Simulations of the Minimax Optimality of Permutation Tests

Michelle Vered

2025-04-25

1 Introduction

This project explores the use of permutation tests in the context of two-sample and independence testing. As described in the core paper which is the basis for this project (Kim et al. 2022), permutation tests can provide finite sample guarantees on the Type I error rate and can be minimax optimal in a variety of circumstances. In particular, we focus on the use of permutation test based U-statistics as applied to a two-sample comparison of medians as well as in independence testing in high dimensional multinomial problems.

We utilize a data set on housing prices as the basis for various simulation studies to explore minimax optimality of permutation tests. We divide this data set into two groups and use these groups to represent the true population distributions of some random variable.

In one set of simulations, we sample from these two distinct populations. We then compare the frequency of Type II error rates (falsely failing to reject the null hypothesis) between a permutation test and a non-permutation test based approach. In a second set of simulations, we draw two samples both from the same underlying population group. We then compare the prevalence of Type I errors (falsely rejecting the null hypothesis) between permutation and non-permutation test based approaches.

2 Methods

2.1 Permutation Tests

In hypothesis testing, we calculate a test statistic based on the sample of data. Typically, we then assume that the under the null hypothesis, that test statistic should follow some specific distribution, such as a Student's t-distribution or a Chi-squared distribution. We choose a critical value as some point from that distribution, such as the value of the test statistic that corresponds to less than a 5% probability of observing a test statistic as extreme or more extreme than the critical value if the null hypothesis were true. We make a decision about whether to reject or fail to reject the null hypothesis by comparing our observed test statistic to that critical value.

However, the standard repertoire of hypothesis tests do not work in all situations. When dealing with small sample sizes, or when distributional assumptions required for parametric tests are not met (such as normality), one alternative we can turn to would be a permutation test. Permutation tests are a flexible alternative that allow us to create our own null distribution with which to compare our test statistic rather than assuming a specific distribution.

The basic idea of permutation testing is to randomly re-shuffle the labels of observations, then re-calculate the test statistic. This process is repeated many times to create a distribution for the test statistic under the null (Albert 2015). If we are comparing two samples, for example, if these two samples were drawn from the same underlying population distribution (i.e. the null was true), we would expect that the labels

(sample 1 or sample 2) would not matter, so we want to calculate a permutation distribution for the test statistic under that assumption. We then select a critical value for our decision rule based on a quantile of the permutation distribution.

Formally, assume we have observations $X = X_1, ..., X_n$ and $Y = Y_1, ..., Y_m$ and define a test statistic $T(X, Y)$ which measures the difference between the two samples. For each permutation, select n observations from the pooled data $X_1, ..., X_n, Y_1, ..., Y_m$ and label these

$$X^*_{(1)} = X^*_{(1)_1}, ..., X^*_{(1)_n}$$

and the remainder of observations as

$$Y^*_{(1)} = Y^*_{(1)_1}, ..., Y^*_{(1)_n}$$

then calculate a permuted test statistic $T^*_{(1)}(X^*, Y^*)$. Repeat this process n times to get

$$T^*_{(1)}, ..., T^*_{(n)}$$

(or repeat many times for a large sample where calculating every possible permutation of the data is computationally unfeasible). Choose a critical value c^* as a quantile from the distribution of $T^*_{(1)}, ..., T^*_{(n)}$.

It is worth noting that bootstrapping is a related, but slightly different, re-sampling procedure for creating a distribution of the test statistic under the null. As compared with permutation testing, in a bootstrap approach, we re-sample from the original data, with replacement, to generate each new sample rather than permuting the labels of the original data (Janssen and Pauls 2003). However, bootstrapping similarly creates a distribution for the test statistic for which we can find a particular quantile to serve as the critical value.

2.2 Optimality

We explore several different approaches to hypothesis testing that allow us to illustrate the performance of the permutation-based approach to U-statistics. We focus on two applications in particular: testing whether two numeric samples come from the same underlying population with the same median (two-sample testing) and testing whether there is an association between two categorical variables (independence testing). In two sample testing, our null hypothesis is that the two samples come from the same underlying distribution with the same median. In independence testing, our null hypothesis is that there is no association between the categorical variables.

To assess the performance of different approaches to hypothesis testing, we are interested in measuring the Type I and Type II error rates produced by our estimators. Type I error rates occur when the null hypothesis is in fact true, but an estimator leads us to falsely reject the null. Type II error rates occur when the null hypothesis is in fact false, but an estimator fails to reject the null. The Type II error rates may occur when the test is underpowered and fails to detect an effect.

In practice, understanding the power of statistical test can be very important, as statisticians are often called upon to help design studies and determine optimal sample sizes for experiments or models to reach sufficient statistical power. Overestimating the sample size needed can lead to unneeded costs in the study implementation/data collection phase, but underestimate the required sample size can result in wasted effort running a study that produces only inconclusive results.

Previous work has explored the minimax optimality of permutation tests in large samples (Robinson 1973) and for simple test statistics such as the t-statistic and F-statistic (Janssen 2005). A primary contribution of the key paper for this project (Kim et. al. 2022) is to provide a non-asymptotic framework to analyze the minimax Type II error of the permutation test. Their framework is particularly useful for small sample sizes. In one set of Monte Carlo simulations, they focus on simulations with $n_1 = n_2 = 100$.

We can define a minimax optimal estimator as one the minimizes the maximum risk. Formally, the maximum risk of an estimator $\widehat{\theta}$ can be expressed as

$$\bar{R}(\widehat{\theta}) = \sup_{\theta} R(\theta, \widehat{\theta})$$

and a minimax optimal estimator will be one such that

$$\sup_{\theta} R(\theta, \widehat{\theta}) = \inf_{\tilde{\theta}} \sup_{\theta} R(\theta, \tilde{\theta})$$

where the infimum is over all estimators $\tilde{\theta}$ (Wasserman 2010).

In practice, a hypothesis testing approach would be minimax optimal in terms of the Type II error rate when the decision rule it produces minimizes the likelihood of failing to reject the null hypothesis when the null is in fact false, as compared to all other decision rules. This "worst case" situation would be most likely to occur when there is a very small difference between the expected distributions under the null and alternate hypotheses.

As an added step in this project, we also run simulations to measure the Type I error rate. Finite sample guarantees on the Type I error rate are much more well-established, as long as the exchangeability assumption is met. These results are covered in older elements of the literature on permutation tests (Hoeffding 1952).

In our simulations, we will use the King County, Washington housing price data as if it were the underlying true population distribution, so simulations will not necessarily be measuring the Type I and Type II error rates under the worst case, closest possible degree of similarity between the null and alternate hypotheses. We also won't be comparing permutation test based approaches with every other possible hypothesis test. But we will be able to measure the Type II minimax optimality of permutation tests in the sense of looking across many different samples from the population and looking at the frequency of Type II errors across the "unluckiest" samples with the worst possible configuration of observations selected to be in the sample.

2.3 U-Statistics

Here we specifically focus on the use of permutation-based approaches with a class of statistics known as U-statistics. We can define U_n as a U-statistic with kernel h of degree m as

$$U_n = \binom{n}{m}^{-1} \sum_{(i_i,...,i_m) \in I_{n,m}} h(X_{i_1}, ..., X_{i_m})$$

where h is a symmetric function over m arguments and we are averaging the outcomes $h(X_{i_1}, ..., X_{i_m})$ across all possible ordered tuples $I_{n,m} = \{(i_1, ..., i_m) : 1 \leq i_1 < ... < i_m \leq n\}$ (Chen 2005, Wasserman 2010). Many estimators can be written to fit the format of this class of statistics, such as the sample variance.

2.4 Two Sample Testing

In two-sample testing, we consider two samples of numeric data. Under the null hypothesis, we assume both samples come from the same underlying population distribution.

There are many options for performing this type of test. t-tests are a traditional, parametric approach but assume the underlying data is normally distributed. The Mann-Whitney U-test, also called the Wilcoxon rank-sum test, is a non-parametric alternative. The Mann-Whitney U-test is a test of both distribution shape and location (the central tendency/median). If we are able to assume that one sample is a shifted version of the distribution of the other sample, then the Mann-Whitney U-test is a test that directly allows us to test for a difference in medians (Hart 2001). The Mann-Whitney U-statistic is calculated by first defining a rank function that orders the data in both samples

$$r(x_i) = \text{rank of } x_i \text{ in } x_1, x_2, ..., x_{n_X}, y_1, y_2, ..., y_{n_Y}$$

Then the test statistic is the smaller of

$$U_1 = n_X n_Y + \frac{n_X(n_X + 1)}{2} - \sum_{i=1}^{n_X} r(x_i)$$

and

$$U_2 = n_X n_Y + \frac{n_Y(n_Y+1)}{2} - \sum_{j=1}^{n_Y} r(y_j)$$

While the Mann-Whitney U-test is non-parametric in that it does not assume a normal distribution or any other specific distribution for the underlying data, in the standard test we do assume that the distribution of the U-statistic under the null can be approximated by the normal distribution. Critical values are then taken that correspond to probabilities of the normal distribution. This approximation tends to work with sufficiently large sample sizes.

For a permutation test based Mann-Whitney U-statistic, the U-statistic itself would be calculated in exactly the same way, but the reference distribution for finding the critical value would be created through permutation.

2.5 Independence Testing

In independence testing, we consider two categorical variables. Under the null hypothesis, we assume there is no association between these variables. Here we specifically look to a binary categorical variable and a second categorical variable that has a large number of categories.

A standard approach to multinomial independence testing is to use the Chi-squared test. However, this approach has a number of drawbacks. Kim et al. (2022) highlight the particular utility of permutation testing in high-dimensional multinomial settings, when the the number of categories is large and may even be larger than the number of observations in the sample. The drawbacks of traditional tests for these high dimensional multinomial settings was investigated in Balakrishnan and Wasserman (2018), who suggest permutation tests as a potential solution to explore.

Barrett et. al. (2020) separately also consider the optimality of U-statistic based permutation tests for independence testing and arrive at similar results for minimax optimality as Kim et. al. (2022) but by taking a different mathematical approach using different assumptions on smoothness of permutation distributions.

Kim et al. (2022) propose a new U-statistic for independence testing in high dimensional multinomial settings. Let p_y and p_z be multinomial distributions on a discrete domain $\mathbb{S}_d$ with kernel

$$h_{ts}(y_1, y_2; z_1, z_2) = g(y_1, y_2) + g(z_1, z_2) - g(y_1, z_2) - g(y_2, z_1)$$

defined by the following bivariate function

$$g_{\text{Multi}}(x, y) = \sum_{k=1}^{d} \mathbb{I}(x = k)\mathbb{I}(y = k)$$

and the corresponding U-statistic

$$U_{n_1,n_2} = \frac{1}{(n_1)_{(2)}(n_2)_{(2)}} \sum_{(i_1,i_2)\in i_2^{n_1}} \sum_{(j_1,j_2)\in i_2^{n_2}} h_{ts}(Y_{i_1}, Y_{i_2}; Z_{j_1}, Z_{j_2})$$

A unique contribution of this project is implementing calculation of this test statistic into R.

3 Data Exploration

3.1 Overview

We now turn to an exploration of the data set used for running the simulations in this project.

The data set we have selected for this project provides information on the sale price and other characteristics of houses which were sold in King County, Washington (the Seattle area). The data set includes 21613 observations on homes sold between 2014 - 2015. Each observation in the data set represents one sale of a property.

After importing, we applied some basic data cleaning by re-classifying the data types for each of the columns when necessary and converted prices to be in thousands of dollars. We also engineered two Boolean variables to convert column that records the square footage of a basement into a binary classification, and did the same for a column indicating the year a house was renovated. Overall summary statistics for this data set are presented below:

Table 1: Data summary

Name	data
Number of rows	21613
Number of columns	23
Column type frequency:	
character	1
Date	1
factor	4
logical	3
numeric	14
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
id	0	1	7	10	0	21436	0

Variable type: Date

skim_variable	n_missing	complete_rate	min	max	median	n_unique
date	0	1	2014-05-02	2015-05-27	2014-10-16	372

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
view	0	1	TRUE	5	0: 19489, 2: 963, 3: 510, 1: 332
condition	0	1	TRUE	5	3: 14031, 4: 5679, 5: 1701, 2: 172
grade	0	1	TRUE	12	7: 8981, 8: 6068, 9: 2615, 6: 2038
zipcode	0	1	FALSE	70	981: 602, 980: 590, 981: 583, 980: 574

Variable type: logical

skim_variable	n_missing	complete_rate	mean	count
waterfront	0	1	0.01	FAL: 21450, TRU: 163

skim_variable	n_missing	complete_rate	mean	count
has_basement	0	1	0.39	FAL: 13126, TRU: 8487
was_renovated	0	1	0.04	FAL: 20699, TRU: 914

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
price	0	1	540	367.1	75	322	450	645	7700
bedrooms	0	1	3	0.9	0	3	3	4	33
bathrooms	0	1	2	0.8	0	2	2	2	8
sqft_living	0	1	2080	918.4	290	1427	1910	2550	13540
sqft_lot	0	1	15107	41420.5	520	5040	7618	10688	1651359
floors	0	1	1	0.5	1	1	2	2	4
sqft_above	0	1	1788	828.1	290	1190	1560	2210	9410
sqft_basement	0	1	292	442.6	0	0	0	560	4820
yr_built	0	1	1971	29.4	1900	1951	1975	1997	2015
yr_renovated	0	1	84	401.7	0	0	0	0	2015
lat	0	1	48	0.1	47	47	48	48	48
long	0	1	-122	0.1	-123	-122	-122	-122	-121
sqft_living15	0	1	1987	685.4	399	1490	1840	2360	6210
sqft_lot15	0	1	12768	27304.2	651	5100	7620	10083	871200

These summary statistics show a relatively clean data set already, with no missing values in any of the columns.

We will focus on only a subset of the columns available in this data set. Of particular interest are the columns:

- *has_basement*: Boolean variable indicating if the house has a basement or not
- *price*: numeric variable indicating the sale price of the home, in thousands of dollars
- *zipcode*: categorical variable

Since this paper focuses on the application of permutation-test based U-statistics in the context of two sample testing and independence testing, we are interested in dividing the data set into two samples. We can do that using the *has_basement* column. While *waterfront* is also a Boolean variable available in the data set, the very small overall number of waterfront properties makes this column less suitable for comparison of hypothesis testing approaches. We also considered splitting the data based on the was *was_renovated* column, but the *has_basement* column seemed to better fit the assumption of a shifted distribution required for the Mann-Whitney U-test, so we decided to focus there for the purpose of comparing approaches to two-sample and independence testing.

Diving the data set based on homes with and without a basement, we have:

- **Population 1:** 8,487 properties with basements (39.3% of the data set)
- **Population 2:** 13,126 properties with basements (60.7% of the data set)

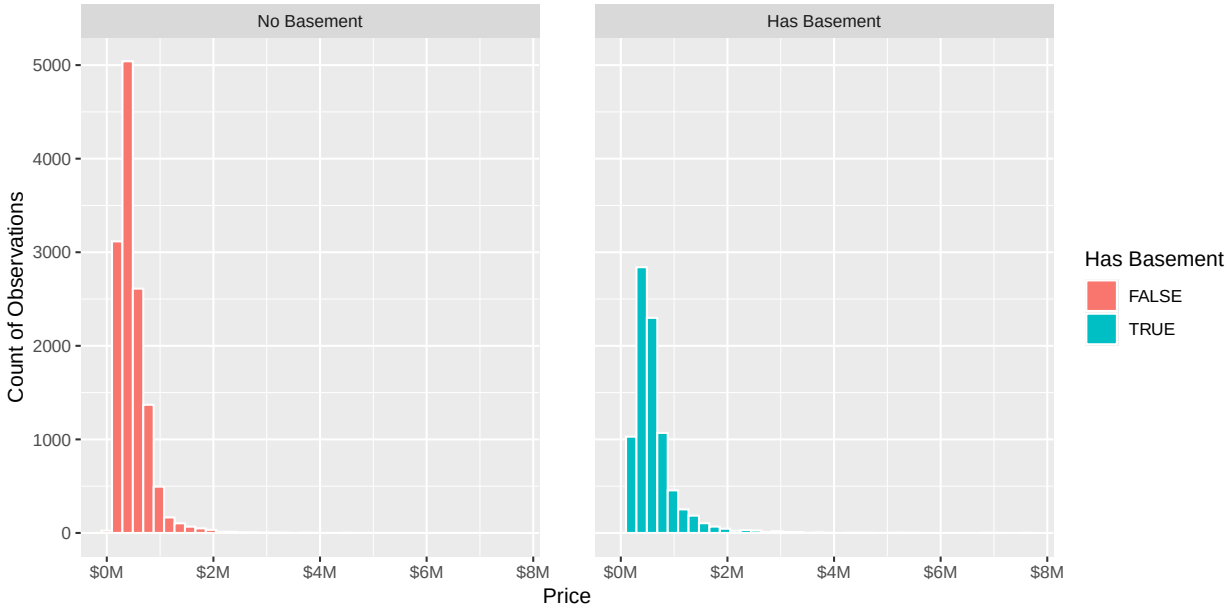
3.2 Price

We can compare quantiles for the distribution of prices between these two types of properties.

has_basement	Price Q25	Median Price	Price Q75
FALSE	299	412	595
TRUE	375	515	712

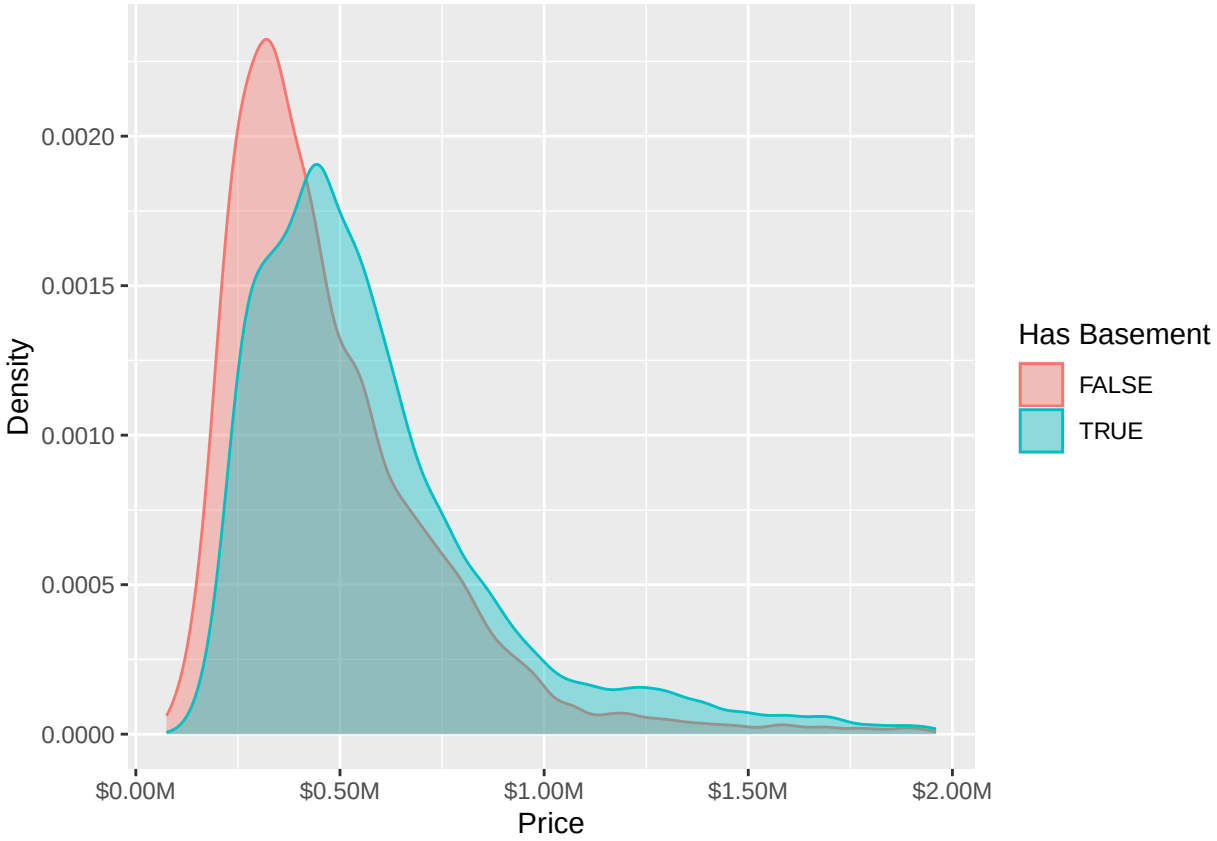
It seems clear from examining the quantiles of each population's distribution that homes with and without basements represent two distinct populations with different distributions of the price variable.

We can also visualize the whole distribution of prices for these two groups of houses.



There a few outliers - extremely expensive homes - which shift the histograms and make it difficult to see prices clearly for houses under $2 million. It is also harder to directly compare the two distributions since there are differing numbers of observations in the two groups.

To hone in on a comparison of price distributions, we can generate a new overlapping density plot and exclude outliers (the largest 1% of home prices).

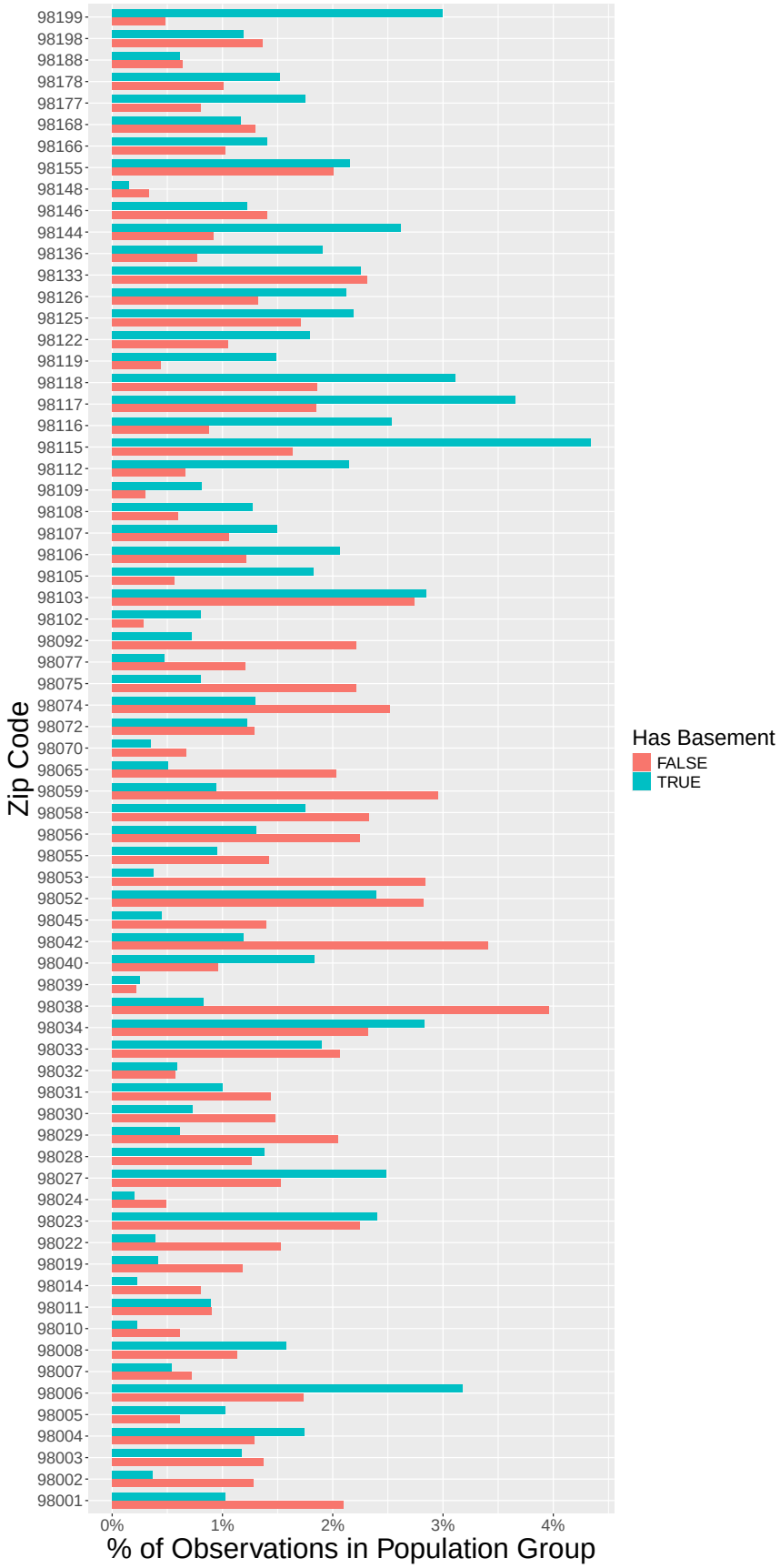


From this density plot, and the quantiles for each distribution provided above, it appears as though the distribution of prices for homes with a basement may be a shifted version of the distribution of prices for homes without a basement. This is an ideal set-up for the use of a Mann-Whitney U-test to compare medians, as discussed in more detail in the methods section.

3.3 Zip Codes

We can also compare the distribution of zip codes between the two populations. Counts of homes per zip code can be thought of as draws from a multinomial distribution.

To better understand how the distributions of zip codes might compare in the two populations, homes with and without basements, we can examine a bar chart. The bar chart below visualizes the percentage of homes in each population that are located in a particular zip code.


Zip Code
98199
98198
98188
98178
98177
98168
98166
98155
98148
98146
98144
98136
98133
98126
98125
98122
98119
98118
98117
98116
98115
98112
98109
98108
98107
98106
98105
98103
98102
98092
98077
98075
98074
98072
98070
98065
98059
98058
98056
98055
98053
98052
98045
98042
98040
98039
98038
98034
98033
98032
98031
98030
98029
98028
98027
98024
98023
98022
98019
98014
98011
98010
98008
98007
98006
98005
98004
98003
98002
98001
0%
1%
2%
3%
4%
% of Observations in Population Group
Has Basement
FALSE
TRUE

It seems quite clear from this bar chart that the two populations follow different underlying distributions. Since we are treating the data set as our underlying populations, clear differences on the graph should be enough to tell us about how zip code distributions compare between two populations.

This data represents an ideal scenario to examine independence testing for high dimensional multinomial distributions since there are 70 distinct zip codes included in the King County, Washington data set.

4 Results

We now walk through the results of various simulation studies to assess the performance of the permutation test approach.

4.1 Two-Sample Testing

We first perform two-sample testing utilizing a Mann-Whitney U-statistic to compare the median and shape of two distributions for the price variable.

4.1.1 Type II Error

We start with data We'll start with a scenario where the null hypothesis is actually false because we are drawing the two samples from separate populations - homes with and without basements. For each simulation, we record whether our test correctly rejects the null hypothesis or fails to do so at a significance level of $\alpha = 0.05$, checking both a standard Mann-Whitney U-test and one utilizing a permutation distribution.

Table 8: Type II Error Rate of Two-Sample Testing Using a Mann-Whitney U-Statistic

Observations	Standard.Type.II.Error.Rate	Permutation.Type.II.Error.Rate
10	73.2%	76.8%
20	60.8%	62.2%
30	49.8%	50.8%
50	32.4%	33.4%
100	8.6%	8.8%

From these results, we see that the permutation based approach performs similarly to the standard Mann-Whitney U test in terms of minimizing the Type II error rate across all sample sizes.

4.1.2 Type I Error

Now we run simulations where the null hypothesis should in fact be true. We are drawing both samples from the same underlying population, where houses do not have a basement.

Table 9: Type I Error Rate of Two-Sample Testing Using a Mann-Whintey U-Statistic

Observations	Standard.Type.I.Error.Rate	Permutation.Type.I.Error.Rate
10	10%	5.2%
20	8%	4%
30	8.8%	4.8%

Observations	Standard.Type.I.Error.Rate	Permutation.Type.I.Error.Rate
50	8.6%	3.2%
100	10.2%	4.6%

From these simulations, it appears the permutation test peforms slightly better than the standard Mann-Whitney U test that uses a normal approximation for the test statistic distributionin terms of the Type I error rate across all sample sizes.

The permutation based approach, at least for the relatively small sample sizes considered, appears to match the performance of the standard Mann-Whitney U-test as far as Type II error rates are concerned while reducing Type I error rates. This highlights the value of the permutation-based approach as a tool for two-sample hypothesis testing in small sample sizes with non-normally distributed data.

4.2 Independence Testing

Next we turn to independence testing. We are testing the null hypothesis that there is no relationship between whether or not a house has a basement and which zip code it is located in. This represents a high-dimensional setting, given the high number of zip codes and anticipated sample sizes we will study ($n \leq 100$).

The Chi-squared test can be used for this type of testing procedure. However, p-values for the Chi-squared distribution assume certain minimum expected counts per cell. In a larger Chi-squared table, the test assumes that all expected counts must be at least 1 and no more than 20% of cells can have expected frequencies less than 5, or else the p-value may not be valid. We anticipate this to be a major issue in this circumstance and will likely lead to the presence of additional Type I and Type II errors when the Chi-squared test is applied. We perform the Chi-squared test anyway in order to highlight the utility of the alternative permutation-based approach using a U-statistic proposed by Kim et al. (2022).

For the sake of comparison, we also report out Type I and II error rates for a Chi-squared test performed with simulated p-values (generated through Monte Carlo simulation using a bootstrapping procedure). Bootstrapping is similar to the permutation-based approach for calculating p-values. However, the Chi-squared test with simulated p-values still differs significantly from the Kim et al. in that it is not a U-statistic and the test statistic follows a different design/formula from the one specifically proposed by Kim for high-dimensional multinomial testing (2022). We include it to provide another reference point for comparison of the new approach, since p-values calculated through reference to the standard χ^2 distribution are expected to be relatively inaccurate for this use case.

4.2.1 Type II Errors

We now perform independence testing across a number of simulations, comparing the Type II error rates generated through use of a χ^2 test with standard p-values (calculated through reference to a χ^2 distribution), a χ^2 test with simulated (Monte Carlo) p-values, through the multinomial U-statistic permutation test proposed Kim et al (2022).

Table 10: Type II Error Rate of Multinomial Independence Testing

Observations	Standard.Chi.Square.Type.II.Error.Rate	Bootstrap.Chi.Square.Type.II.Error.Rate	Permutation.Type.II.Error.Rate
10	100%	100%	90%

From these simulations, it appears the permutation test performs...

4.2.2 Type I Errors

Table 11: Type I Error Rate of Multinomial Independence Testing

Observations	Standard.Chi.Square.Type.I.Error.Rate	Bootstrap.Chi.Square.Type.I.Error.Rate	Permutation.Type.I.Error.Rate
10	10%	0%	20%

From these simulations, it appears the permutation test performs...

5 Conclusions

Our simulations show that the permutation-test based approach is a useful tool when it comes to hypothesis testing for small sample sizes. The theoretical guarantees on error rates given by Kim et al. are evident in the results of the simulations.

We first looked at two-sample testing, comparing the medians of two numeric samples of data. At least for the small sample sizes considered, the Type II error rate for p-values obtained through permutation versus through approximation to the normal distribution are roughly similar. However, the permutation approach appeared to reduce Type I error rates. This highlights the value of the permutation-based approach as a tool for two-sample hypothesis testing in small sample sizes with non-normally distributed data.

We then looked at independence testing for two categorical variables. We found ...

However, we do note that permutation testing has some drawbacks, namely the computational complexity. Permutation testing is not a very practical approach for larger data sets due to the time required to run. Even here with very small sample sizes, the simulations took a long time to run, particularly when calculating the multinomial U-statistic proposed by Kim et al. Future implementations of that statistic might wish to consider parallelizing computation, since permuted test statistics can be calculated simultaneously rather than sequentially. This would speed up the process of generating a permutation distribution.

Still, permutation tests are useful for smaller sample sizes. While so much of today's data comes in large data sets, there are settings in which we need to draw inferences from small sample sizes, such as medical studies with a complex or costly procedure or for a rare disease where it is difficult to recruit a large pool of subjects. Another example is in comparing the performance of large machine learning or AI models where training such models may take days or longer, so training these models many times to produce a large data set upon which to draw inferences is impractical.

6 References

7 Appendix (Code)

The following appendix contains all the code used in this project. The code also be found, broken out more into separate files/functions, on github at www.github.com/mvered.

```
# packages
library(tidyverse)
library(skimr)
library(quickcode)
library(kableExtra)
```

```r
# formatting settings
knitr::opts_chunk$set(echo = FALSE, warning=FALSE, message=FALSE)
options(digits=1)
options(scipen=999)

# global params
n_sims <- 500
n_perms <- 500
n_obs <- c(10, 20, 30, 50, 100)

# calculate mann-whitney u statistic
mann_whitney_u <- function(x, y){
  n_x <- length(x)
  n_y <- length(y)
  pooled <- c(x, y)

  ranks <- rank(pooled, ties.method = "average")
  #u_x <- sum(ranks[1:n_x]) -  (n_x*(n_x+1))/2
  #u_y <- sum(ranks[(n_x+1):(n_x+n_y)]) - (n_y*(n_y+1))/2
  u_x <- n_x*n_y + (n_x*(n_x+1))/2 - sum(ranks[1:n_x])
  #u_y <- n_x*n_y + (n_y*(n_y+1))/2 - sum(ranks[(n_x+1):(n_x+n_y)])
  return(u_x)
}

# new multinomial U-test statistic proposed by Kim et al

# this is functionally the same thing as sum_{k=1}^d I(x=k)I(y=k)
# x and y should be individual observations
# page 235 of original paper
g_multi <- function(x, y){
  if (x == y){
    return(1)
  } else if (x != y){
    return(0)
  }
}

# kernel function
# takes as inputs 2 element tuples from each distribution
# page 234 of original paper
h_ts <- function(y1,y2,z1,z2){
  total <- g_multi(y1,y2) + g_multi(z1,z2) - g_multi(y1,z2) - g_multi(y2,z1)
  return(total)
}

# calculate u statistic
# expects vectors of sample values (categorical data)
# page 234 of original paper
multinomial_u <- function(sample1, sample2){

  # get initial part of u statistic
  n1 <- length(sample1)
  n2 <- length(sample2)
```

```
  frac <- 1/(n1*(n1-1)*n2*(n2-1))

  # get tuples
  y_tuples <- combn(sample1, 2, simplify=TRUE)
  z_tuples <- combn(sample2, 2, simplify=TRUE)

  # keep running sum
  sum <- 0

  # outer sum
  for (i in 1:ncol(y_tuples)){

    # inner sum
    for (j in 1:ncol(z_tuples)){

      # run kernel function
      sum <- sum + h_ts(y_tuples[1,i], y_tuples[2,i],
                        z_tuples[1,j], z_tuples[2, j])
    }

  }

  # return total u statistic
  return(frac*sum)

}

# read in data
data <- read.csv("../data/home_data.csv") |>

  # fix date column formatting
  mutate(date = paste0(substr(date,1,4),"-",
                       substr(date,5,6),"-",
                       substr(date,7,8)),
         date = as.Date(date),

  # zip is categorical
  zipcode = factor(zipcode),

  # waterfront is binary
  waterfront = case_when(
    waterfront == 0 ~ FALSE,
    waterfront == 1 ~ TRUE),

  # these are ordinal data
  view = factor(view, levels = seq(0,4,by=1), ordered = TRUE),
  condition = factor(condition, levels = seq(1,5,by=1), ordered=TRUE),
  grade = factor(grade, levels=seq(1,13,by=1), ordered=TRUE),

  # id is a label
  id = as.character(id),

  # convert price to be in thousands of dollars
```

```
  price = price/1000,

  # create a couple additional binary variables
  has_basement = case_when(
    sqft_basement > 0 ~ TRUE,
    sqft_basement == 0 ~ FALSE),
  was_renovated = case_when(
    yr_renovated == 0 ~ FALSE,
    yr_renovated > 0 ~ TRUE),
  )

# summary statistics for data set overall
data |>
  skim() |>
  select(-numeric.hist)

# total counts on properties with/without basement
n_has_basement <- data |> filter(has_basement == TRUE) |> nrow()
n_no_basement <- data |> filter(has_basement == FALSE) |> nrow()

# create table with quantiles for properties with/without basement
data |>
  group_by(has_basement) |>
  summarize(`Price Q25` = quantile(price, 0.25),
            `Median Price` = quantile(price, 0.5),
            `Price Q75` = quantile(price, 0.75)) |>
  kable()

# histogram showing distribution of price for basement/no basement houses
data |>
  ggplot(aes(x = price, fill=has_basement)) +
  geom_histogram(bins = 40, color="white") +
  facet_wrap(~has_basement,
             labeller = labeller(has_basement = c(`FALSE` = "No Basement",
                                                  `TRUE` = "Has Basement"))) +
  scale_x_continuous(labels = scales::dollar_format(scale = 1e-3, suffix = "M")) +
  labs(x = "Price", y = "Count of Observations",
       fill = "Has Basement") +
  theme(panel.spacing = unit(2, "lines"))

# threshold for outliers to hide on plot so we see bulk of points more clearly
outlier_threshold <- quantile(data$price, .99)

# zoomed in histogram showing distribution of price for basement/no basement houses
data |>
  filter(price < outlier_threshold) |>
  ggplot(aes(x = price, fill=has_basement, color=has_basement)) +
  geom_density(alpha=0.4) +
  scale_x_continuous(labels = scales::dollar_format(scale = 1e-3, suffix = "M")) +
  labs(x = "Price", y = "Density",fill="Has Basement",color="Has Basement") +
  theme(panel.spacing = unit(2, "lines"))

# get percentage of each group per zip code
```

```r
zip_by_basement <- data |>
  group_by(has_basement, zipcode) |>
  summarize(count = n()) |>
  ungroup() |>
  group_by(has_basement) |>
  mutate(percentage = (count/sum(count)))

# coordinate flipped column chart of zip code distributions
zip_by_basement |>
  ggplot(aes(x = zipcode, y=percentage, fill=has_basement)) +
  geom_col(width = 0.7, position = position_dodge(width = 0.8)) +
  scale_y_continuous(labels = scales::percent) +
  coord_flip() +
  labs(x = "Zip Code", y = "% of Observations in Population Group", fill = "Has Basement") +
  theme(
    axis.text = element_text(size = 15),
    axis.title = element_text(size = 26),
    legend.title = element_text(size = 20),
    legend.text = element_text(size = 15)
  )

# set up for sampling procedure
with_basement <- data |>
  filter(has_basement == TRUE)
no_basement <- data |>
  filter(has_basement == FALSE)


# get p-value for mann-whitney u using permutation distribution
mann_whitney_perm <- function(x, y, u_observed, n_perms){

  #n_perms <- 1000
  pooled <- c(x, y)
  perm_dist <- rep(NA, n_perms)

  for (i in 1:n_perms){
    permuted_data <- sample(pooled, replace = FALSE)
    x_star <- permuted_data[1:length(x)]
    y_star <- permuted_data[(length(x)+1):length(pooled)]
    perm_dist[i] <- mann_whitney_u(x_star, y_star)
  }
  p_value <-(sum(abs(perm_dist) >= abs(u_observed)) + 1) / (n_perms + 1)
  return(p_value)
}

# get p-value for mann-whitney u using normal approximation
mann_whitney_normal <- function(x, y, u_observed){
  n_x <- length(x)
  n_y <- length(y)

  mu <- (n_x*n_y)/2
  sigma <- sqrt((n_x*n_y*(n_x+n_y+1))/12)
  z_score <- (u_observed - mu)/sigma
```

```r
  p_value <- 1-pnorm(abs(z_score))
  return(p_value)
}

# simulation function for two-sample testing
simulation_two_sample <- function(x, y, n = 100, alpha = 0.05){

  # "original" sample data
  x <- sample(x$price, size = n, replace = FALSE)
  y <- sample(y$price, size = n, replace = FALSE)

  # empty vector to store results
  results <- c(NA, NA)
  names(results) <- c("standard", "permutation")

  # calculate mann-whitney u statistic
  u_observed <- mann_whitney_u(x, y)

  # get p-value from normal approximation
  standard_p <- mann_whitney_normal(x, y, u_observed)
  if (standard_p <= alpha){
    results["standard"] <- "reject"
  } else {
    results["standard"] <- "fail to reject"
  }

  # get p-value from permutation distribution
  permutation_p <- mann_whitney_perm(x, y, u_observed, n_perms)
  if (permutation_p <= alpha){
     results["permutation"] <- "reject"
  } else {
     results["permutation"] <- "fail to reject"
  }

  # return results of this simulation
  return(results)
}

# run a simulation where the correct answer is to reject the null
error_t2_standard <- rep(NA, length(n_obs))
error_t2_perm <- rep(NA, length(n_obs))

# run the simulation for different sample sizes
for (j in 1:length(n_obs)){
  standard <- rep(NA, n_sims)
  perm <- rep(NA, n_sims)

  for (i in 1:n_sims){
    temp <- simulation_two_sample(x = no_basement, y = with_basement,
                                  n = n_obs[j], n_perms, alpha=0.05)
    standard[i] <- temp["standard"]
    perm[i] <- temp["permutation"]
  }
```

```
  error_standard <- (length(standard[standard == "fail to reject"])/n_sims)*100
  error_t2_standard[j] <- paste0(round(error_standard,2),"%")

  error_perm <- (length(perm[perm == "fail to reject"])/n_sims)*100
  error_t2_perm[j] <- paste0(round(error_perm,2),"%")
}

# tally up type II errors
two_sample_sims.type_2 <- data.frame(`Observations` = n_obs,
                                     `Standard Type II Error Rate` = error_t2_standard,
                                     `Permutation Type II Error Rate` = error_t2_perm)

two_sample_sims.type_2 <- readRDS("../output/two_sample_sims_type_2.rds")

two_sample_sims.type_2 |>
  kable(caption = "Type II Error Rate of Two-Sample Testing Using a Mann-Whitney U-Statistic")

# run a simulation where the correct answer is fail to reject
error_t1_standard <- rep(NA, length(n_obs))
error_t1_perm <- rep(NA, length(n_obs))


for (j in 1:length(n_obs)){
  standard <- rep(NA, n_sims)
  perm <- rep(NA, n_sims)

  for (i in 1:n_sims){
    temp <- simulation_two_sample(with_basement, with_basement, n_obs[j], n_perms, alpha=0.05)
    standard[i] <- temp["standard"]
    perm[i] <- temp["permutation"]
  }

  error_standard <- (length(standard[standard == "reject"])/n_sims)*100
  error_t1_standard[j] <- paste0(round(error_standard,2),"%")

  error_perm <- (length(perm[perm == "reject"])/n_sims)*100
  error_t1_perm[j] <- paste0(round(error_perm,2),"%")
}

two_sample_sims.type_1 <- data.frame(`Observations` = n_obs,
                                     `Standard Type I Error Rate` = error_t1_standard,
                                     `Permutation Type I Error Rate` = error_t1_perm)

# two sample type I error table
two_sample_sims.type_1 <- readRDS("../output/two_sample_sims_type_1.rds")

two_sample_sims.type_1 |>
  kable(caption = "Type I Error Rate of Two-Sample Testing Using a Mann-Whintey U-Statistic")

# conversion for ease of testing
no_basement <- no_basement |>
  mutate(zipcode = as.character(zipcode))
with_basement <- with_basement |>
```

```r
  mutate(zipcode = as.character(zipcode))


# get p-value for multinomial u using permutation distribution
get_multinomial_u_p <- function(x, y, n_perms){

  # observed sample statistic
  u_observed <- multinomial_u(x, y)

  # set up for permutation distribution
  pooled <- c(x, y)
  perm_dist <- rep(NA, n_perms)

  for (i in 1:n_perms){
    permuted_data <- sample(pooled, replace = FALSE)
    x_star <- permuted_data[1:length(x)]
    y_star <- permuted_data[(length(x)+1):length(pooled)]
    perm_dist[i] <- multinomial_u(x_star, y_star)
  }
  p_value <-(sum(abs(perm_dist) >= abs(u_observed)) + 1) / (n_perms + 1)
  print(p_value)
  return(p_value)
}

# simulation function for two-sample testing
simulation_independence_test <- function(x, y, n = 100, alpha = 0.05){

  # "original" sample data
  x <- sample(x, size = n, replace = FALSE)
  y <- sample(y, size = n, replace = FALSE)

  # empty vector to store results
  results <- c(NA, NA, NA)
  names(results) <- c("chisq_standard","chisq_simulated","permutation")
  #results <- c(NA, NA)
  #names(results) <- c("chisq_standard","permutation")

  # perform standard chi-squared test
  chisq_standard_p <- chisq.test(x, y)$p.value
    if (chisq_standard_p <= alpha){
    results["chisq_standard"] <- "reject"
  } else {
    results["chisq_standard"] <- "fail to reject"
  }

  # perform chi-squared test with simulated/bootstrap p-values
  chisq_simulated_p <- chisq.test(x, y,
                                  simulate.p.value = TRUE, B=n_perms)$p.value
  if (chisq_simulated_p <= alpha){
    results["chisq_simulated"] <- "reject"
  } else {
    results["chisq_simulated"] <- "fail to reject"
  }
```

```
  # perform multinomial u test with permutation
  multinomial_u_p <- get_multinomial_u_p(x, y)
  if (multinomial_u_p <= alpha){
     results["permutation"] <- "reject"
  } else {
     results["permutation"] <- "fail to reject"
  }

  # return results of this simulation
  return(results)
}

# run a simulation where the correct answer is to reject the null
error_t2_chisq_standard <- rep(NA, length(n_obs))
error_t2_chisq_simulated <- rep(NA, length(n_obs))
error_t2_perm <- rep(NA, length(n_obs))

# run the simulation for different sample sizes
for (j in 1:length(n_obs)){
  chisq_standard <- rep(NA, n_sims)
  chisq_simulated <- rep(NA, n_sims)
  perm <- rep(NA, n_sims)

  # run the simulations
  for (i in 1:n_sims){
    # run one simulation
    temp <- simulation_independence_test(x = no_basement$zipcode,
                                         y = with_basement$zipcode,
                                         n = n_obs[j], n_perms = n_perms,
                                         alpha=0.1)
    # save info on whether null was rejected or failed to reject
    chisq_standard[i] <- temp["chisq_standard"]
    chisq_simulated[i] <- temp["chisq_simulated"]
    perm[i] <- temp["permutation"]
  }

  # Error rate for standard chi-square test
  error_chisq_standard <- (length(chisq_standard[chisq_standard== "fail to reject"])/
                             n_sims)*100
  error_t2_chisq_standard[j] <- paste0(round(error_chisq_standard,2),"%")

  # Error rate for chi-squared test with bootstrap p-value
  error_chisq_simulated <- (length(chisq_simulated[chisq_simulated== "fail to reject"])/
                              n_sims)*100
  error_t2_chisq_simulated[j] <- paste0(round(error_chisq_simulated,2),"%")

  # Error rate for permuation test
  error_perm <- (length(perm[perm == "fail to reject"])/n_sims)*100
  error_t2_perm[j] <- paste0(round(error_perm,2),"%")
}

# tally up type II errors
independence_sims.type_2 <- data.frame(`Observations` = n_obs,
```

```
                                        `Standard Chi-Square Type II Error Rate` = error_t2_chisq_standa
                                        `Bootstrap Chi-Square Type II Error Rate` = error_t2_chisq_simul
                                        `Permutation Type II Error Rate` = error_t2_perm)



# independence testing type 2 error table
independence_sims.type_2 <- readRDS("../output/independence_sims_type_2.rds")

independence_sims.type_2 |>
  kable(caption = "Type II Error Rate of Multinomial Independence Testing")

# initialize empty vectors to store type I error rate for each sample size
error_t1_chisq_standard <- rep(NA, length(n_obs))
error_t1_chisq_simulated <- rep(NA, length(n_obs))
error_t1_perm <- rep(NA, length(n_obs))

# run the simulation for different sample sizes
for (j in 1:length(n_obs)){
  chisq_standard <- rep(NA, n_sims)
  chisq_simulated <- rep(NA, n_sims)
  perm <- rep(NA, n_sims)

  # run simulations
  for (i in 1:n_sims){
    # run one simulation
    temp <- simulation_independence_test(x = with_basement$zipcode,
                                         y = with_basement$zipcode,
                                         n = n_obs[j], n_perms = n_perms,
                                         alpha=0.1)
    # store whether we rejected or failed to reject the null
    chisq_standard[i] <- temp["chisq_standard"]
    chisq_simulated[i] <- temp["chisq_simulated"]
    perm[i] <- temp["permutation"]
  }

  # calculate Type I error rate
  error_chisq_standard <- (length(chisq_standard[chisq_standard== "reject"])/
                             n_sims)*100
  error_t1_chisq_standard[j] <- paste0(round(error_chisq_standard,2),"%")

  error_chisq_simulated <- (length(chisq_simulated[chisq_simulated== "reject"])/
                              n_sims)*100
  error_t1_chisq_simulated[j] <- paste0(round(error_chisq_simulated,2),"%")

  error_perm <- (length(perm[perm == "reject"])/n_sims)*100
  error_t1_perm[j] <- paste0(round(error_perm,2),"%")
}

# tally up type II errors
independence_sims.type_1 <- data.frame(`Observations` = n_obs,
                                       `Standard Chi-Square Type I Error Rate` = error_t1_chisq_standar
                                       `Bootstrap Chi-Square Type I Error Rate` = error_t1_chisq_simula
```

```
                                    `Permutation Type I Error Rate` = error_t1_perm)



# independence testing type 2 error table
independence_sims.type_1 <- readRDS("../output/independence_sims_type_1.rds")

independence_sims.type_1 |>
  kable(caption = "Type I Error Rate of Multinomial Independence Testing")
```