Data preparation and feature selection

```
library(stringr)
library(dplyr)
```

The tasks:

- Data preparation: prepare a data set for the use in a machine learning task
- Feature selection: select features that might be relevant for a classification task

Load, inspect, and transform the data

The data originates from the Nutrition facts for Starbucks Menu, where further information about it can be found (including the description of all the variables).

Note: the original data set has been slightly adapted for this class.

Load the data

```
sb_drinks = read.csv("data/starbucks_drinks.csv")
```

Inspect the data

```
glimpse(sb_drinks)
```

```
## Rows: 240
## Columns: 20
## $ Beverage_category     <chr> "Coffee", "Coffee", "Coffee", "Coffee", "Classic~
## $ Beverage              <chr> "Brewed Coffee", "Brewed Coffee", "Brewed Coffee~
## $ Milk                  <chr> "None", "None", "None", "None", "Nonfat_Milk", "~
## $ Size                  <chr> "Short", "Tall", "Grande", "Venti", "Short", "Gr~
## $ Calories              <int> 3, 4, 5, 5, 70, 100, 70, 100, 150, 110, 130, 190~
## $ Total_Fat_g           <chr> "0.1", "0.1", "0.1", "0.1", "0.1", "3.5", "2.5",~
## $ Trans_Fat_g           <dbl> 0.0, 0.0, 0.0, 0.0, 0.1, 2.0, 0.4, 0.2, 3.0, 0.5~
## $ Saturated_Fat_g       <dbl> 0.0, 0.0, 0.0, 0.0, 0.0, 0.1, 0.0, 0.0, 0.2, 0.0~
## $ Sodium_mg             <int> 0, 0, 0, 0, 5, 15, 0, 5, 25, 0, 5, 30, 0, 10, 35~
## $ Total_Carbohydrates_g <int> 5, 10, 10, 10, 75, 85, 65, 120, 135, 105, 150, 1~
## $ Cholesterol_mg        <int> 0, 0, 0, 0, 10, 10, 6, 15, 15, 10, 19, 19, 13, 2~
## $ Dietary_Fibre_g       <int> 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 2, ~
## $ Sugars_g              <int> 0, 0, 0, 0, 9, 9, 4, 14, 14, 6, 18, 17, 8, 23, 2~
## $ Protein_g             <dbl> 0.3, 0.5, 1.0, 1.0, 6.0, 6.0, 5.0, 10.0, 10.0, 8~
## $ Vitamin_A_DV          <chr> "0%", "0%", "0%", "0%", "10%", "10%", "6%", "15%~
## $ Vitamin_C_DV          <chr> "0%", "0%", "0%", "0%", "0%", "0%", "0%", "0%", ~
## $ Calcium_DV            <chr> "0%", "0%", "0%", "2%", "20%", "20%", "20%", "30~
## $ Iron_DV               <chr> "0%", "0%", "0%", "0%", "0%", "0%", "8%", "0%", ~
## $ Caffeine_mg           <chr> "175", "260", "330", "410", "75", "75", "75", "7~
## $ Is_coffe_drink        <chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes",~
```

The first thing to do is to examine column types and ponder if the type of each column corresponds to its expected type. For example, `Total_Fat_g` is loaded as a char variable even though we would expect it to be numeric. The same is with `Caffeine_mg`. This happens due to same irregularities in the numeric data, which prevent their proper loading from the file. Let's check:

```
unique(sb_drinks$Total_Fat_g) |> sort()
```

```
##  [1] "0"   "0.1" "0.2" "0.3" "0.4" "0.5" "1"   "1.5" "10"  "11"  "13"  "15"
## [13] "2"   "2.5" "3"   "3 2" "3.5" "4"   "4.5" "5"   "6"   "7"   "8"   "9"
```

Note the presence of erroneous value "3 2"

```
unique(sb_drinks$Caffeine_mg) |> sort()
```

```
##  [1] ""    "-"   "0"   "10"  "100" "105" "110" "120" "125" "130" "140" "145"
## [13] "15"  "150" "165" "170" "175" "180" "20"  "225" "235" "25"  "260" "30"
## [25] "300" "330" "410" "50"  "55"  "65"  "70"  "75"  "80"  "85"  "90"  "95"
```

Note the missing values in the form of " " and "-"

Since these two variables are essentially numeric, we will transform them into numeric type

```
sb_drinks$Total_Fat_g <- as.numeric(sb_drinks$Total_Fat_g)
```

```
## Warning: NAs introduced by coercion
```

```
sb_drinks$Caffeine_mg <- as.numeric(sb_drinks$Caffeine_mg)
```

```
## Warning: NAs introduced by coercion
```

Note that this process introduces NA (for those values that are not numbers) and we will deal with this a bit later.

Now, consider the variables related to the presence of some vitamins and minerals in Starbucks drinks:

```
unique(sb_drinks$Vitamin_A_DV)
```

```
##  [1] "0%"  "10%" "6%"  "15%" "20%" "30%" "25%" "8%"  "4%"  "2%"  "50%"
```

Obviously, this is also essentially a numeric variable, but due to the used notation, it was stored as a char. Let's transform this and the other three variables with the same value encoding pattern, into numeric

```
char_to_num <- function(x) {
  str_replace(x, pattern = "%", replacement = "") |> as.numeric()
}

sb_drinks$Vitamin_A_DV <- char_to_num(sb_drinks$Vitamin_A_DV)
```

```r
class(sb_drinks$Vitamin_A_DV)
```

```
## [1] "numeric"
```

```r
sb_drinks$Vitamin_C_DV <- char_to_num(sb_drinks$Vitamin_C_DV)
sb_drinks$Calcium_DV <- char_to_num(sb_drinks$Calcium_DV)
sb_drinks$Iron_DV <- char_to_num(sb_drinks$Iron_DV)
```

Let's now examine the remaining character variables and see if we can transform them into factors

```r
char_vars <- c(1:4, 20)
apply(sb_drinks[,char_vars], 2, n_distinct)
```

```
## Beverage_category          Beverage              Milk              Size
##                 9                32                 6                 5
##    Is_coffe_drink
##                 2
```

Considering the small number of observations in the data set (240), the second variable - `Beverage` - has too many distinct values to be used as factors in any prediction task. So, we will leave it as is and transform the other four.

Let's first examine the distribution of those four variables, that is, how frequent the distinct values are

```r
char_vars <- c(1, 3, 4, 20)
apply(sb_drinks[,char_vars], 2, function(x) table(x, useNA = 'ifany'))
```

```
## $Beverage_category
## x
##           Classic Espresso Drinks                            Coffee
##                                56                                 4
##       Frappuccino® Blended Coffee       Frappuccino® Blended Crème
##                                36                                13
## Frappuccino® Light Blended Coffee            Shaken Iced Beverages
##                                12                                18
##         Signature Espresso Drinks                         Smoothies
##                                40                                 9
##                  Tazo® Tea Drinks
##                                52
##
## $Milk
## x
##           -     2%_Milk        None Nonfat_Milk     Soymilk  Whole_Milk
##           1          50          25          83          65          16
##
## $Size
## x
## Grande  Short   Tall  Venti   <NA>
##    165     16     29     28      2
##
## $Is_coffe_drink
## x
##  No Yes
## 102 138
```

Note that for `Milk` and `Size` we, in fact, have 5 and 4 distinct values, respectively, as one of the values is a mark of the missing value(s).

To avoid having "-" a level of the `Milk` variable, we will first replace it with NA and then proceed to transform the variable to factor

```
sb_drinks$Milk[sb_drinks$Milk == "-"] <- NA
sb_drinks$Milk <- as.factor(sb_drinks$Milk)
class(sb_drinks$Milk)
```

```
## [1] "factor"
```

```
levels(sb_drinks$Milk)
```

```
## [1] "2%_Milk"     "None"        "Nonfat_Milk" "Soymilk"     "Whole_Milk"
```

Observe that factor levels are given in the alphabetical order; more precisely, based on the ASCII values of the leading characters.

Now, for the `Size` variable. This variable can be considered *ordinal* since its values have an ordering - a way to sort from the smallest to the largest. So, when transforming it to a factor, we should explicitly define the order of values, that is, factor levels

```
sb_drinks$Size <- factor(sb_drinks$Size, levels = c("Short", "Tall", "Grande", "Venti"))
levels(sb_drinks$Size)
```

```
## [1] "Short"  "Tall"   "Grande" "Venti"
```

IMPORTANT note: only ordinal variables - such as `Size` - can be transform to a number, to be used with algorithms (like kNN or k-means) that accepts numeric variables only. True categorical variables - such as `Milk` - should **not** be turned into numbers.

Let's now consider the `Beverage_category` variable:

```
table(sb_drinks$Beverage_category) |> sort()
```

```
##
##                              Coffee                           Smoothies
##                                   4                                   9
## Frappuccino® Light Blended Coffee        Frappuccino® Blended Crème
##                                  12                                  13
##             Shaken Iced Beverages       Frappuccino® Blended Coffee
##                                  18                                  36
##         Signature Espresso Drinks                  Tazo® Tea Drinks
##                                  40                                  52
##           Classic Espresso Drinks
##                                  56
```

Considering the small number of occurrences of values 'Coffee', 'Smoothies', 'Frappuccino® Light Blended Coffee', and 'Frappuccino® Blended Creme', it is very likely - especially for 'Coffee' and 'Smoothies' - that when splitting data into training and test sets, we will end up with all 'Coffee' drinks or all 'Smoothies' drinks in the test set. This would cause a problem for many classifiers as they haven't "seen" such values in the training set and do not know how to handle them. To avoid such problems, while still making use of the information about beverage type, we can create a new variable - say `Beverage_group` - by aggregating similar beverage categories:

```
sb_drinks$Beverage_group = "Coffee"
sb_drinks$Beverage_group[sb_drinks$Beverage_category == 'Shaken Iced Beverages'] = "Ice-cold_Drinks"
sb_drinks$Beverage_group[startsWith(sb_drinks$Beverage_category, 'Frappuccino')] = "Frappuccino"
sb_drinks$Beverage_group[sb_drinks$Beverage_category %in% c('Smoothies', 'Tazo® Tea Drinks')] = "Non-Co
```

```
table(sb_drinks$Beverage_group) |> sort()
```

```
##
##   Ice-cold_Drinks       Frappuccino Non-Coffee_Drinks            Coffee
##                18                61                61               100
```

The final step is to transform the new variable (`Beverage_group`) into a factor:

```
sb_drinks$Beverage_group = as.factor(sb_drinks$Beverage_group)
```

The only thing left to do is to transform the binary variable `Is_coffee_drink` into a factor:

```
sb_drinks$Is_coffe_drink <- as.factor(sb_drinks$Is_coffe_drink)
levels(sb_drinks$Is_coffe_drink)
```

```
## [1] "No"  "Yes"
```

Note: Binary variables almost always serve as Boolean variables - stating if something is true or false - and can be transformed (from factor) to numeric to be used in an algorithm that works with numeric variables only.

Check the types again to verify that all variables are now of the expected type

```
glimpse(sb_drinks)
```

```
## Rows: 240
## Columns: 21
## $ Beverage_category     <chr> "Coffee", "Coffee", "Coffee", "Coffee", "Classic~
## $ Beverage              <chr> "Brewed Coffee", "Brewed Coffee", "Brewed Coffee~
## $ Milk                  <fct> None, None, None, None, Nonfat_Milk, 2%_Milk, So~
## $ Size                  <fct> Short, Tall, Grande, Venti, Short, Grande, Grand~
## $ Calories              <int> 3, 4, 5, 5, 70, 100, 70, 100, 150, 110, 130, 190~
## $ Total_Fat_g           <dbl> 0.1, 0.1, 0.1, 0.1, 0.1, 3.5, 2.5, 0.2, 6.0, 4.5~
## $ Trans_Fat_g           <dbl> 0.0, 0.0, 0.0, 0.0, 0.1, 2.0, 0.4, 0.2, 3.0, 0.5~
## $ Saturated_Fat_g       <dbl> 0.0, 0.0, 0.0, 0.0, 0.0, 0.1, 0.0, 0.0, 0.2, 0.0~
## $ Sodium_mg             <int> 0, 0, 0, 0, 5, 15, 0, 5, 25, 0, 5, 30, 0, 10, 35~
## $ Total_Carbohydrates_g <int> 5, 10, 10, 10, 75, 85, 65, 120, 135, 105, 150, 1~
## $ Cholesterol_mg        <int> 0, 0, 0, 0, 10, 10, 6, 15, 15, 10, 19, 19, 13, 2~
## $ Dietary_Fibre_g       <int> 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 2, ~
## $ Sugars_g              <int> 0, 0, 0, 0, 9, 9, 4, 14, 14, 6, 18, 17, 8, 23, 2~
## $ Protein_g             <dbl> 0.3, 0.5, 1.0, 1.0, 6.0, 6.0, 5.0, 10.0, 10.0, 8~
## $ Vitamin_A_DV          <dbl> 0, 0, 0, 0, 10, 10, 6, 15, 15, 10, 20, 20, 15, 3~
## $ Vitamin_C_DV          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 2, 0, ~
## $ Calcium_DV            <dbl> 0, 0, 0, 2, 20, 20, 20, 30, 30, 30, 40, 40, 40, ~
## $ Iron_DV               <dbl> 0, 0, 0, 0, 0, 0, 8, 0, 0, 15, 0, 0, 15, 0, 0, 2~
## $ Caffeine_mg           <dbl> 175, 260, 330, 410, 75, 75, 75, 75, 75, 75, 150,~
## $ Is_coffe_drink        <fct> Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes, Yes~
## $ Beverage_group        <fct> Coffee, Coffee, Coffee, Coffee, Coffee, Coffee, ~
```

Imputing missing values

Imputation is the replacement of missing values with the best approximations we can come up with. The reason for doing that: no algorithm works with missing values. Even if some of them *seem* to work, they either use internal algorithms to impute the missing values or simply discard observations or variables with missing values.

First, let's check for the presence of missing values. The `summary` function provides that information for numeric and factor variables:

```
summary(sb_drinks)
```

```
##  Beverage_category    Beverage                  Milk        Size
##  Length:240         Length:240         2%_Milk    :50   Short : 16
##  Class :character   Class :character   None       :25   Tall  : 29
##  Mode  :character   Mode  :character   Nonfat_Milk:83   Grande:165
##                                        Soymilk    :65   Venti : 28
##                                        Whole_Milk :16   NA's  :  2
##                                        NA's       : 1
##
##     Calories      Total_Fat_g      Trans_Fat_g    Saturated_Fat_g
##  Min.   :  0.0   Min.   : 0.000   Min.   :0.000   Min.   :0.00000
##  1st Qu.:120.0   1st Qu.: 0.200   1st Qu.:0.100   1st Qu.:0.00000
##  Median :190.0   Median : 2.500   Median :0.500   Median :0.00000
##  Mean   :195.4   Mean   : 2.928   Mean   :1.318   Mean   :0.03792
##  3rd Qu.:260.0   3rd Qu.: 4.500   3rd Qu.:2.000   3rd Qu.:0.10000
##  Max.   :510.0   Max.   :15.000   Max.   :9.000   Max.   :0.30000
##                  NA's   :1
##    Sodium_mg      Total_Carbohydrates_g Cholesterol_mg  Dietary_Fibre_g
##  Min.   : 0.000   Min.   :  0.0         Min.   : 0.00   Min.   :0.0000
##  1st Qu.: 0.000   1st Qu.: 70.0         1st Qu.:21.00   1st Qu.:0.0000
##  Median : 5.000   Median :125.0         Median :34.50   Median :0.0000
##  Mean   : 6.417   Mean   :130.0         Mean   :36.28   Mean   :0.8125
##  3rd Qu.:10.000   3rd Qu.:172.5         3rd Qu.:51.00   3rd Qu.:1.0000
##  Max.   :40.000   Max.   :340.0         Max.   :90.00   Max.   :8.0000
##
##     Sugars_g       Protein_g       Vitamin_A_DV     Vitamin_C_DV
##  Min.   : 0.00   Min.   : 0.000   Min.   : 0.000   Min.   :  0.000
##  1st Qu.:18.75   1st Qu.: 3.000   1st Qu.: 4.000   1st Qu.:  0.000
##  Median :32.00   Median : 6.000   Median : 8.000   Median :  0.000
##  Mean   :33.24   Mean   : 7.031   Mean   : 9.912   Mean   :  3.679
##  3rd Qu.:44.00   3rd Qu.:10.000   3rd Qu.:15.000   3rd Qu.:  0.000
##  Max.   :84.00   Max.   :20.000   Max.   :50.000   Max.   :100.000
##
##    Calcium_DV       Iron_DV        Caffeine_mg    Is_coffe_drink
##  Min.   : 0.00   Min.   : 0.000   Min.   :  0.0   No :102
##  1st Qu.:10.00   1st Qu.: 0.000   1st Qu.: 55.0   Yes:138
##  Median :20.00   Median : 2.000   Median : 75.0
##  Mean   :20.93   Mean   : 7.508   Mean   : 88.4
##  3rd Qu.:30.00   3rd Qu.:10.000   3rd Qu.:130.0
##  Max.   :60.00   Max.   :50.000   Max.   :410.0
##                                   NA's   :2
##            Beverage_group
##  Coffee           :100
```

```
##  Frappuccino      : 61
##  Ice-cold_Drinks  : 18
##  Non-Coffee_Drinks: 61
##
##
##
```

An alternative way to check for missing values in numeric and factor variables:

```
apply(sb_drinks[,3:21], 2, function(x) sum(is.na(x))) |> sort()
```

```
##              Calories           Trans_Fat_g       Saturated_Fat_g
##                     0                     0                     0
##            Sodium_mg Total_Carbohydrates_g        Cholesterol_mg
##                     0                     0                     0
##       Dietary_Fibre_g             Sugars_g             Protein_g
##                     0                     0                     0
##          Vitamin_A_DV          Vitamin_C_DV            Calcium_DV
##                     0                     0                     0
##               Iron_DV        Is_coffe_drink        Beverage_group
##                     0                     0                     0
##                  Milk           Total_Fat_g                  Size
##                     1                     1                     2
##           Caffeine_mg
##                     2
```

For character variables, we can check for missing values as follows:

```
char_vars <- c(1,2)
apply(sb_drinks[,char_vars], 2, function(x) sum(is.na(x) | trimws(x) == "" | trimws(x) == "-"))
```

```
## Beverage_category          Beverage
##                 0                 0
```

Note: you should include a check for any other character you've observe or expect to appear in any of the examined character variables

So, we have a couple of missing values in two numeric variables (Total_Fat_g, Caffeine_mg) and in two factor variables (Milk, Size).

Categorical variables with a small number of missing values

If there are only a couple of missing values in a factor variable, they are replaced by the 'majority class', that is, the most dominant value of that variable. When doing so, it is recommended to identify the dominant value among the subset of observations that are similar to the one with the missing value. For example, in our case, we can identify the dominant value for `Milk` in the category of beverages that the one with the missing value belongs to:

```
milk_missing_category <- sb_drinks$Beverage_category[is.na(sb_drinks$Milk)]
milk_missing_category
```

```
## [1] "Signature Espresso Drinks"
```

```r
sb_drinks$Milk[sb_drinks$Beverage_category == milk_missing_category] |> table()
```

```
##
##     2%_Milk         None Nonfat_Milk     Soymilk  Whole_Milk
##          12            4           12          11           0
```

As we have a tie between two values and the third close behind, let's see how things stand at the level of the overall data set

```r
table(sb_drinks$Milk)
```

```
##
##     2%_Milk         None Nonfat_Milk     Soymilk  Whole_Milk
##          50           25           83          65          16
```

Based on the above, we'll replace the missing value with "Nonfat_Milk":

```r
sb_drinks$Milk[is.na(sb_drinks$Milk)] <- "Nonfat_Milk"
```

Note: a better approach would be to find k nearest neighbors of the instance with the missing `Milk` value, based on all the attributes (except the missing one) and to replace the missing value with the majority value of those k nearest neighbors.

Next, we need to handle the missing values in the `Size` variable. Since the size of a drink is fairly independent of beverage category or group (can be verified with the Chi-square test), we will use the majority class on the overall data set to fill out the missing values

```r
table(sb_drinks$Size)
```

```
##
##  Short   Tall Grande  Venti
##     16     29    165     28
```

```r
sb_drinks$Size[is.na(sb_drinks$Size)] <- "Grande"
```

Numeric variables with a small number of missing values

In case a numeric variable has a few missing values, we typically replace the missing values with the average value of the variable on a subset of observations that are similar - in "important ways" - to the observation(s) with the missing value. For example, in case of the missing value of the `Total_Fat_g` variable, we can find an instance or a couple of them that are most similar in `Trans_Fat_g`, `Saturated_Fat_g`, and `Calories` values. This can be done, for example, using the kNN algorithm.

Here, we will use a simpler approach - take the average on a group of observations that belong to the same (relevant) category as the instance(s) with the missing value(s). For example, for the total fat value, the size of the drink and the kind of milk used are relevant, so we will compute average `Total_Fat_g` on a subset determined by these two variables:

```r
# First, identify values of Milk and Size for the observation with the missing value for Total_Fat_g
tot_fat_NA_milk <- sb_drinks$Milk[is.na(sb_drinks$Total_Fat_g)]
print(tot_fat_NA_milk)
```

```
## [1] Soymilk
## Levels: 2%_Milk None Nonfat_Milk Soymilk Whole_Milk
```

```
tot_fat_NA_size <- sb_drinks$Size[is.na(sb_drinks$Total_Fat_g)]
print(tot_fat_NA_size)
```

```
## [1] Grande
## Levels: Short Tall Grande Venti
```

```
# Next, compute the average (median) value of Total_Fat_g on all the drinks with the above identified s
avg_tot_fat <- median(sb_drinks$Total_Fat_g[sb_drinks$Milk == tot_fat_NA_milk &
                                            sb_drinks$Size == tot_fat_NA_size],na.rm = T)
avg_tot_fat
```

```
## [1] 3.5
```

```
# Finally, replace the missing value with the computed average value
sb_drinks$Total_Fat_g[is.na(sb_drinks$Total_Fat_g)] <- avg_tot_fat
```

Note: a more precise approach would be to first determine if the variable in question - `Total_Fat_g` in this case - has Normal distribution, and if it does, to use mean as the average value, otherwise, use median. However, since median and mean are very similar in case of Normal distribution and since we are already doing estimation, to simplify things, we use median right away.

Next, we replace the missing values of `Caffeine_mg` in a similar manner as for `Total_Fat_g`. For the quantity of caffeine, we can consider relevant the category and size of drink:

```
caffeine_NA_bev_cat <- sb_drinks$Beverage_category[is.na(sb_drinks$Caffeine_mg)]
print(caffeine_NA_bev_cat)
```

```
## [1] "Shaken Iced Beverages" "Smoothies"
```

```
caffeine_NA_size <- sb_drinks$Size[is.na(sb_drinks$Caffeine_mg)]
print(caffeine_NA_size)
```

```
## [1] Grande Grande
## Levels: Short Tall Grande Venti
```

Since here we have more than one missing value, we are using a loop to compute the average value on a relevant subset and then replace the missing values with the computed average value

```
caffeine_NA_size <- caffeine_NA_size[1] #since the two are the same

for(cat in caffeine_NA_bev_cat) {
  avg_caffeine <- median(sb_drinks$Caffeine_mg[sb_drinks$Beverage_category == cat &
                                               sb_drinks$Size == caffeine_NA_size], na.rm = T)
  sb_drinks$Caffeine_mg[sb_drinks$Beverage_category == cat &
                          sb_drinks$Size == caffeine_NA_size &
                          is.na(sb_drinks$Caffeine_mg)] <- avg_caffeine
}
```

There should be no more NAs,let's check

```
all(complete.cases(sb_drinks))
```

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

IMPORTANT note: if a variable has a lot of missing values, it should be removed from the data set. The above described imputation approaches are only suitable for a small number of missing values. There are sophisticated imputation methods that can be used to fill out many more missing values, but these are out of the scope of this course.

Feature selection

To select features to be used for creating a prediction model, we have to examine if and to what extent features are associated with the response (outcome) variable.

If we are familiar with the domain of the problem (prediction task), we can start from the knowledge and/or intuition about the predictors. On the other hand, if the domain is unknown to us (for example, predictions related to some sophisticated biological processes) or variable names are withdrawn (as is often the case in sensitive domains), we have to rely on some well establish general methods for feature selection (such as forward or backward selection).

We have already explored how to select variables in case of the regression task. Likewise, we have seen criteria for selecting variables for clustering. Now, we will consider the classification task.

First, let's create the outcome variable. For the sake of an example, we will create a variable that distinguishes between drinks that can be considered a healthier choice and those that are less healthy choice. We will define here healthier choice as drinks with below average value of sugar, cholesterol, and trans fat.

```
avg_sugar <- median(sb_drinks$Sugars_g)
avg_cholesterol <- median(sb_drinks$Cholesterol_mg)
avg_trans_fat <- median(sb_drinks$Trans_Fat_g)

sb_drinks$HealthierChoice <- ifelse(sb_drinks$Sugars_g < avg_sugar &
                                      sb_drinks$Cholesterol_mg < avg_cholesterol &
                                      sb_drinks$Trans_Fat_g < avg_trans_fat,
                                    yes = "Yes", no = "No")

sb_drinks$HealthierChoice <- as.factor(sb_drinks$HealthierChoice)
```

```
table(sb_drinks$HealthierChoice) |> prop.table() |> round(4)
```

```
##
##     No    Yes
## 0.7375 0.2625
```

Remove the variables that were used for the creation of outcome variable as these should not be used in a prediction model

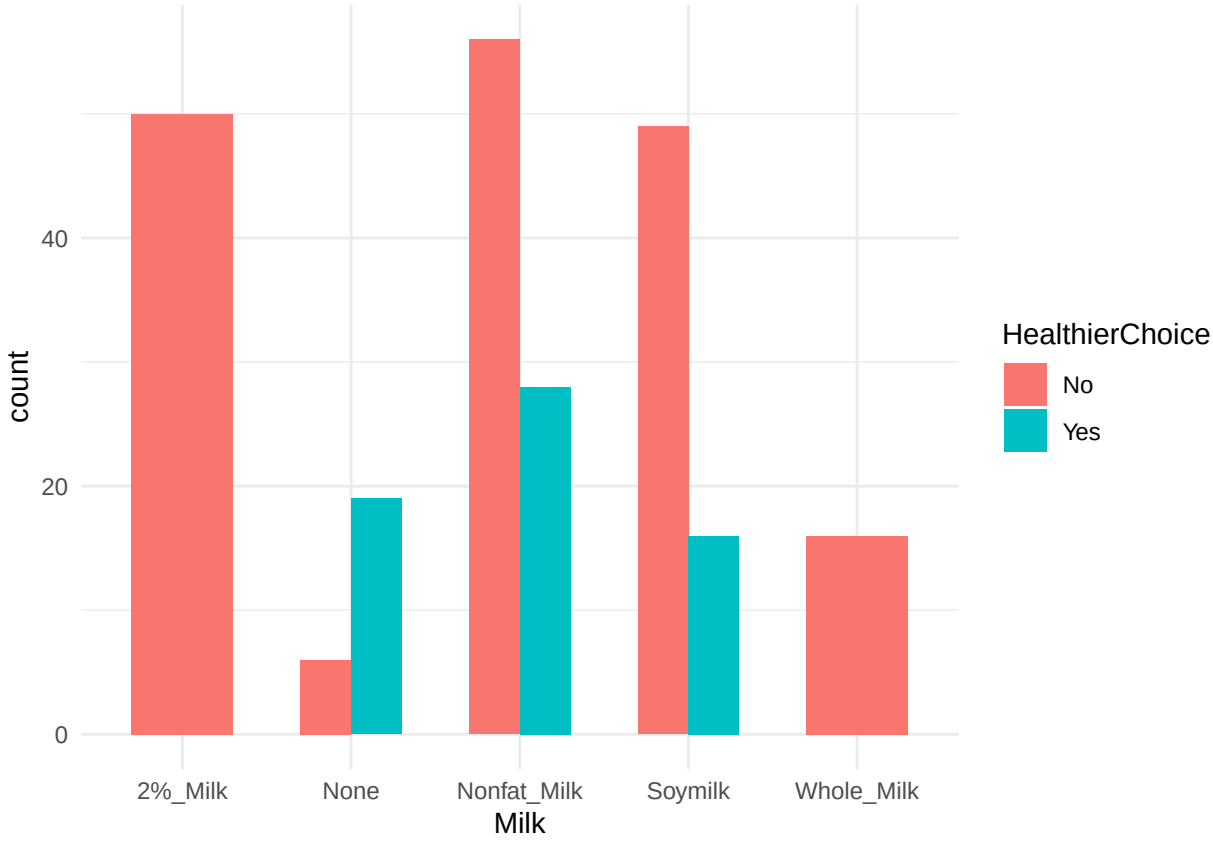
```
sb_drinks$Sugars_g <- NULL
sb_drinks$Cholesterol_mg <-NULL
sb_drinks$Trans_Fat_g <- NULL
```

We will examine the relevance of remaining variables for predicting the outcome, through visual exploratory analysis

```
library(ggplot2)
```

To examine association between a factor variable and the outcome (also factor) variable, we typically use bar plots. For example, we can check how milk used in drink is related to the drink being considered a healthier choice:

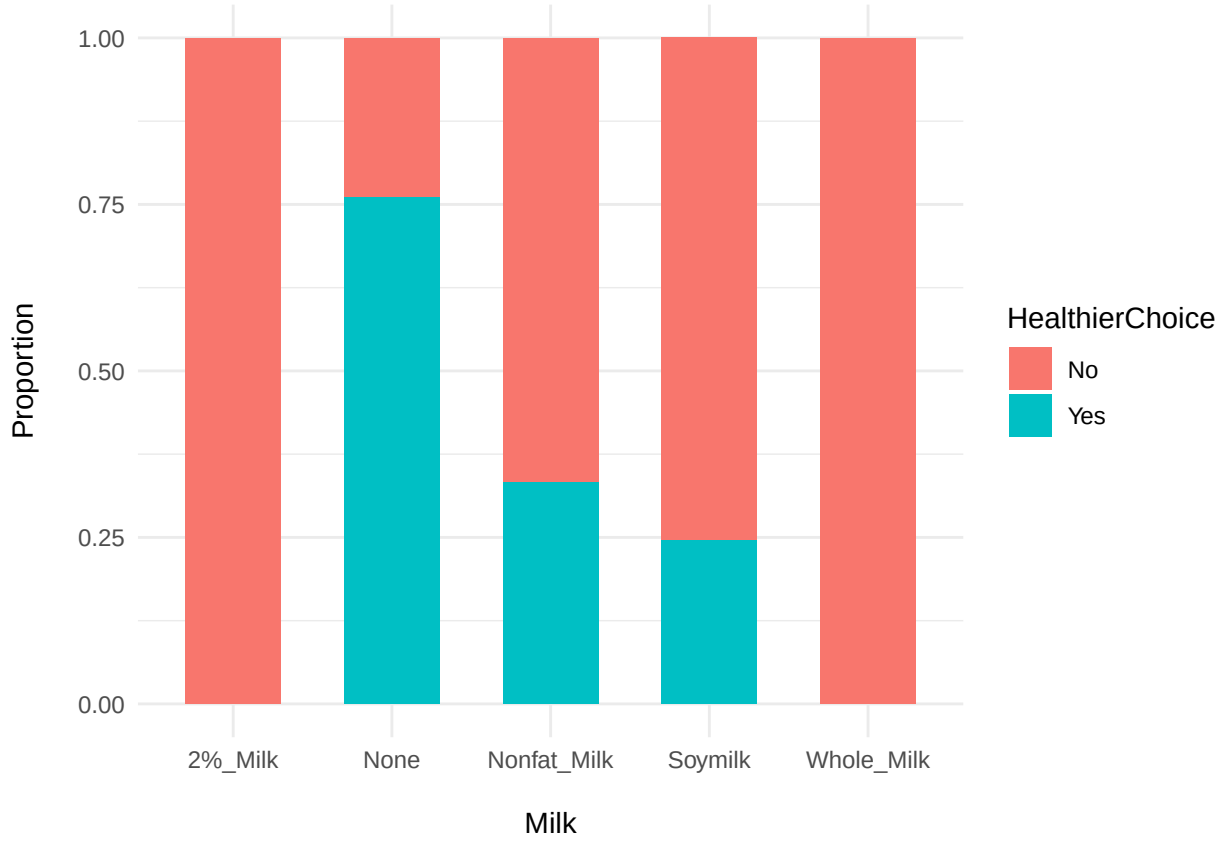
```
ggplot(sb_drinks,
       aes(x = Milk, fill = HealthierChoice)) +
  geom_bar(position = "dodge", width = 0.6) +
  theme_minimal()
```



What we are looking for is the difference in distribution of the outcome variable across the values of the examined variable. Specifically, in the context of the `Milk` variable, we want to see if there is equal or different probability of the Yes and No values for Soy milk vs Nonfat_Milk vs. Whole_Milk ... If they are different, it means that `Milk` variable is relevant for the prediction of the outcome variable as its values allow for differentiating between drinks that are healthier choice from those that are not.

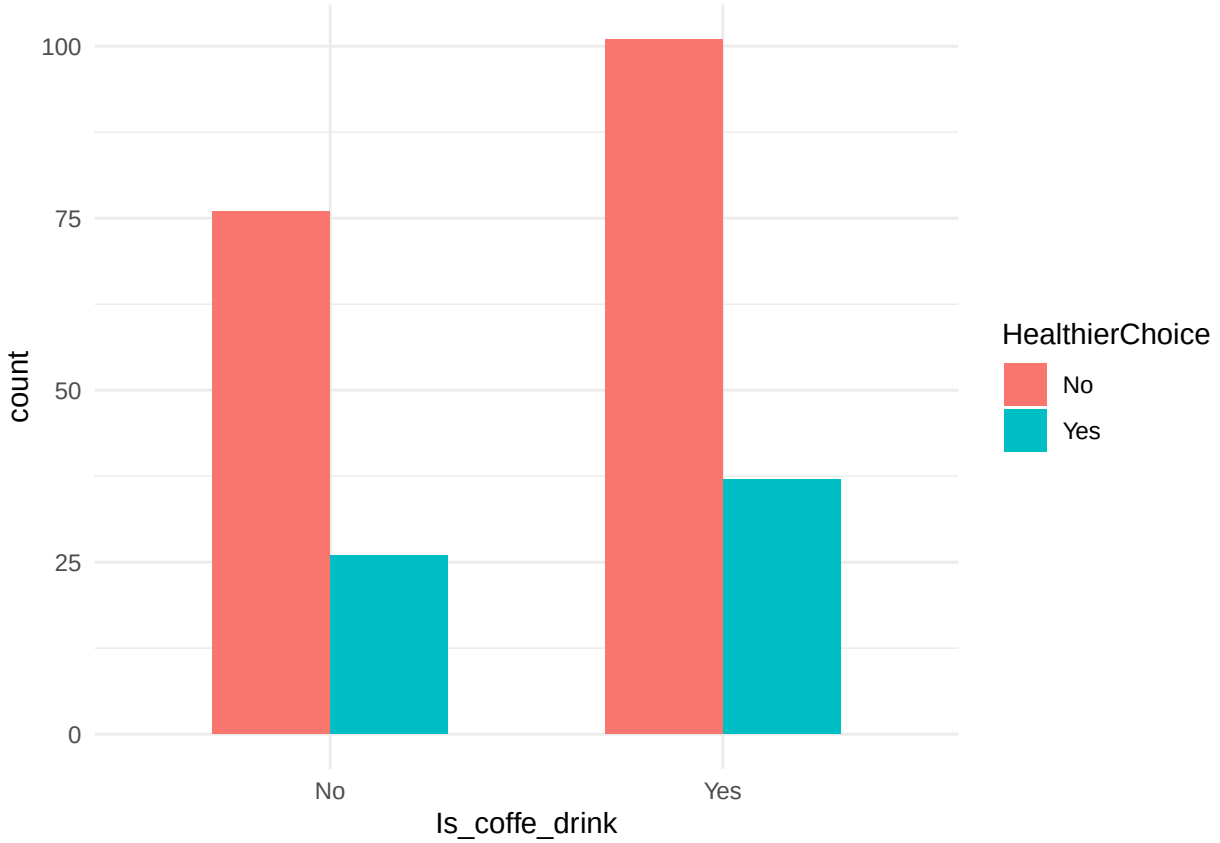
It looks fairly clear from the above plot that Milk is associated with how healthy a drink is. To get an even better view of that, instead of counts (absolute values), we will use proportions, by just changing the value of the `position` parameter of the `geom_bar` function:

```
ggplot(sb_drinks,
       aes(x = Milk, fill = HealthierChoice)) +
  geom_bar(position = "fill", width = 0.6) + # note that the position parameter now has value "fill"
  labs(x= "\nMilk", y = "Proportion\n") +
  theme_minimal()
```



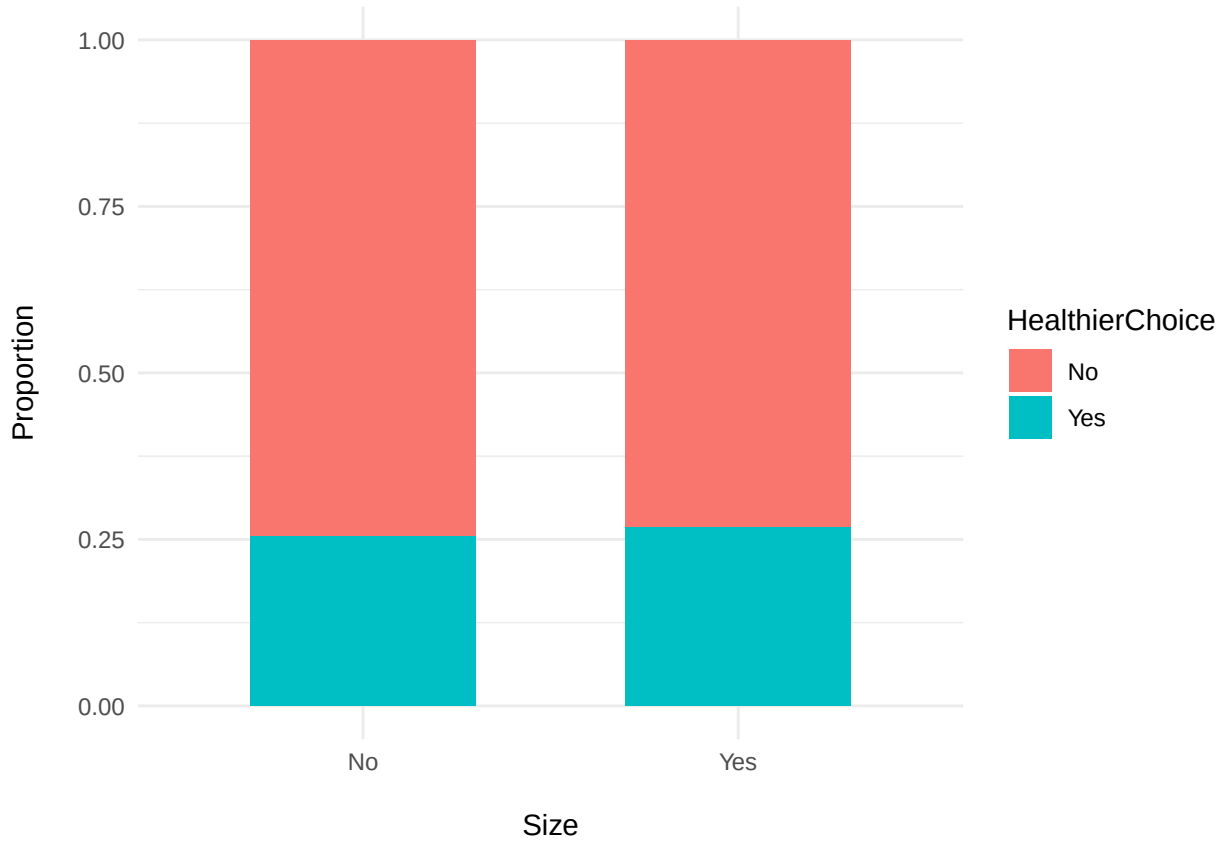
Let's now check if knowing if a drink is a coffee drink helps in determining if it is a healthier choice:

```
ggplot(sb_drinks,
       aes(x = Is_coffe_drink, fill = HealthierChoice)) +
  geom_bar(position = "dodge", width = 0.6) +
  theme_minimal()
```



Not very clear from the plot above as there are many more coffee drinks than other kinds of drinks. So, we will use proportions instead:

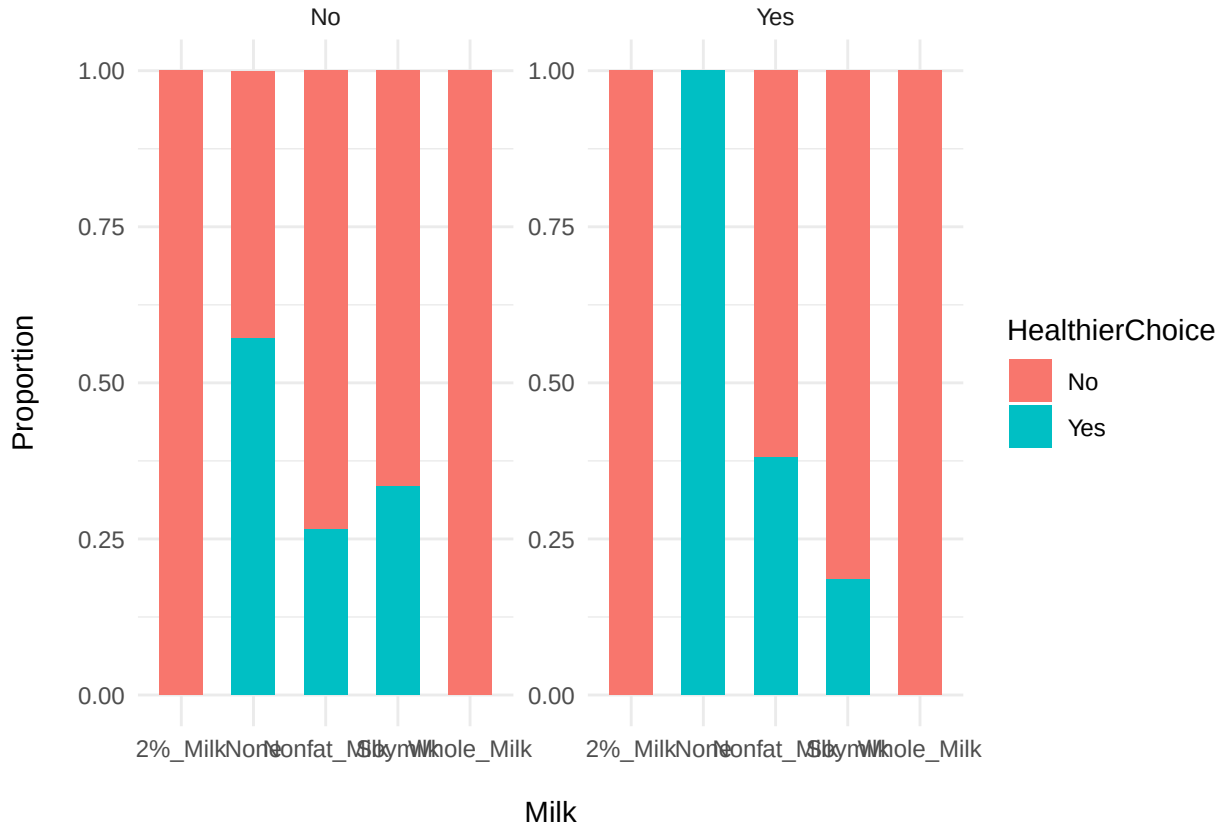
```
ggplot(sb_drinks,
       aes(x = Is_coffe_drink, fill = HealthierChoice)) +
  geom_bar(position = "fill", width = 0.6) +
  labs(x= "\nSize", y = "Proportion\n") +
  theme_minimal()
```



Very small difference, suggesting that the `Is_coffe_drink` variable would be a poor predictor of the outcome variable

We can also examine interaction of two variables

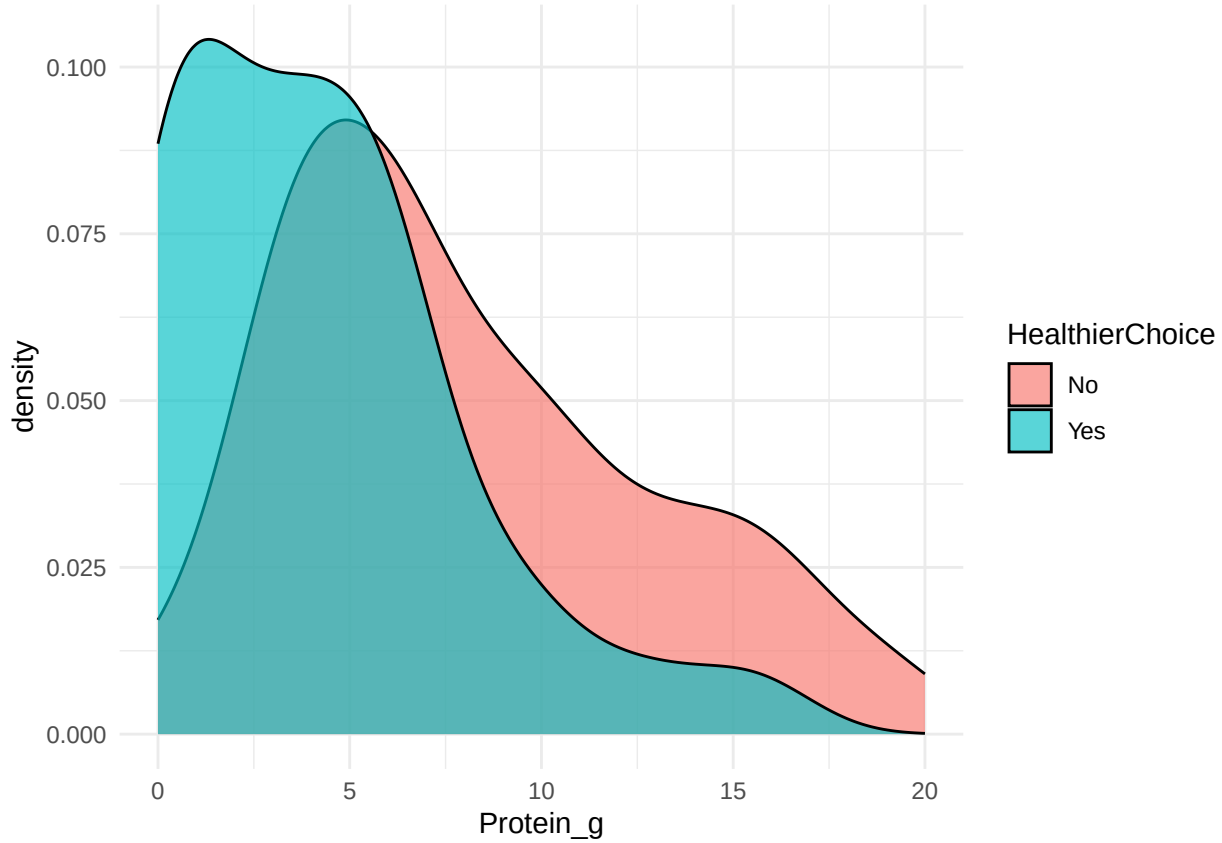
```
ggplot(sb_drinks,
       aes(x = Milk, fill = HealthierChoice)) +
  geom_bar(position = "fill", width = 0.6) +
  labs(x= "\nMilk", y = "Proportion\n") +
  facet_wrap(~Is_coffe_drink, scales = "free") +
  theme_minimal()
```



We can observe that coffee drinks with no milk or with nonfat milk are more likely to be healthier choice than non-coffee drinks with no milk, or non-fat milk. The opposite is true for soy milk. So, if we are working with a classification algorithm that can leverage interaction of variables (e.g., logistic regression), using the interaction of these two variables can bring some additional signal to the model. This also shows that even though `Is_coffe_drink` on its own is not relevant for prediction, in interaction with `Milk`, it can be relevant.

For continuous variables, we use histogram or probability density function to inspect the relation between an input variable and the outcome. In such plots, we examine the presence or extent of difference in the distribution of the variable's values for the different values of the outcome variable. For example, we can examine how relevant the quantity of proteins is for distinguishing between less and more healthy drink choices:

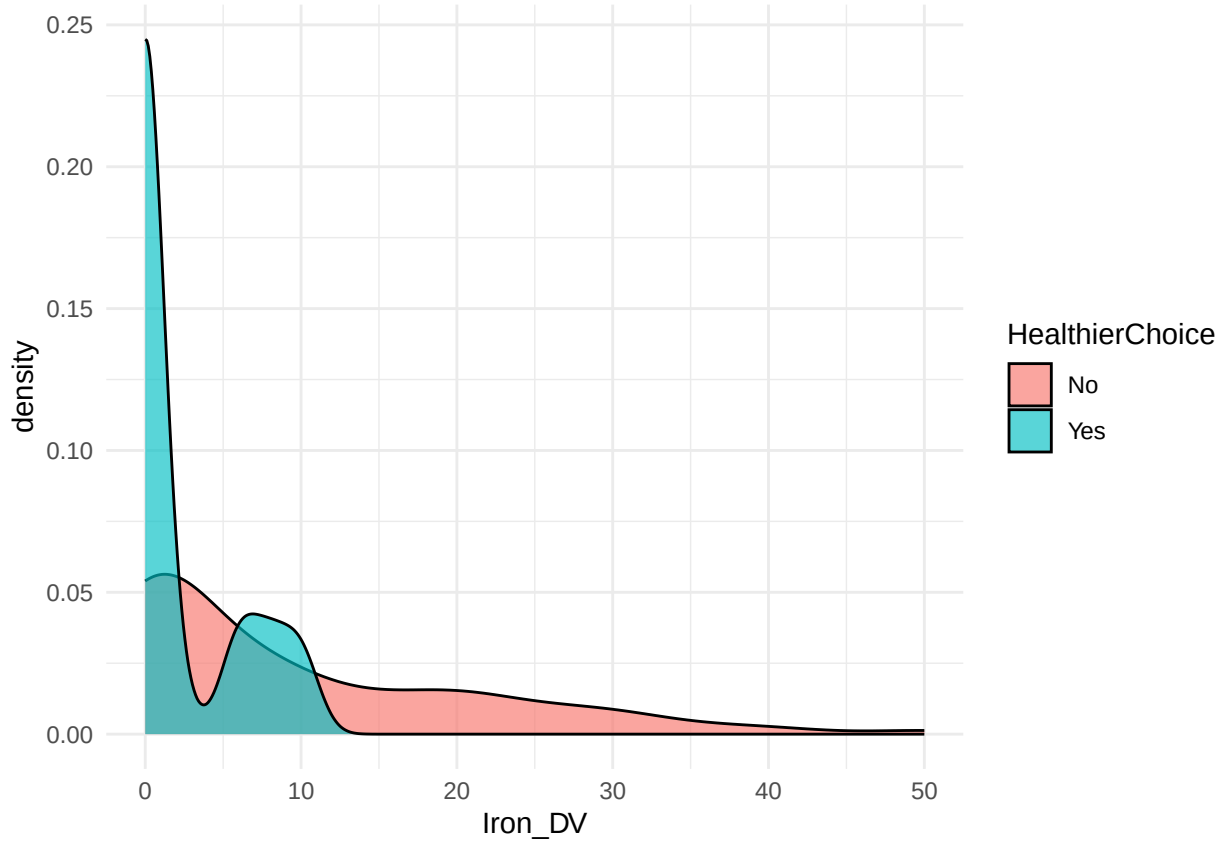
```
ggplot(sb_drinks,
       aes(x = Protein_g, fill = HealthierChoice)) +
  geom_density(alpha=0.65) +
  theme_minimal()
```



We can observe a clear difference in the distributions: healthier choices tend to have lower quantity of proteins. So, the `Protein_g` variable is expected to be a relevant predictor.

Let's take one more more - for example, `Iron_DV`

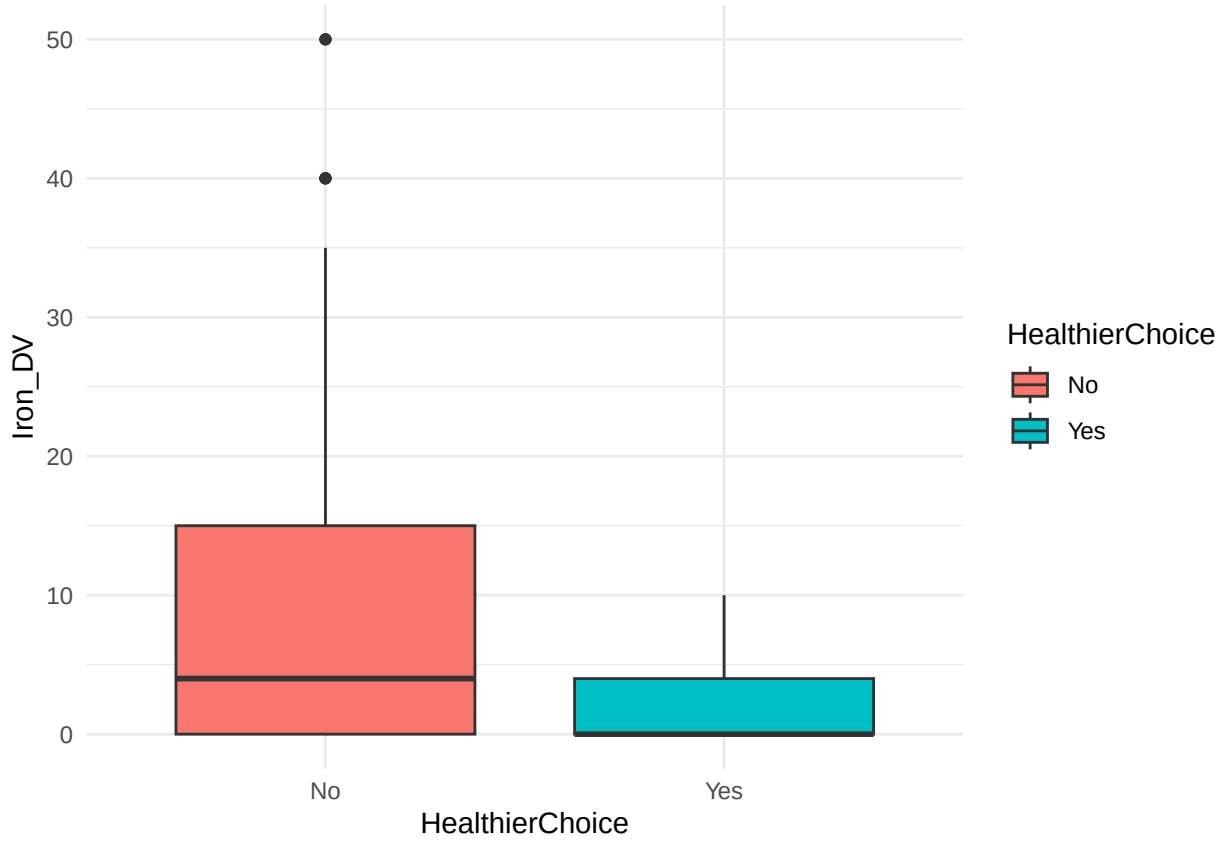
```
ggplot(sb_drinks,
       aes(x = Iron_DV, fill = HealthierChoice)) +
  geom_density(alpha=0.65) +
  theme_minimal()
```



Again, healthier choices tend to have lower values of iron and very different distribution than less healthy choices, suggesting that `Iron_DV` is also expected to be a good predictor of the outcome variable.

If it's easier for you, you may also use boxplots:

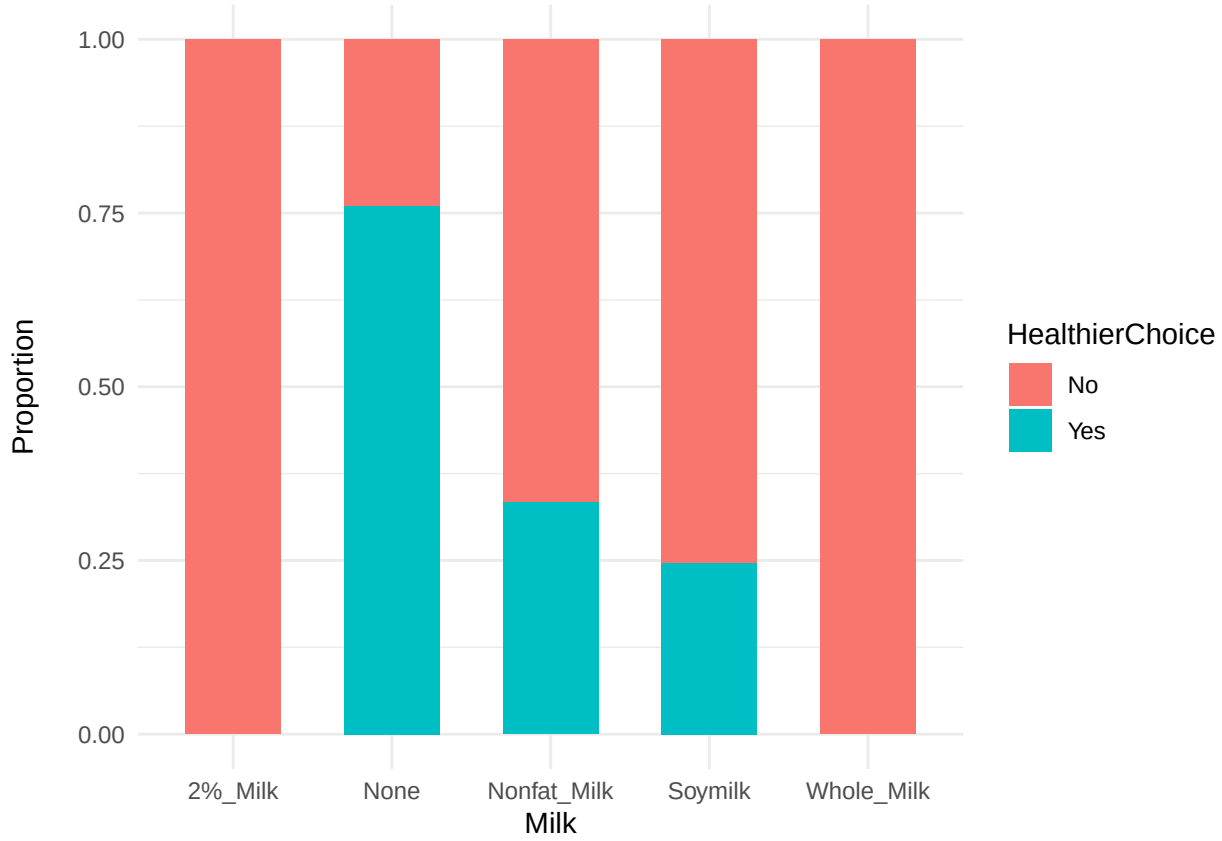
```
ggplot(sb_drinks,
       aes(x = HealthierChoice, y = Iron_DV, fill = HealthierChoice)) +
  geom_boxplot() +
  theme_minimal()
```



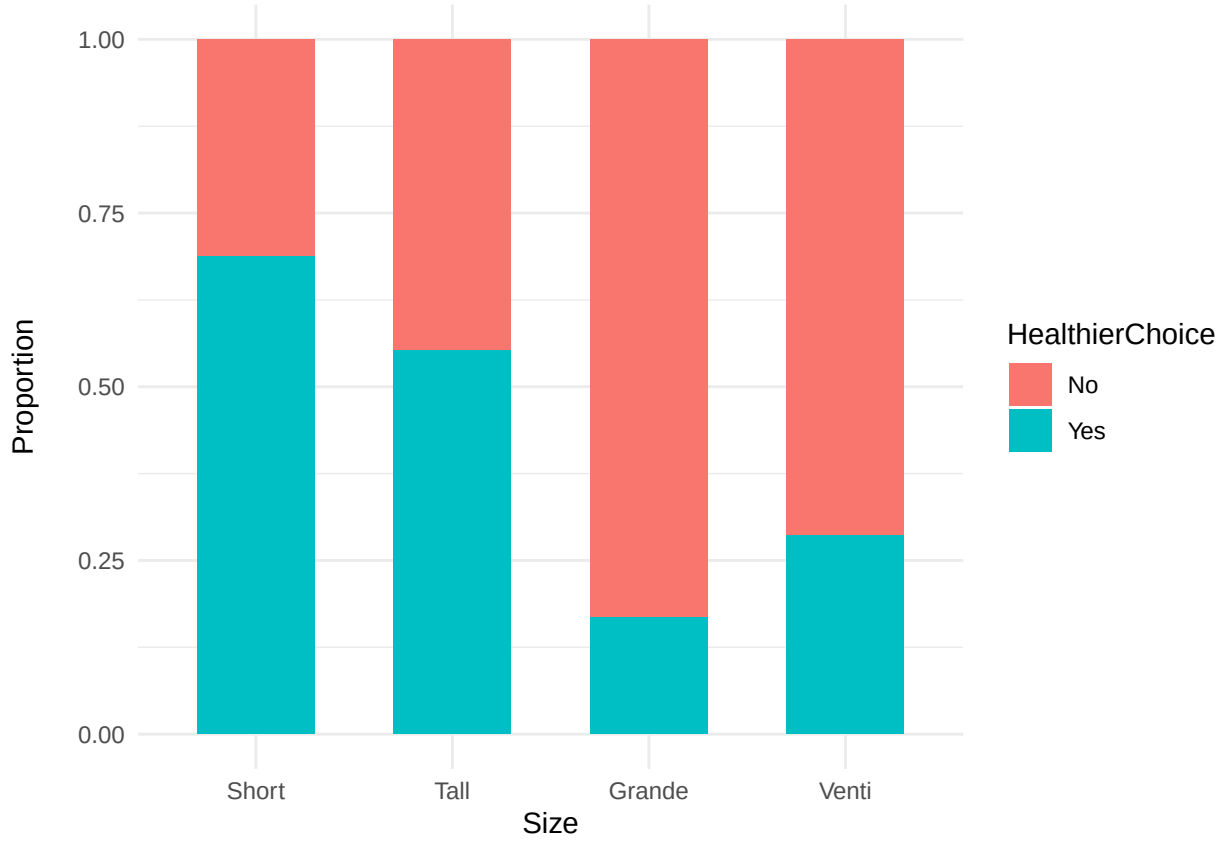
Finally, here is how you can quickly create plots for all factor and continuous variables:

```
# take the names of the factor variables
factor_var_names = colnames(sb_drinks)[c(3,4,17)]

# function for plotting a variable with the given name
# the notation .data[[var_name]] in the code below allow us to access column from the 'current' data fr
plot_factor_var <- function(var_name) {
  ggplot(sb_drinks,
       aes(x = .data[[var_name]], fill = HealthierChoice)) +
  geom_bar(position = "fill", width = 0.6) +
  labs(y = "Proportion\n") +
  theme_minimal()
}

# applying the plotting function to all factor variables
lapply(factor_var_names, plot_factor_var)
```

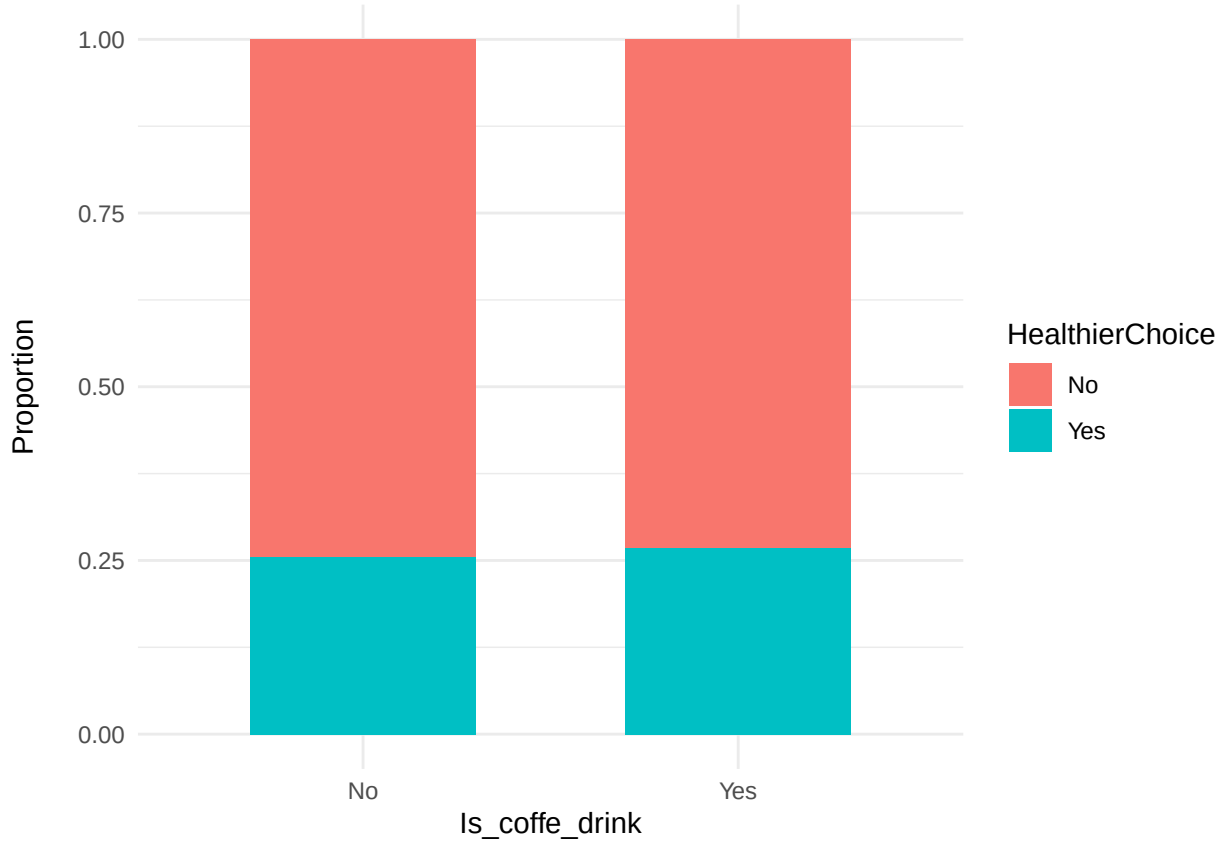
```
## [[1]]
```


1.00
0.75
0.50
0.25
0.00
Proportion
2%_Milk
None
Nonfat_Milk
Soymilk
Whole_Milk
Milk
HealthierChoice
No
Yes

```
##
## [[2]]
```


1.00
0.75
0.50
0.25
0.00
Proportion
Short
Tall
Grande
Venti
Size
HealthierChoice
No
Yes

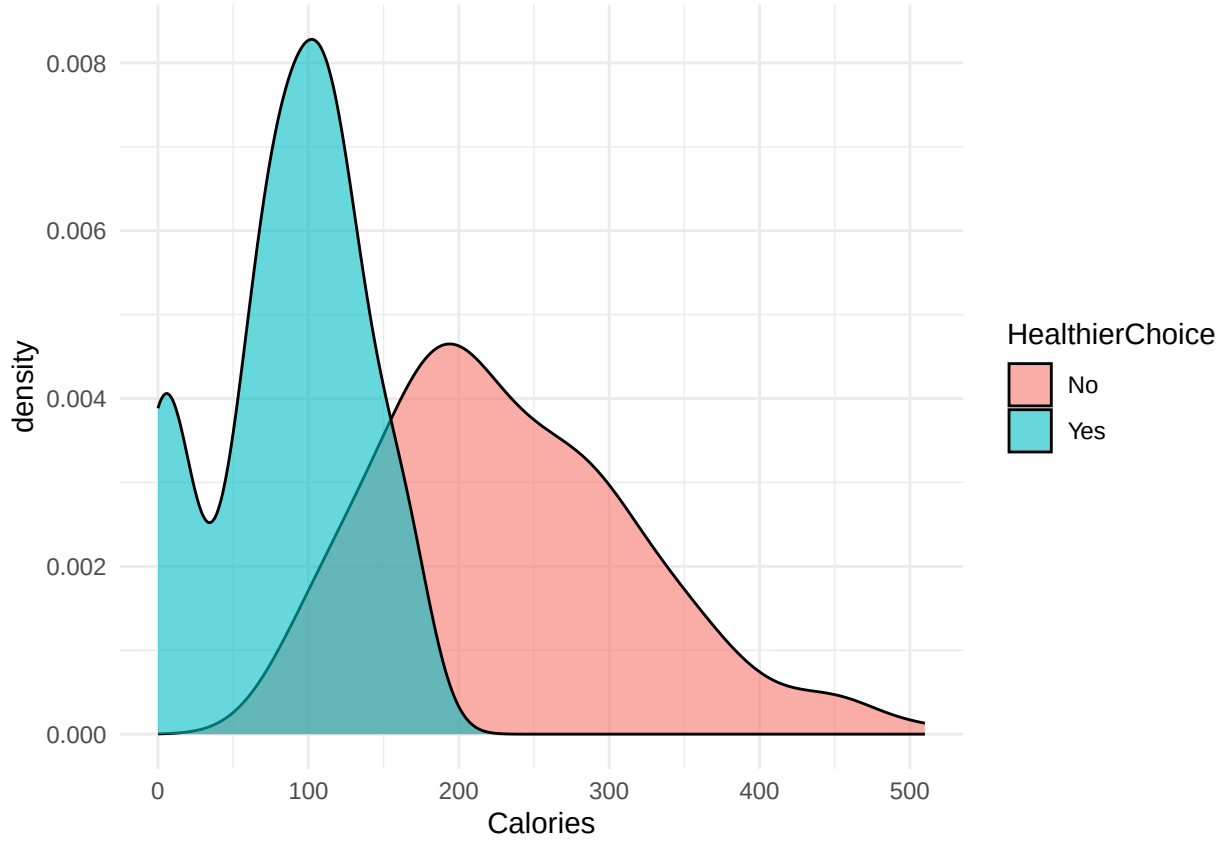
```
##
## [[3]]
```



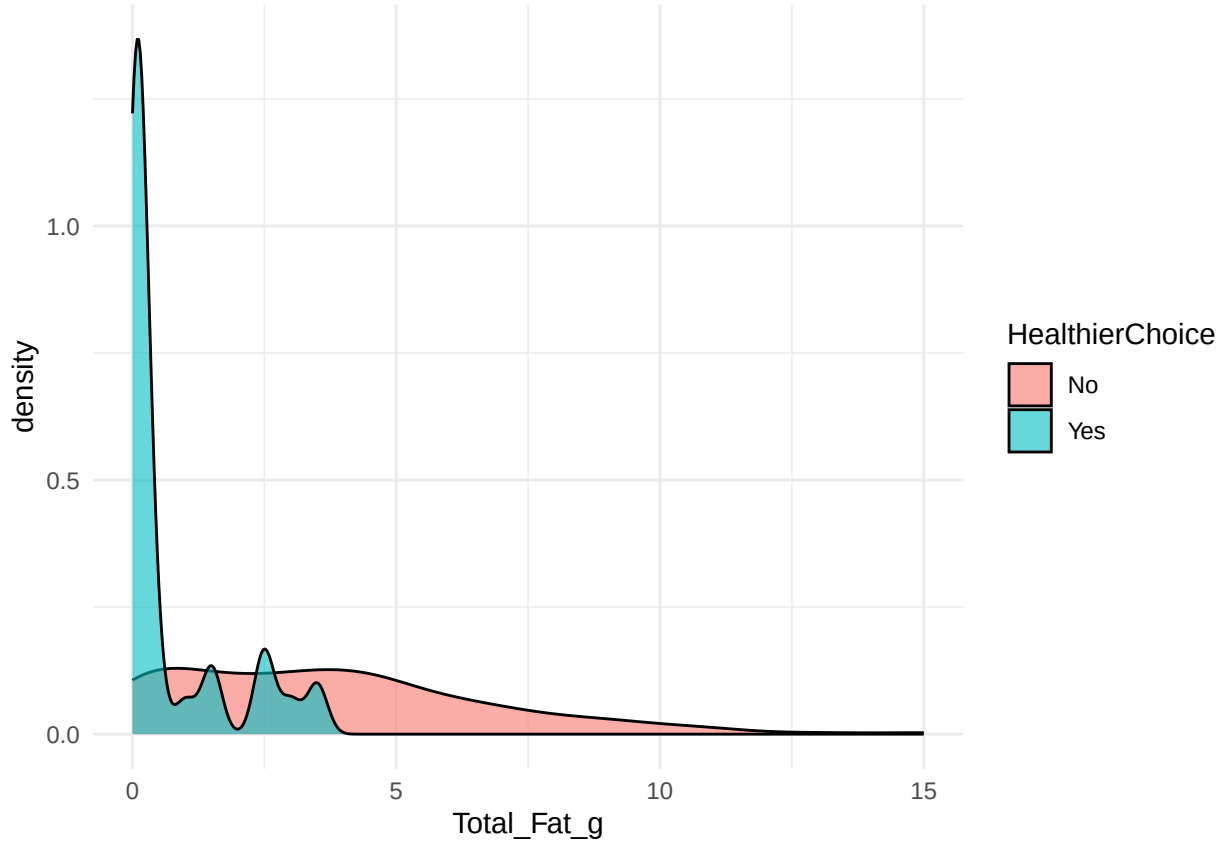
Similar for numeric variables

```
# take the names of the factor variables
num_var_names = colnames(sb_drinks)[c(5:16)]

# function for plotting a variable with the given name
plot_num_var <- function(var_name) {
  ggplot(sb_drinks,
       aes(x = .data[[var_name]], fill = HealthierChoice)) +
  geom_density(alpha = 0.6) +
  theme_minimal()
}

# applying the plotting function to all factor variables
lapply(num_var_names, plot_num_var)
```

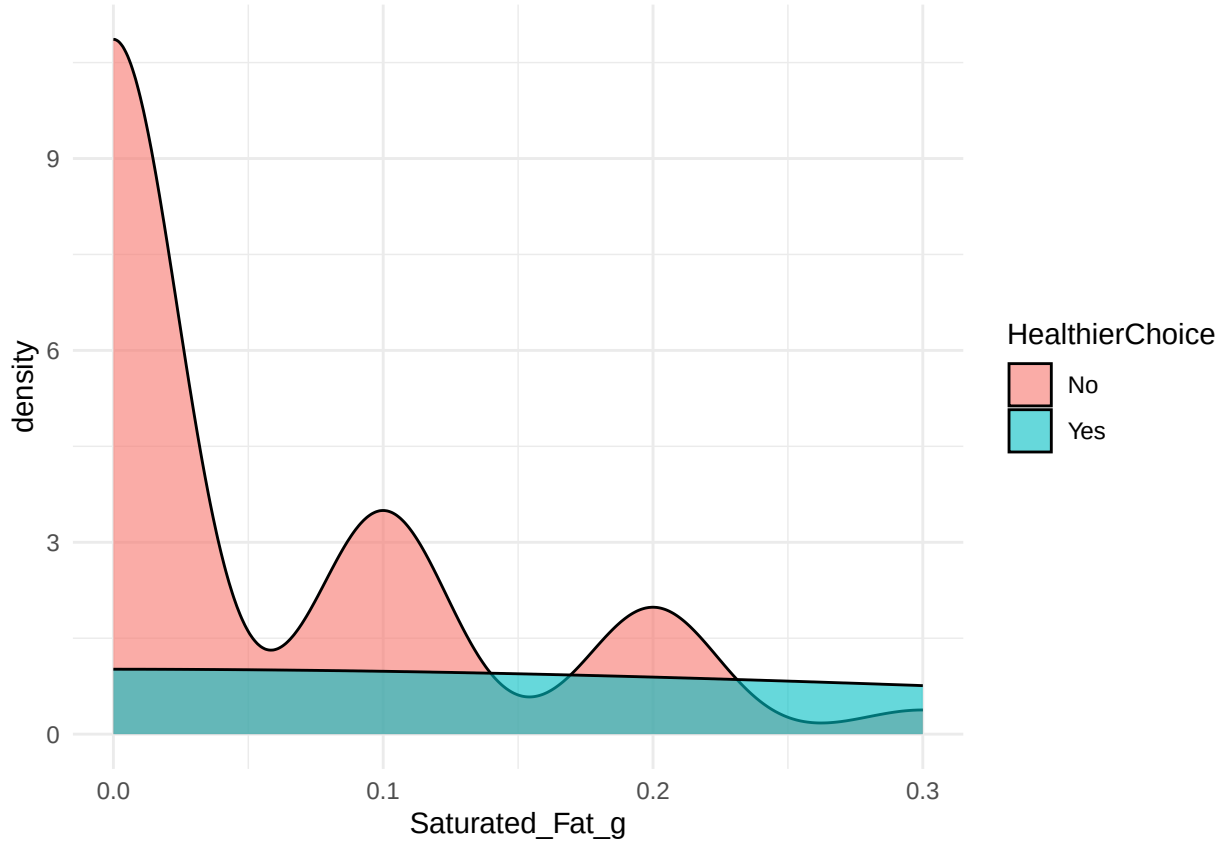
```
## [[1]]
```


0.008
0.006
0.004
0.002
0.000
density
0
100
200
300
400
500
Calories
HealthierChoice
No
Yes

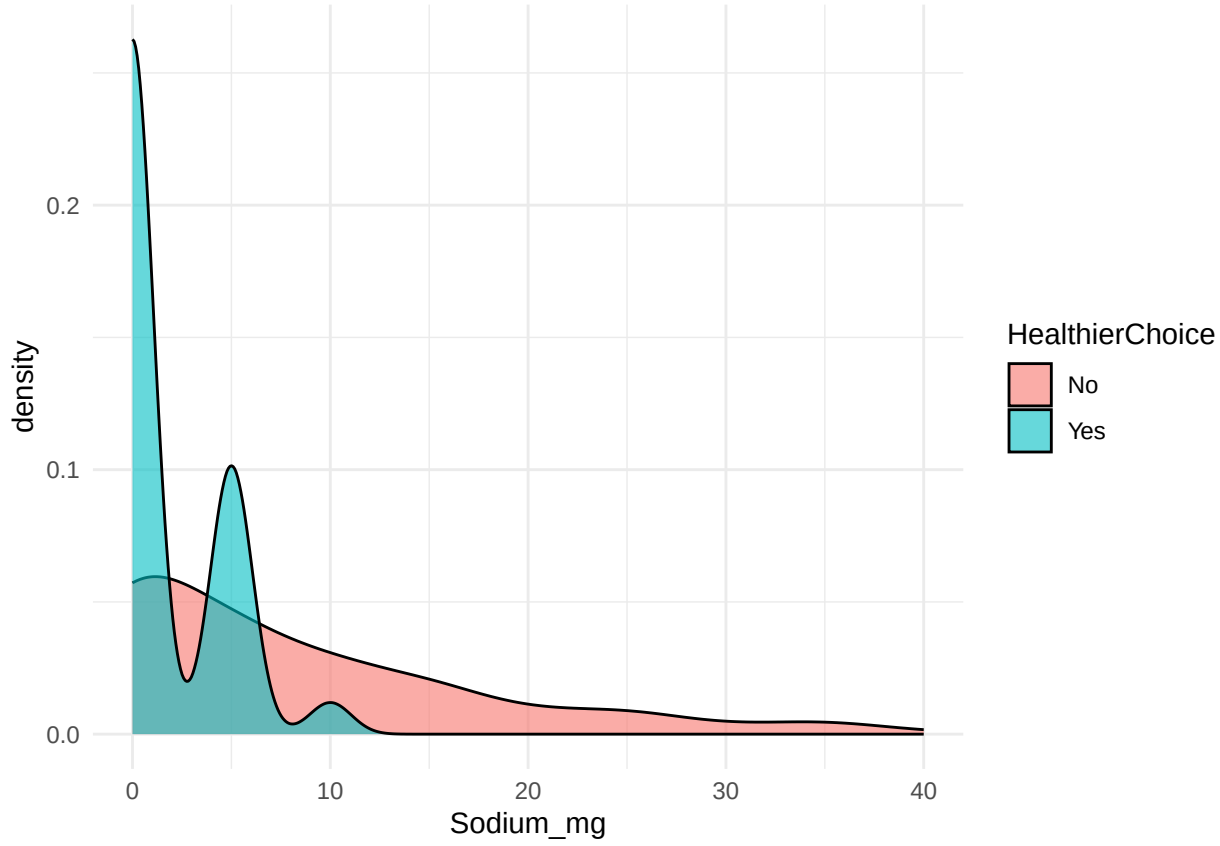
```
##
## [[2]]
```



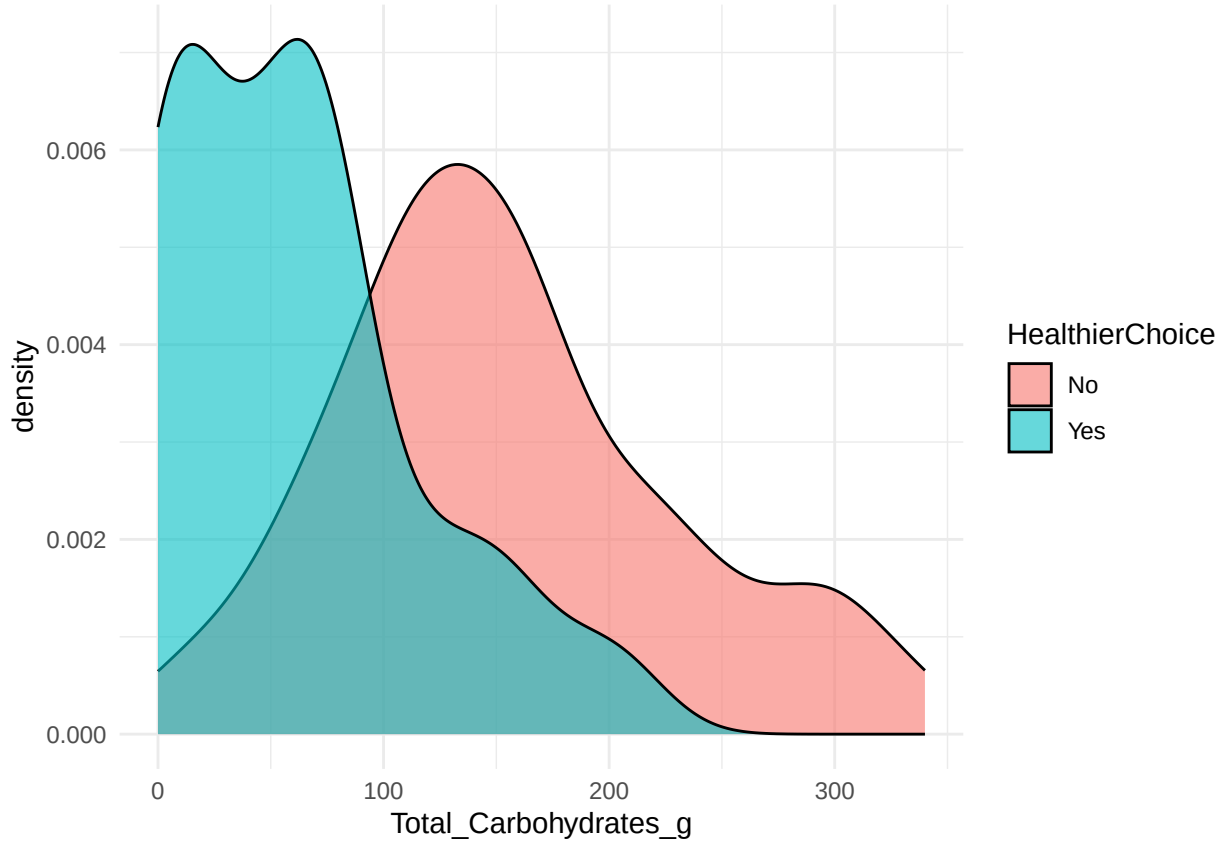

```
##
## [[3]]
```


density
9
6
3
0
0.0
0.1
0.2
0.3
Saturated_Fat_g
HealthierChoice
No
Yes

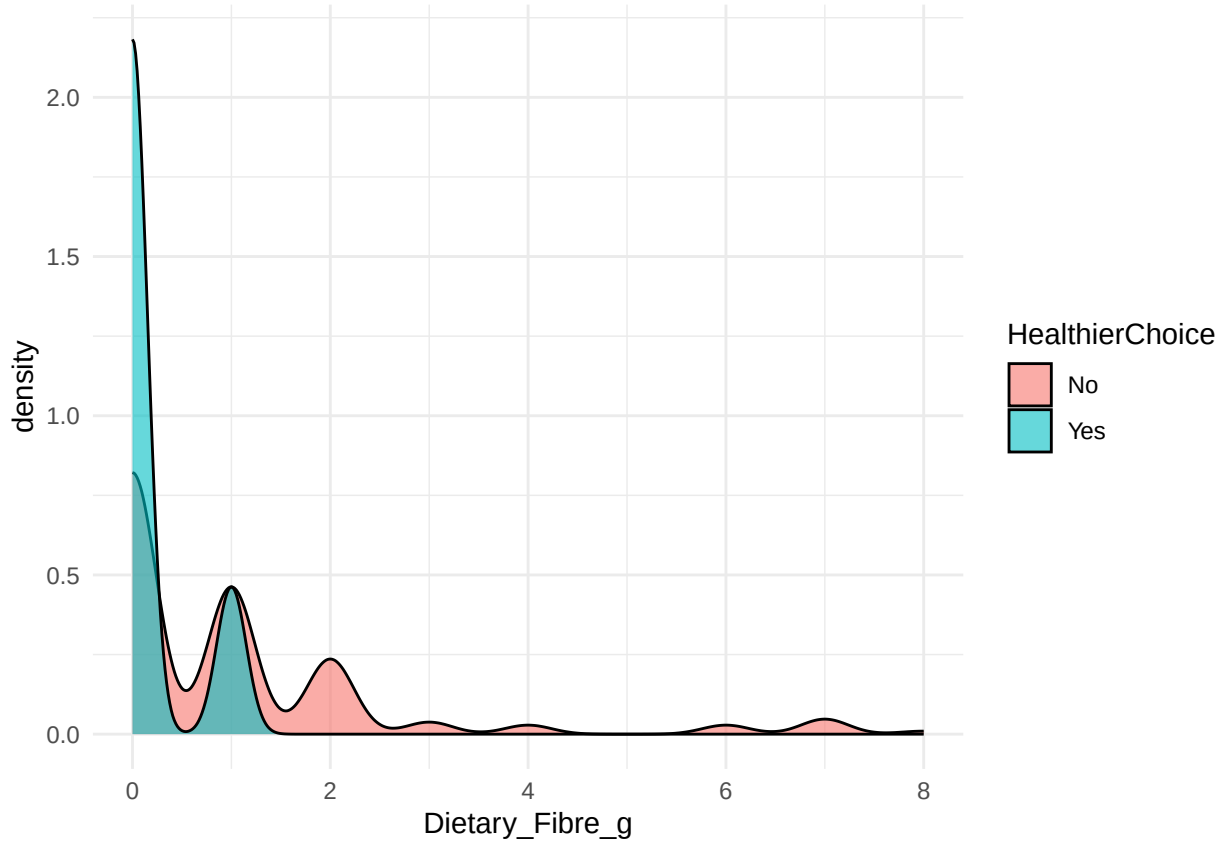
```
##
## [[4]]
```


density
0.2
0.1
0.0
0
10
20
30
40
Sodium_mg
HealthierChoice
No
Yes

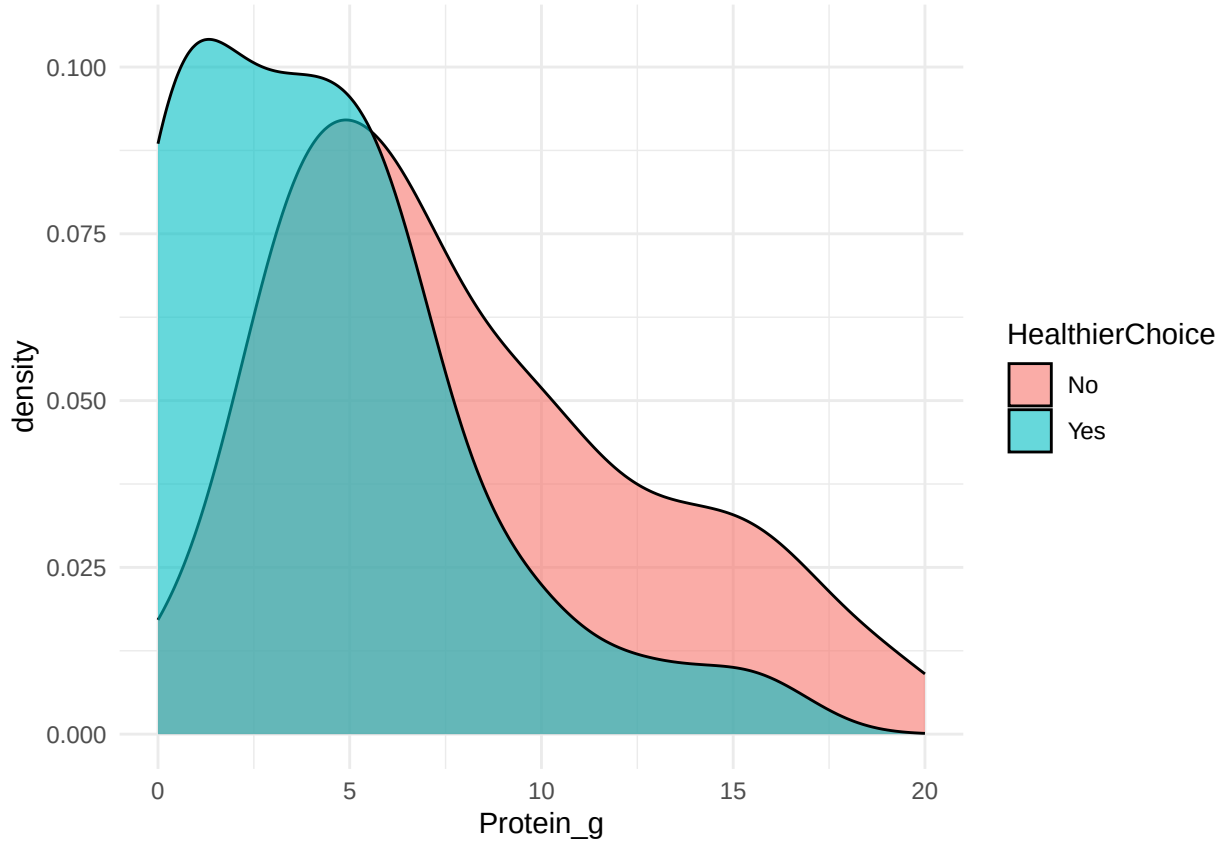
```
##
## [[5]]
```


0.006
0.004
0.002
0.000
density
0
100
200
300
Total_Carbohydrates_g
HealthierChoice
No
Yes

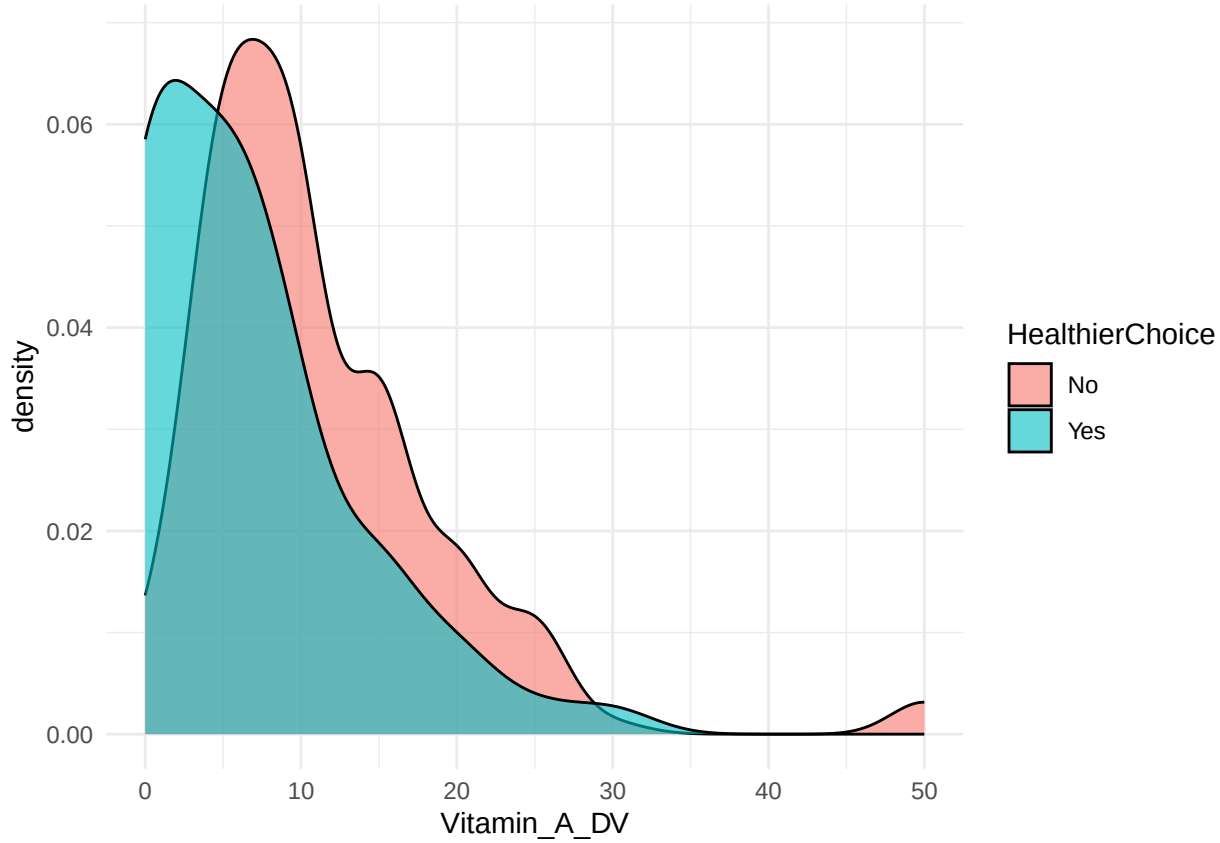
```
##
## [[6]]
```


2.0
1.5
1.0
0.5
0.0
density
0
2
4
6
8
Dietary_Fibre_g
HealthierChoice
No
Yes

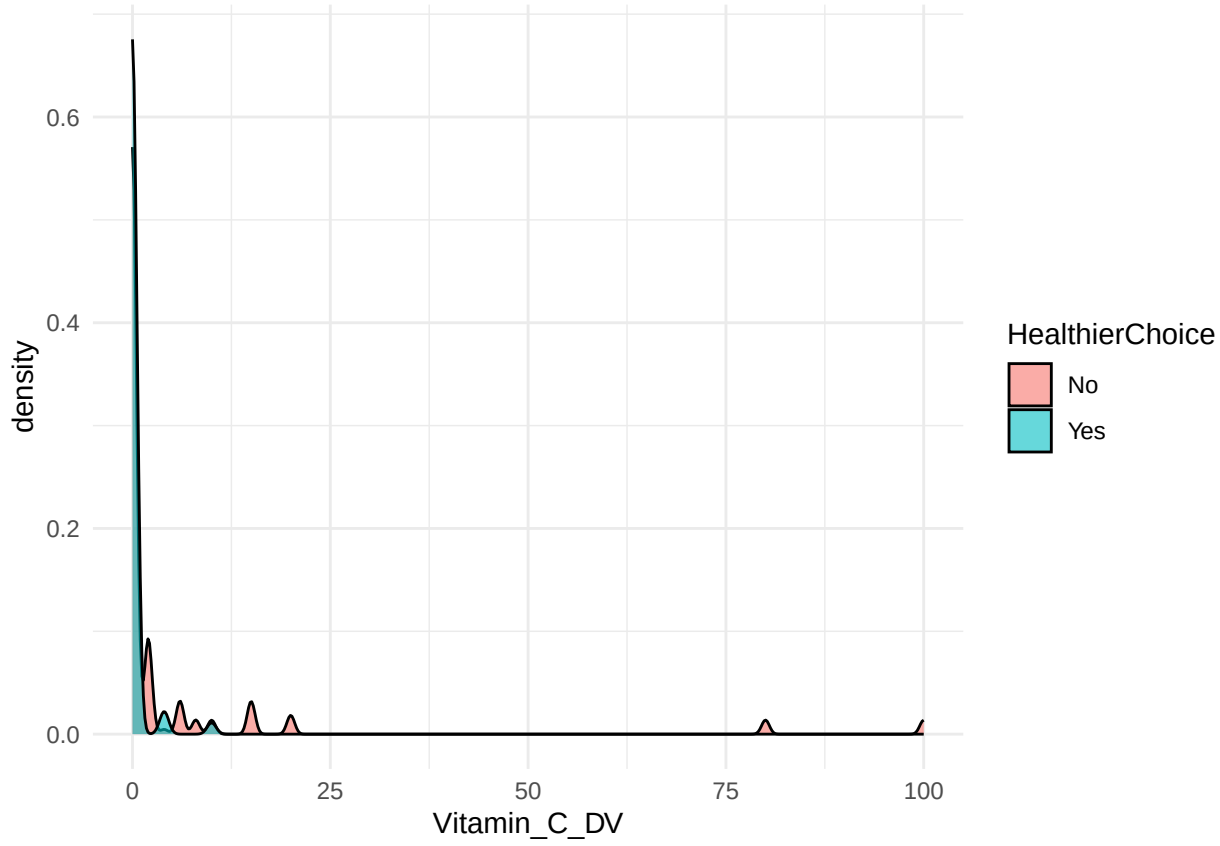
```
##
## [[7]]
```


0.100
0.075
0.050
0.025
0.000
density
0
5
10
15
20
Protein_g
HealthierChoice
No
Yes

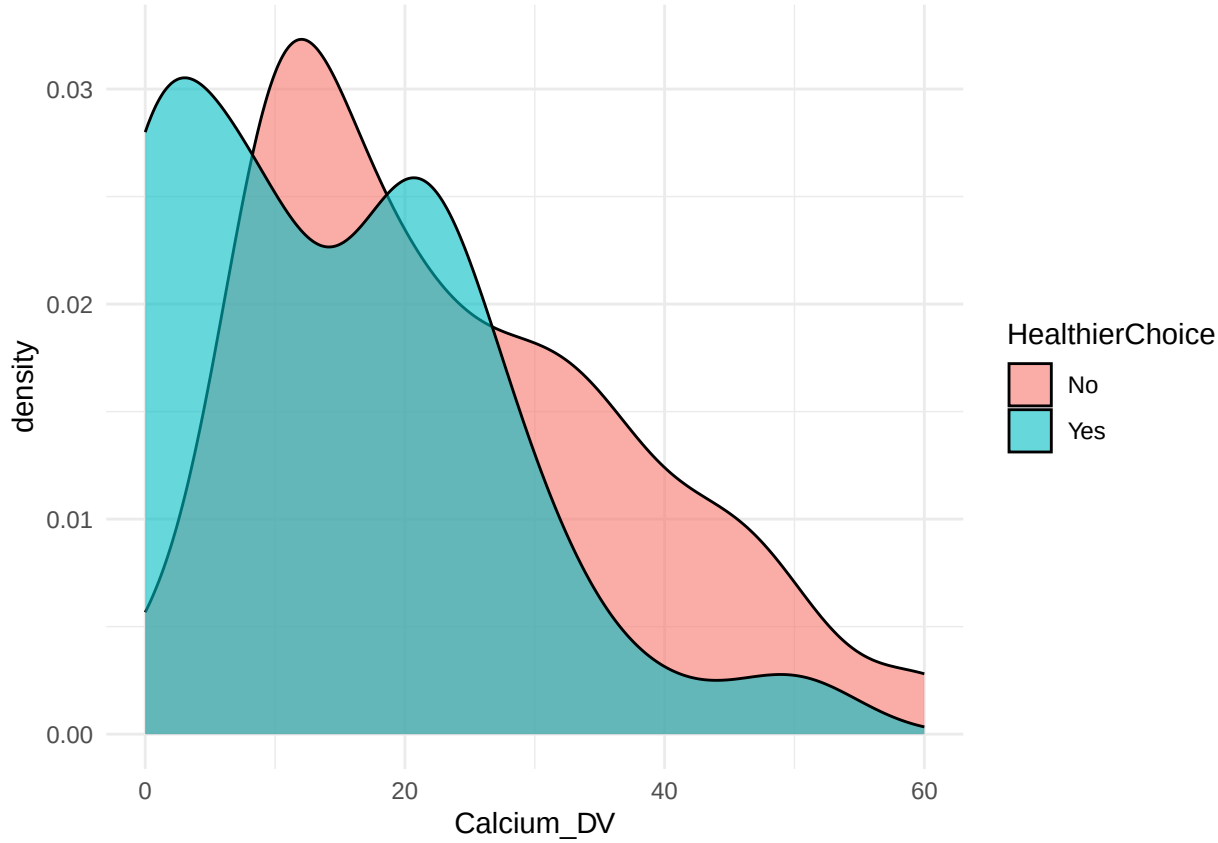
```
##
## [[8]]
```


0.06
0.04
0.02
0.00
density
0
10
20
30
40
50
Vitamin_A_DV
HealthierChoice
No
Yes

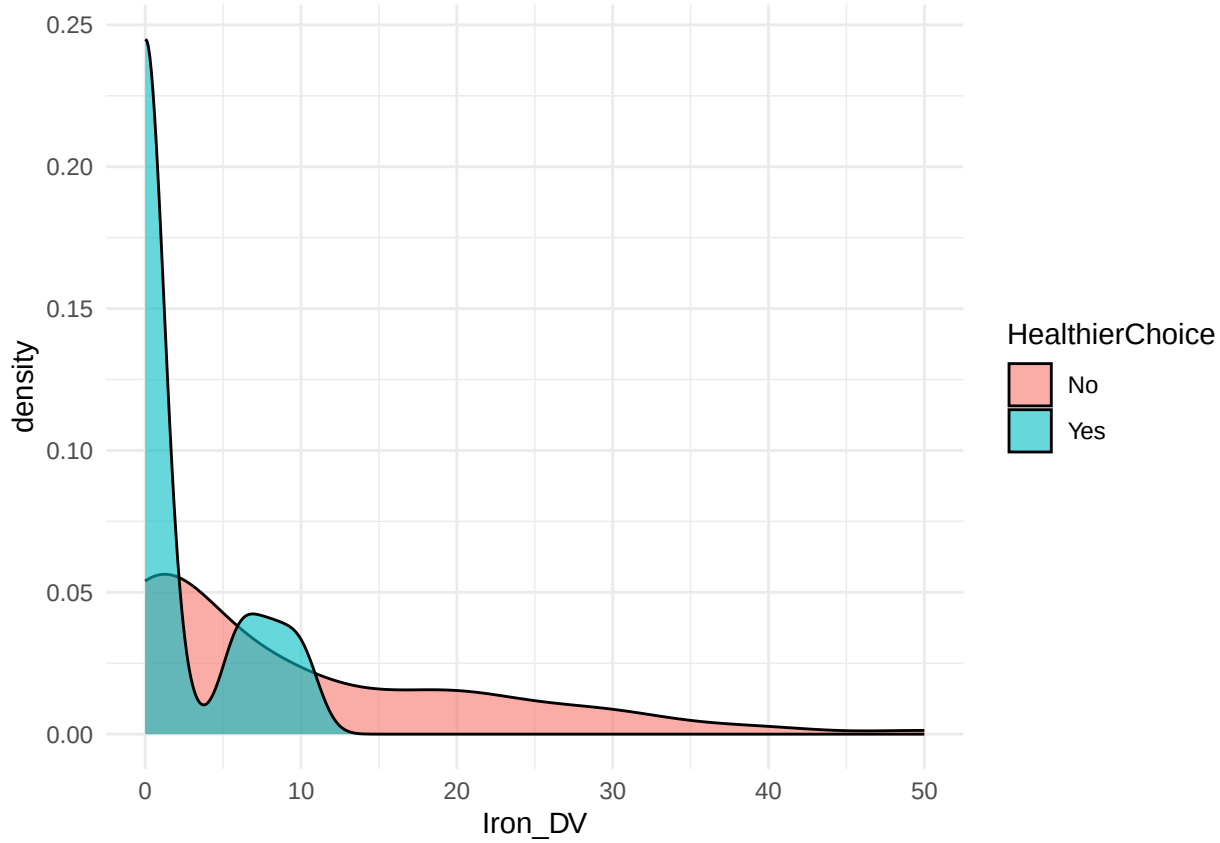
```
##
## [[9]]
```


0.6
0.4
0.2
0.0
density
0
25
50
75
100
Vitamin_C_DV
HealthierChoice
No
Yes

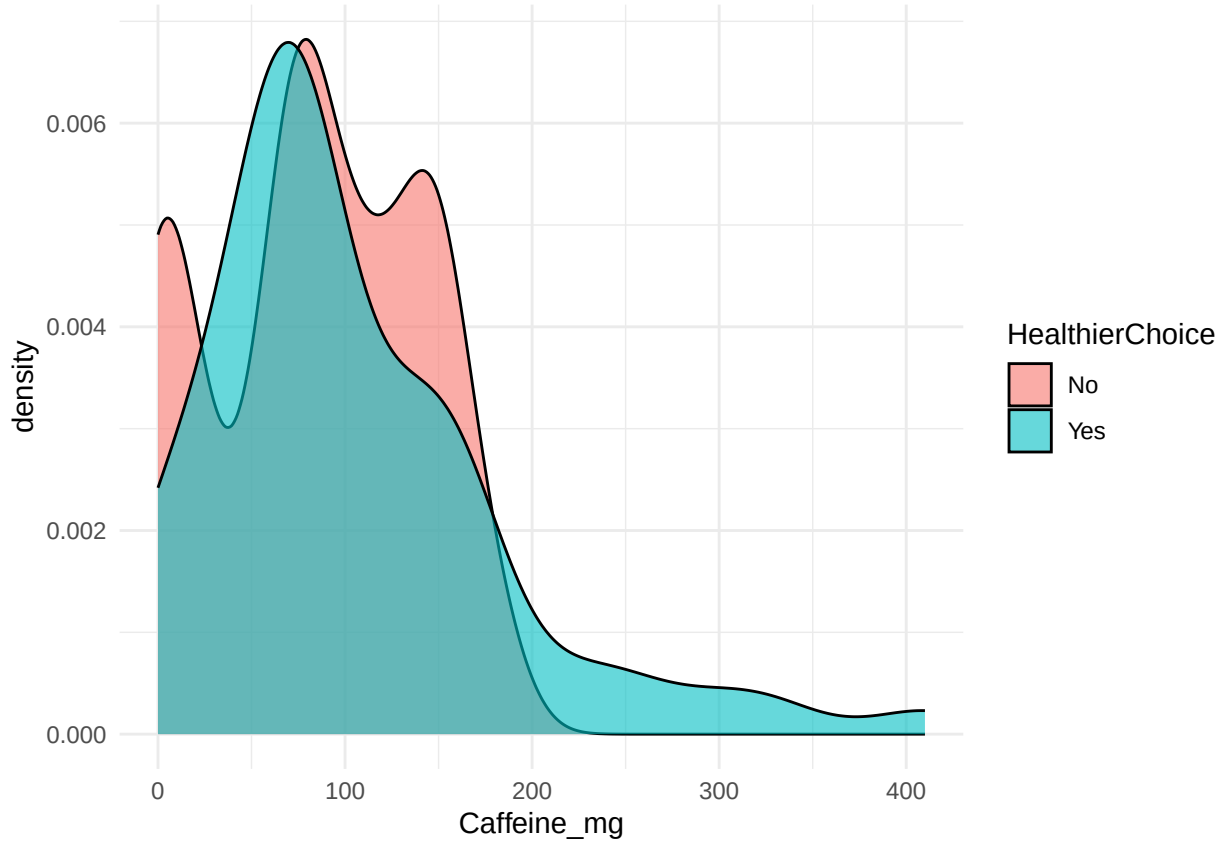
```
##
## [[10]]
```


0.03
0.02
0.01
0.00
density
0
20
40
60
Calcium_DV
HealthierChoice
No
Yes

```
##
## [[11]]
```


0.25
0.20
0.15
0.10
0.05
0.00
density
0
10
20
30
40
50
Iron_DV
HealthierChoice
No
Yes

```
##
## [[12]]
```


0.006
0.004
0.002
0.000
density
0
100
200
300
400
Caffeine_mg
HealthierChoice
No
Yes