_First <int> 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1…  
## $ mathcat\_Second <int> 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0…  
## $ mathcat\_Third <int> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0…  
## $ mathcat\_Fourth <int> 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0…

Next, we create the model and find the contrasts for each of our new outcome variables. Here I show the code for doing this on the “cat” outcome variable.

# Specifying the model  
lmer(mathcat\_First ~ minority + ses + (1 | schid), data = hsb\_dummied) %>%  
 # Getting adjusted means by treatment but using satterthwaite degres of freedom  
 emmeans("minority", options = get\_emm\_option("satterthwaite")) %>%  
 # Showing schtype contrasts  
 pairs()

## contrast estimate SE df t.ratio p.value  
## minority0 - minority1 -0.102 0.0105 3095 -9.747 <.0001  
##   
## Degrees-of-freedom method: kenward-roger

Doing this for every level of a categorical variable can quickly become quite tedious so for the rest of the dummy variables, I automate the creation of the formulas and model calculations.

# Collecting all dummy variables  
dummy\_names <- names(select(hsb\_dummied, matches("mathcat")))  
# Iterating on each dummy variable  
mixed\_contrasts <- purrr::map(dummy\_names, function(.response\_variable) {  
 dummy\_formula <- as.formula(paste(.response\_variable, "~ minority + ses + (1|schid)"),  
 env = rlang::env(!!!hsb\_dummied)  
 )  
 # Running model and contrasts using our constructed formula  
 lmer(dummy\_formula) %>%  
 emmeans("minority", options = get\_emm\_option("satterthwaite")) %>%  
 pairs() %>%  
 # Creating into a tabular data.frame  
 tidy() %>%  
 # Encoding categorical level  
 mutate(  
 Categorical\_level = .response\_variable,  
 minority\_contrast = estimate,  
 .keep = "none", .before = 1  
 )  
}) %>%  
 bind\_rows()  
mixed\_contrasts

| Categorical\_level | minority\_contrast |
| --- | --- |
| mathcat\_First | -0.10189653 |
| mathcat\_Second | -0.07731576 |
| mathcat\_Third | 0.03281047 |
| mathcat\_Fourth | 0.14212793 |

Now to perform the omnibus test so we can see the statistical significance of these contrasts. Here we effectively reverse the model so that schtype is the response variable and combined\_cat is a predictor variable. Doing this allows us to run minority as a quantitative response dummy variable although first we need to make sure that minority is coded as a numeric {0,1} variable.

unique(hsb\_cat$minority)

## [1] 0 1

# Confirmed that it is a {0,1} numeric dummy

Now we calculate two models, a null and effect model. Both models will use our new numeric schtype variable as a response variable and account for the nesting within schid but only the effect model contains our categorical variable combined\_cat as a predictor. Then we will compare the two models with an ANOVA test. Because the only difference in the two models is the inclusion of the categorical variable as a predictor, the resulting p-value will tell us if the inclusion of the categorical variable has a statistically significant effect on the model’s ability to fit for schtype.

mixed\_null\_model <- lmer(minority ~ ses + (1 | schid), data = hsb\_cat)  
mixed\_effect\_model <- lmer(minority ~ mathcat + ses + (1 | schid), data = hsb\_cat)  
mixed\_anova <- anova(mixed\_null\_model, mixed\_effect\_model)

## refitting model(s) with ML (instead of REML)

tidy(mixed\_anova)

| term | npar | AIC | BIC | logLik | deviance | statistic | df | p.value |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| mixed\_null\_model | 4 | 4840.185 | 4867.704 | -2416.093 | 4832.185 | NA | NA | NA |
| mixed\_effect\_model | 7 | 4670.473 | 4718.631 | -2328.236 | 4656.473 | 175.7125 | 3 | 7.435548e-38 |

Here we see the relevant p-value results from performing an ANOVA test on the effect and null model. We can now combine the p-value here with the contrasts we calculated previously to create a single table.

mixed\_contrasts %>%  
 # Adding in the non-missing p-value  
 mutate(P\_Value = na.omit(mixed\_anova$`Pr(>Chisq)`))

| Categorical\_level | minority\_contrast | P\_Value |
| --- | --- | --- |
| mathcat\_First | -0.10189653 | 7.435548e-38 |
| mathcat\_Second | -0.07731576 | 7.435548e-38 |
| mathcat\_Third | 0.03281047 | 7.435548e-38 |
| mathcat\_Fourth | 0.14212793 | 7.435548e-38 |

## Linear mixed-effects model with random intercept level covariates

The method above can run into a practical issue with many model specifications because of how it creates a null and effect model by using the contrasting variable as the response variable. In our example, the variable we were calculating contrasts on was minority which was defined on a child level. Suppose we instead wanted to use schtype which is defined at the schid level? Here I recreate the null and effect model specification using schtype in place of minority

# Making schtype into a {0,1} dummy variable  
hsb\_cat <- hsb\_cat %>%  
 mutate(schtype = as.integer(levels(schtype)[schtype]))  
# Calculating null model  
lmer(schtype ~ ses + (1 | schid), data = hsb\_cat)

## Warning in optwrap(optimizer, devfun, getStart(start, rho$pp), lower =  
## rho$lower, : convergence code -4 from nloptwrap: NLOPT\_ROUNDOFF\_LIMITED:  
## Roundoff errors led to a breakdown of the optimization algorithm. In this case,  
## the returned minimum may still be useful. (e.g. this error occurs in NEWUOA if  
## one tries to achieve a tolerance too close to machine precision.)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :  
## unable to evaluate scaled gradient

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :  
## Model failed to converge: degenerate Hessian with 1 negative eigenvalues

## Linear mixed model fit by REML ['lmerModLmerTest']  
## Formula: schtype ~ ses + (1 | schid)  
## Data: hsb\_cat  
## REML criterion at convergence: -161116.2  
## Random effects:  
## Groups Name Std.Dev.   
## schid (Intercept) 7.404e-02  
## Residual 2.486e-06  
## Number of obs: 7185, groups: schid, 160  
## Fixed Effects:  
## (Intercept) ses   
## 4.376e-01 4.185e-12   
## optimizer (nloptwrap) convergence code: -4 (NLOPT\_ROUNDOFF\_LIMITED: Roundoff errors led to a breakdown of the optimization algorithm. In this case, the returned minimum may still be useful. (e.g. this error occurs in NEWUOA if one tries to achieve a tolerance too close to machine precision.)) ; 0 optimizer warnings; 2 lme4 warnings

In our case here, the model technically runs but with a host of warnings stating that the model failed to converge. In these cases, the solution is to aggregate the data up to the definition level of our new predictor. In this case, because schtype is a schid level variable, we aggregate by grouping at schid, replacing quantitative variables with their mean. In the case of categorical variables, we must separate them out into dummy variables and then take the means of the dummy variables, effectively making these into proportional measures.

hsb\_aggregated <- hsb\_cat %>%  
 fastDummies::dummy\_cols("mathcat") %>%  
 group\_by(schid, schtype) %>%  
 summarise(across(where(is.numeric), mean), .groups = "drop")

We now calculate the p-value using our aggregated dataset. However, because the data is now at the schid level, we actually run this as a linear model instead of a mixed effects model. If there was another grouping level in our original model specification, we would instead run a mixed effects model at that higher level. In other words, because this was originally a 2-level model, students nested in schools, and we aggregated the data to the school level, it would now be run as a 1-level model, i.e. a linear model. If this was a 3-level model, say students nested in classrooms nested in schools and we only aggregated to the classroom level, then we would run this as a 2-level model with classrooms at the individual level and centers at the grouping level.

Besides this, we still construct a null and effect model by putting our contrast variable as the response. We can then pass the two models into an ANOVA omnibus test to find an aggregated p-value.

aggregated\_null\_model <- lm(schtype ~ ses, data = hsb\_aggregated)  
aggregated\_effect\_model <- lm(schtype ~ mathcat\_First + mathcat\_Second + mathcat\_Third + mathcat\_Fourth  
 + ses, data = hsb\_aggregated)  
aggregated\_anova <- anova(aggregated\_null\_model, aggregated\_effect\_model)  
tidy(mixed\_anova)

| term | npar | AIC | BIC | logLik | deviance | statistic | df | p.value |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| mixed\_null\_model | 4 | 4840.185 | 4867.704 | -2416.093 | 4832.185 | NA | NA | NA |
| mixed\_effect\_model | 7 | 4670.473 | 4718.631 | -2328.236 | 4656.473 | 175.7125 | 3 | 7.435548e-38 |

Calculation of the contrasts does not require the use of the aggregated model because the response variable for our contrast calculation uses the student level variable mathcat. For this example, the only thing we are changing with our original mixed-effects model example is the use of schtype instead of minority Here I am skipping ahead directly to calculating all contrasts at once.

# Iterating on each dummy variable  
mixed\_contrasts <- purrr::map(dummy\_names, function(.response\_variable) {  
 dummy\_formula <- as.formula(paste(.response\_variable, "~ schtype + ses + (1|schid)"),  
 env = rlang::env(!!!hsb\_dummied)  
 )  
 # Running model and contrasts using our constructed formula  
 lmer(dummy\_formula) %>%  
 emmeans("schtype", options = get\_emm\_option("satterthwaite")) %>%  
 pairs() %>%  
 # Creating into a tabular data.frame  
 tidy() %>%  
 # Encoding categorical level  
 mutate(  
 Categorical\_level = .response\_variable,  
 minority\_contrast = estimate,  
 .keep = "none", .before = 1  
 )  
}) %>%  
 bind\_rows()  
mixed\_contrasts

| Categorical\_level | minority\_contrast |
| --- | --- |
| mathcat\_First | 0.08280305 |
| mathcat\_Second | 0.04588514 |
| mathcat\_Third | -0.05733052 |
| mathcat\_Fourth | -0.07153394 |

We can now combine the p-values from the omnibus test and these contrasts to get a single table of results.

mixed\_contrasts %>%  
 # Adding in the non-missing p-value  
 mutate(P\_Value = na.omit(aggregated\_anova$`Pr(>F)`))

| Categorical\_level | minority\_contrast | P\_Value |
| --- | --- | --- |
| mathcat\_First | 0.08280305 | 0.0001297741 |
| mathcat\_Second | 0.04588514 | 0.0001297741 |
| mathcat\_Third | -0.05733052 | 0.0001297741 |
| mathcat\_Fourth | -0.07153394 | 0.0001297741 |

### Alternative methods for mixed-effect models

Note that by setting family = binomial, there is already a treatment for binary outcome variables using lme4::glmer. One method I saw in the [lme4 github](https://github.com/lme4/lme4/issues/594) would be to convert a multinomial model into the equivalent Poisson regression (Lee et al., 2017). lme4::glmer supports every GLM family listed in stats::glm which does includes Poisson regression which means we can use this equivalent Poisson formulation in lme4::glmer. However, this is easier said than done and there are also concerns about the computational complexity of such a technique.

A Bayesian mixed effects model should also theoretically handle this situation using the brms package. This option has seen some traction online, but it may be difficult to practically implement because it, like many software libraries designed around probabilistic programming and Bayesian modelling, uses the “Stan” programming language in its backend which basically required another compilation step on program run.

# References

*Advanced R, second edition*. (2019). Chapman and Hall/CRC. <https://doi.org/10.1201/9781351201315>

Baguley, T. (2009). Standardized or simple effect size: What should be reported? *British Journal of Psychology (London, England: 1953)*, *100*(Pt 3), 603–617. <https://doi.org/10.1348/000712608X377117>It is regarded as best practice for psychologists to report effect size when disseminating quantitative research findings. Reporting of effect size in the psychological literature is patchy - though this may be changing - and when reported it is far from clear that appropriate effect size statistics are employed. This paper considers the practice of reporting point estimates of standardized effect size and explores factors such as reliability, range restriction and differences in design that distort standardized effect size unless suitable corrections are employed. For most purposes simple (unstandardized) effect size is more robust and versatile than standardized effect size. Guidelines for deciding what effect size metric to use and how to report it are outlined. Foremost among these are: (i) a preference for simple effect size over standardized effect size, and (ii) the use of confidence intervals to indicate a plausible range of values the effect might take. Deciding on the appropriate effect size statistic to report always requires careful thought and should be influenced by the goals of the researcher, the context of the research and the potential needs of readers.

Lee, J. Y. L., Green, P. J., & Ryan, L. M. (2017). *On the "poisson trick" and its extensions for fitting multinomial regression models*. <https://arxiv.org/abs/1707.08538>

Porter, K., Michalopoulos, C., & Greeney, A. (2013). *QMG Categorical Outcomes Memo*.

Porter, K., Weiss, M. J., & Cullinan, D. (2024). *Linear Probability Models Vs. Logistic Regression*.

Sinharay, S. (2010). An overview of statistics in education. In P. Peterson, E. Baker, & B. McGaw (Eds.), *International encyclopedia of education (third edition)* (Third Edition, pp. 1–11). Elsevier. <https://doi.org/10.1016/B978-0-08-044894-7.01719-X>There have been numerous applications of statistical methods to the field of education, mostly in educational measurement. This article provides short descriptions of several topics in statistics that have found applications to education and provides examples of applications of some of these topics to education.

Stefan, A. M., & Schönbrodt, F. D. (2023). Big little lies: A compendium and simulation of p-hacking strategies. *Royal Society Open Science*, *10*(2). <https://doi.org/10.1098/rsos.220346>In many research fields, the widespread use of questionable research practices has jeopardized the credibility of scientific results. One of the most prominent questionable research practices is p-hacking. Typically, p-hacking is defined as a compound of strategies targeted at rendering non-significant hypothesis testing results significant. However, a comprehensive overview of these p-hacking strategies is missing, and current meta-scientific research often ignores the heterogeneity of strategies. Here, we compile a list of 12 p-hacking strategies based on an extensive literature review, identify factors that control their level of severity, and demonstrate their impact on false-positive rates using simulation studies. We also use our simulation results to evaluate several approaches that have been proposed to mitigate the influence of questionable research practices. Our results show that investigating p-hacking at the level of strategies can provide a better understanding of the process of p-hacking, as well as a broader basis for developing effective countermeasures. By making our analyses available through a Shiny app and R package, we facilitate future meta-scientific research aimed at investigating the ramifications of p-hacking across multiple strategies, and we hope to start a broader discussion about different manifestations of p-hacking in practice.

Szumilas, M. (2010). [Explaining Odds Ratios](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2938757). *Journal of the Canadian Academy of Child and Adolescent Psychiatry*, *19*(3), 227–229.