Datacamp ggplot2 - Part 2 CHIS Exercise

Chinpei Tang

In this chapter we're going to continuously build on our plotting functions and understanding to produce a mosaic plot (aka Merimeko plot). This is a visual representation of a contingency table, comparing two categorical variables. Essentially, our question is which groups are over or under represented in our dataset. To visualize this we'll color groups according to their pearson residuals from a chi-squared test. At the end of it all we'll wrap up our script into a flexible function so that we can look at other variables.

We'll familiarize ourselves with a small number of variables from the 2009 CHIS adult-response dataset (as opposed to children). The data can be found at: http://healthpolicy.ucla.edu/chis/data/Pages/GetCHISData.aspx. The Datacamp has selected the following variables to explore:

- RBMI: BMI Category description
- BMI_P: BMI value
- RACEHPR2: race
- SRSEX: sex
- SRAGE_P: age
- MARIT2: Marital status
- AB1: General Health Condition
- ASTCUR: Current Asthma Status
- AB51: Type I or Type II Diabetes
- POVLL: Poverty level

Since Datacamp loads the data in the background on the tutorial portal, in order to do it externally, the data is downloaded and imported with the following commands. It is saved to the RData file. This codes are not executed here since it takes some times to run.

```
if(F){
  library(foreign)
  adult <- read.spss("ADULT.sav", to.data.frame = T)
  save(adult, file = "CHIS_adult.RData")
}
```

We load some of the other libraries. Since the data imported doesn't look exactly the same from Datacamp, some work were done to mimic the data from Datacamp in order to do the exercises.

```
library(ggplot2)
library(reshape2) # for melt() function
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(ggthemes)
library(car)
```

```
##
## Attaching package: 'car'

## The following object is masked from 'package:dplyr':
##
##     recode
```

```r
# Load the saved RData file
load(file = "C:/Users/Chinpei/Documents/R/CHIS/chis09_adult_spss/chis09_adult_spss/CHIS_adult.RData")

# Some manipulations to make data "looks like" form Datacamp
adult <- adult[c("RBMI", "BMI_P", "RACEHPR2", "SRSEX", "SRAGE_P", "MARIT2", "AB1", "ASTCUR", "AB51", "P
adult$RACEHPR2 <- factor(adult$RACEHPR2, levels = c("LATINO", "ASIAN", "AFRICAN AMERICAN", "WHITE"), la
adult$RBMI <- factor(adult$RBMI, levels = c("UNDERWEIGHT 0-18.49", "NORMAL 18.5-24.99", "OVERWEIGHT 25.
```

We'll filter our dataset to plot a more reliable subset (we'll still retain over 95% of the data).

Before we get into mosaic plots it's worthwhile exploring the dataset using simple distribution plots - i.e. histograms.

Exploring Data

Use the typical commands for exploring the structure of adult to get familiar with the variables: summary() and str().

```r
# Explore the dataset with summary and str
summary(adult)
```

```
##   RBMI          BMI_P        RACEHPR2       SRSEX          SRAGE_P
##   1: 1068   Min.   :12.65   1    : 5753   MALE  :19428   Min.   :18.00
##   2:19831   1st Qu.:22.82   4    : 4874   FEMALE:28186   1st Qu.:44.00
##   3:16152   Median :25.74   5    : 1950                  Median :57.00
##   4:10563   Mean   :26.73   6    :31769                  Mean   :55.67
##             3rd Qu.:29.43   NA's: 3268                   3rd Qu.:69.00
##             Max.   :93.72                                Max.   :85.00
##
##                 MARIT2                     AB1
##   NOT ASCERTAINED  :    0   VERY GOOD      :15490
##   MARRIED          :24936   GOOD           :13658
##   LIVING W/ PARTNER: 2336   EXCELLENT      : 9114
##   WID/SEP/DIV      :13360   FAIR           : 6878
##   NEVER MARRIED    : 6982   POOR           : 2474
##                             NOT ASCERTAINED:    0
##                             (Other)        :    0
##                 ASTCUR                     AB51
##   NOT ASCERTAINED  :    0   NOT ASCERTAINED:    0
##   CURRENT ASTHMA   : 4120   DON'T KNOW     :    0
##   NO CURRENT ASTHMA:43494   REFUSED        :    0
##                              INAPPLICABLE   :42796
##                              TYPE I         :  512
##                              TYPE 2         : 4206
##                              ANOTHER TYPE   :  100
##                   POVLL
##   0-99% FPL          : 5764
##   100-199% FPL       : 8026
##   200-299% FPL       : 6532
##   300% FPL AND ABOVE:27292
##
##
##
```

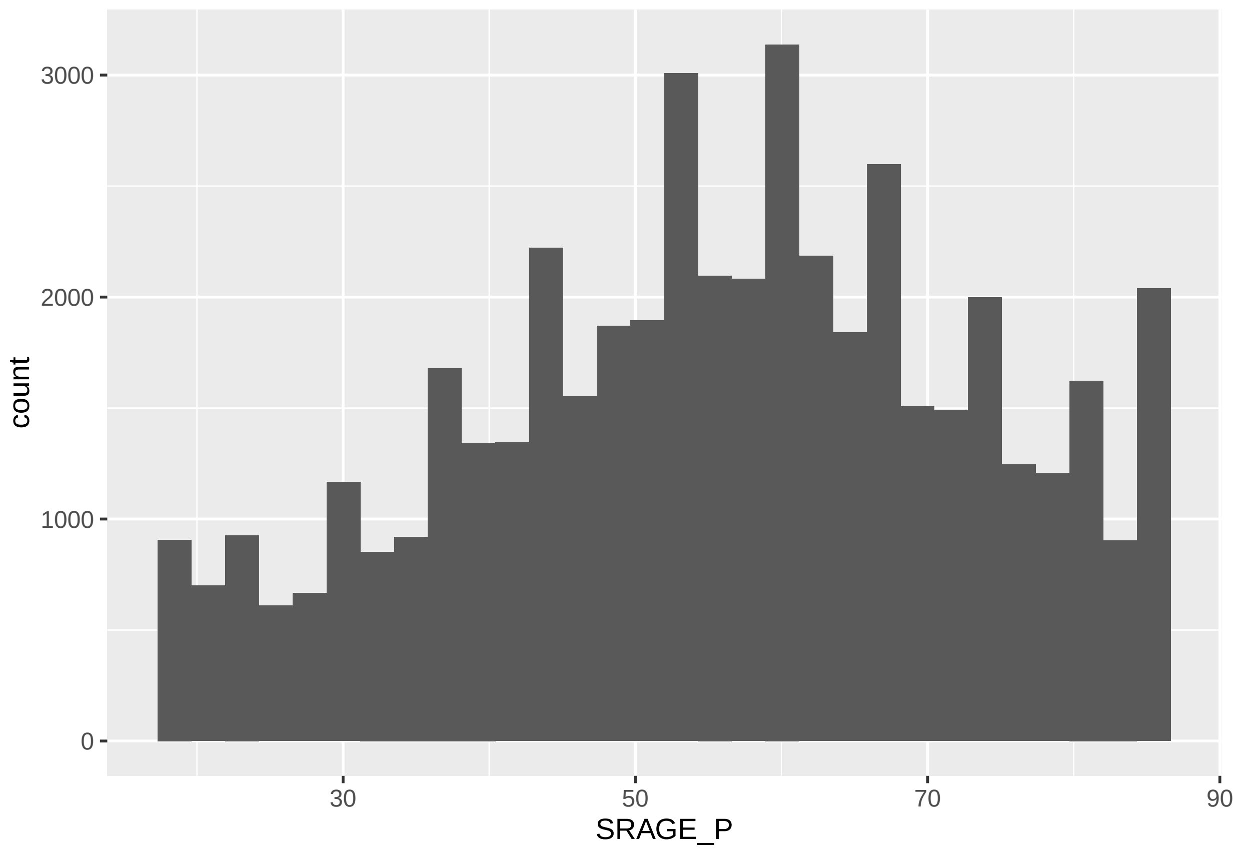
```
str(adult)
```

```
## 'data.frame':    47614 obs. of  10 variables:
##  $ RBMI    : Factor w/ 4 levels "1","2","3","4": 3 3 3 2 3 4 2 3 2 3 ...
##  $ BMI_P   : atomic  28.9 26.1 25.1 25 25.1 ...
##   ..- attr(*, "value.labels")= Named num -9
##   .. ..- attr(*, "names")= chr "NOT ASCERTAINED"
##  $ RACEHPR2: Factor w/ 4 levels "1","4","5","6": 4 4 4 4 4 4 NA 4 4 4 ...
##  $ SRSEX   : Factor w/ 2 levels "MALE","FEMALE": 1 2 1 1 1 2 2 1 2 1 ...
##  $ SRAGE_P : num  32 80 71 39 75 53 67 42 33 67 ...
##  $ MARIT2  : Factor w/ 5 levels "NOT ASCERTAINED",..: 2 4 2 5 2 2 2 2 2 4 ...
##  $ AB1     : Factor w/ 9 levels "NOT ASCERTAINED",..: 5 5 6 5 6 7 6 6 6 5 ...
##  $ ASTCUR  : Factor w/ 3 levels "NOT ASCERTAINED",..: 3 3 2 3 3 2 3 3 3 3 ...
##  $ AB51    : Factor w/ 7 levels "NOT ASCERTAINED",..: 4 4 4 4 4 4 4 4 4 4 ...
##  $ POVLL   : Factor w/ 4 levels "0-99% FPL","100-199% FPL",..: 4 4 4 4 4 4 4 4 3 4 ...
```

As a first exploration of the data, plot two histograms using ggplot2 syntax: one for age (SRAGE_P) and one for BMI (BMI_P). The goal is to explore the dataset and get familiar with the distributions here. Feel free to explore different bin widths. We'll ask some questions about these in the next exercises.

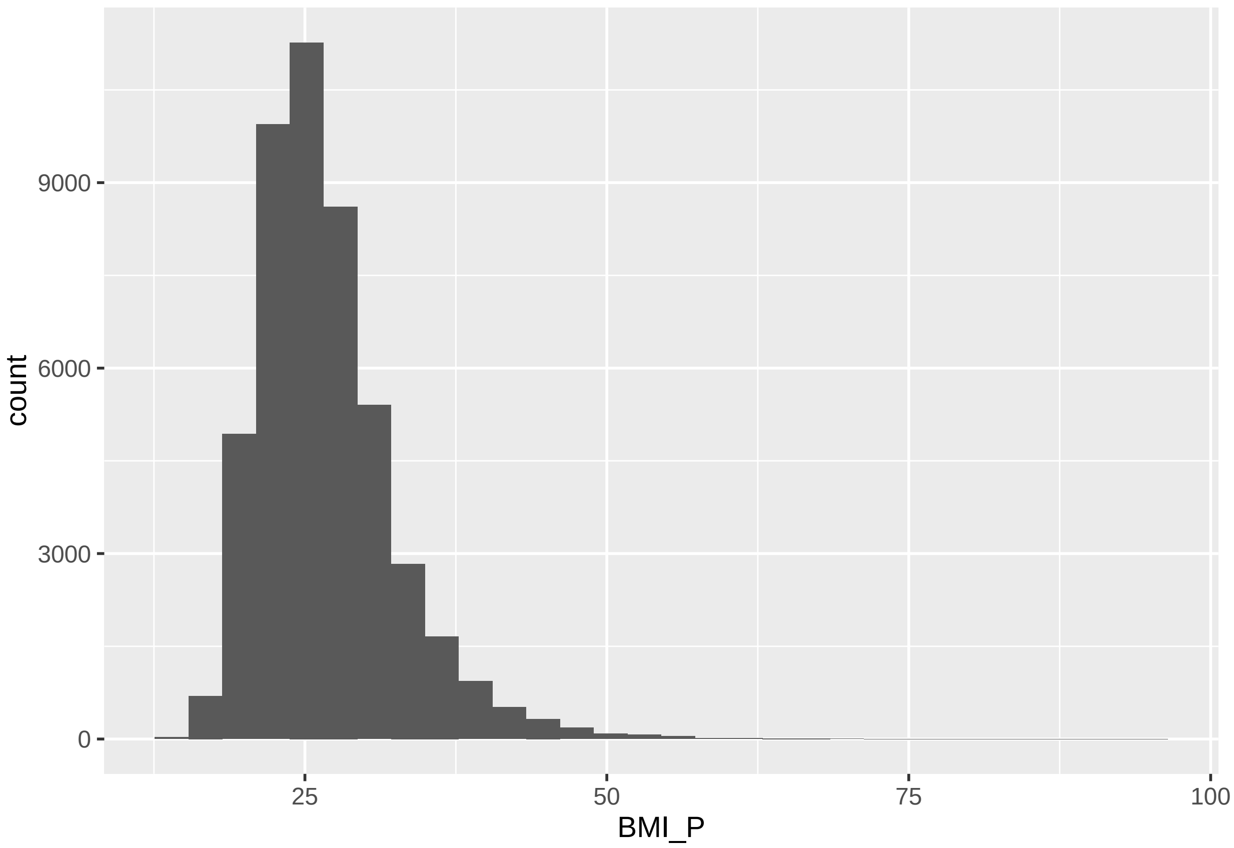
```
# Age histogram
ggplot(adult, aes(x = SRAGE_P)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



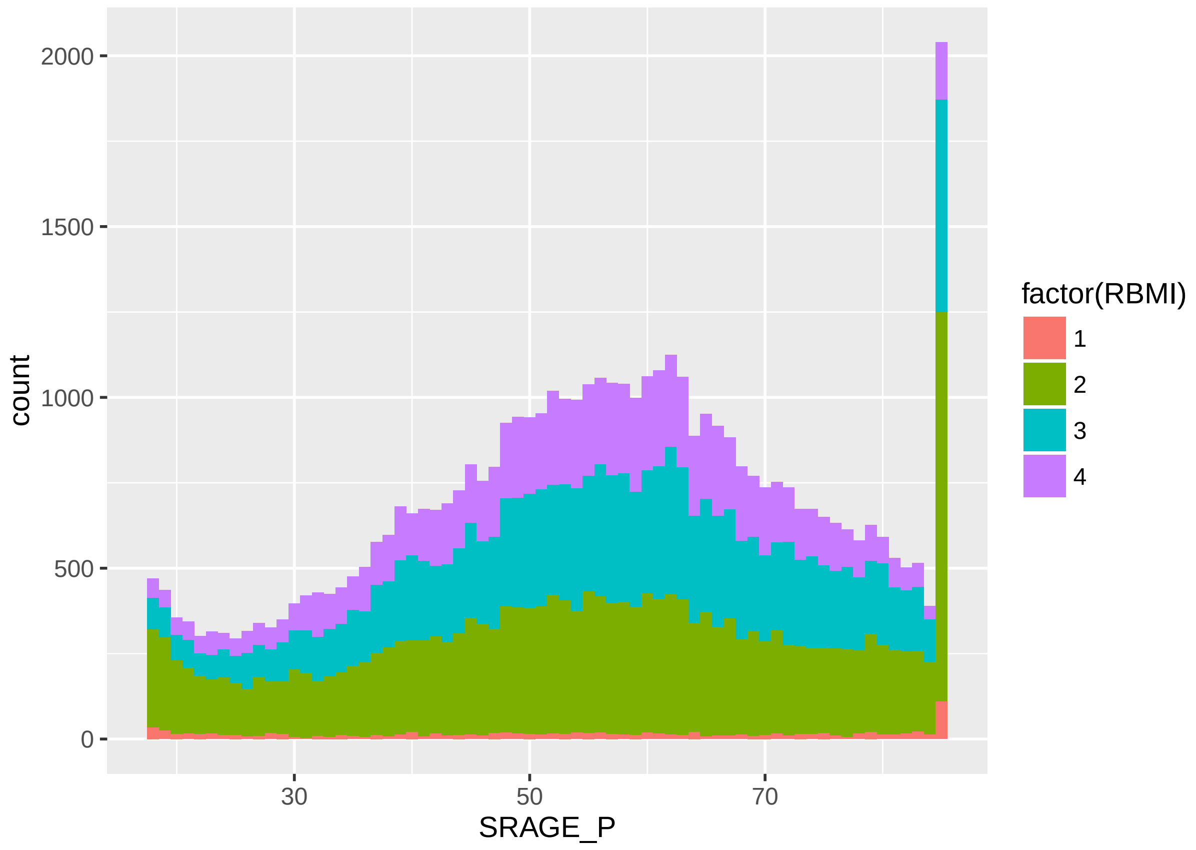
```r
# BMI histogram
ggplot(adult, aes(x = BMI_P)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Next plot a binned-distribution of age, filling each bar according to the BMI categorization. Inside geom_histogram(), set binwidth = 1. You'll want to use factor() to map RBMI as a categorical variable.

```
# Age colored by BMI, default binwidth
ggplot(adult, aes(x = SRAGE_P, fill = factor(RBMI))) +
  geom_histogram(binwidth = 1)
```



Unusual Values

In the previous exercise you used histograms to explore the age and BMI distributions and their relationships to each other in the CHIS dataset. What unusual phenomenon stood out?

The answer is there is an unexpectedly large number of very old people.

Default Binwidths

If you don't specify the binwidth argument inside geom_histogram() you can tell from the message that 30 bins are used by default. This will then specify the binwidth that is used. What is this binwidth for the age variable, SRAGE_P, of the adult dataset?

The answer is 2.23.

Data Cleaning

Now that we have an idea about our data, let's clean it up.

You should have noticed in the age distribution that there is an unusual spike of individuals at 85, which seems like an artifact of data collection and storage. Solve this by only keeping observations for which adult$SRAGE_P is smaller than or equal to 84.

```r
# Remove individual aboves 84
adult <- adult[adult$SRAGE_P < 85, ]
```

There is a long positive tail on the BMIs that we'd like to remove. Only keep observations for which adult BMI_P is bigger than or equal to 16 and adult BMI_P is strictly smaller than 52.

```r
# Remove individuals with a BMI below 16 and above or equal to 52
adult <- adult[adult$BMI_P > 16 & adult$BMI_P <= 52, ]
```

We'll focus on the relationship between the BMI score (& category), age and race. To make plotting easier later on, we'll change the labels in the dataset. Define adult RACEHPR2 as a factor with labels c("Latino", "Asian", "African American", "White"). Do the same for adult RBMI, using the labels c("Under-weight", "Normal-weight", "Over-weight", "Obese").

```r
# Relabel the race variable:
adult$RACEHPR2 <- factor(adult$RACEHPR2, levels = c(1, 4, 5, 6), labels = c("Latino", "Asian", "African
summary(adult$RACEHPR2)
```

```
##           Latino            Asian African American            White 
##             5643             4750             1875            29899 
##             NA's 
##             3193
```

```r
# Relabel the BMI categories variable:
adult$RBMI <- factor(adult$RBMI, levels = c(1, 2, 3, 4), labels = c("Under-weight", "Normal-weight", "O
summary(adult$RBMI)
```

```
##  Under-weight Normal-weight   Over-weight         Obese 
##           905         18692         15530         10233
```

Multiple Histograms

When we introduced histograms we focused on univariate data, which is exactly what we've been doing here. However, when we want to explore distributions further there is much more we can do. For example, there are density plots, which you'll explore in the next course. For now, we'll look deeper at frequency histograms and begin developing our mosaic plots.

The adult dataset, which is cleaned up by now, is availabe in the workspace for you.

For the first instruction, you don't have to add any code. Just have a look at the BMI_fill. This is a scale layer which we can add to a ggplot() command using +: ggplot(...) + BMI_fill. Next, have a look at fix_strips, this is a theme() layer which will make the category titles display more natural when using a faceted plot.

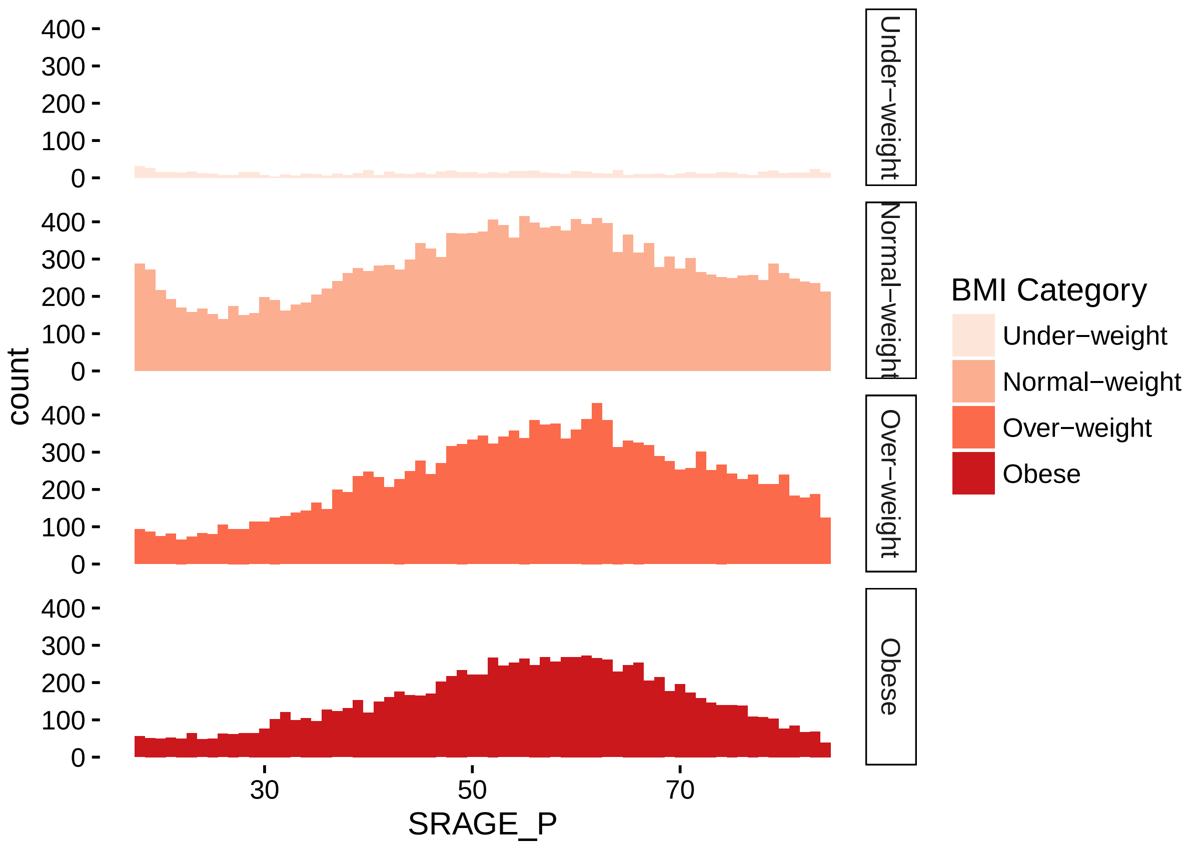
```r
# The dataset adult is available

# The color scale used in the plot
BMI_fill <- scale_fill_brewer("BMI Category", palette = "Reds")

# Theme to fix category display in faceted plot
fix_strips <- theme(strip.text.y = element_text(angle = 0, hjust = 0, vjust = 0.1, size = 14),
                    strip.background = element_blank(),
                    legend.position = "none")
```

The histogram from the first exercise of age colored by BMI has been provided. The predefined theme(), fix_strips has been added to the histogram. Add BMI_fill to this plot using the + operator as well.

In addition, add the following elements to create a pretty insightful plot:

- Use facet_grid() to facet the rows according to RBMI.
- Add the classic theme using theme_classic().

```
# Histogram, add BMI_fill and customizations
ggplot(adult, aes (x = SRAGE_P, fill= factor(RBMI))) +
  BMI_fill +
  geom_histogram(binwidth = 1) +
  fix_strips +
  facet_grid(RBMI ~ .) +
  theme_classic()
```

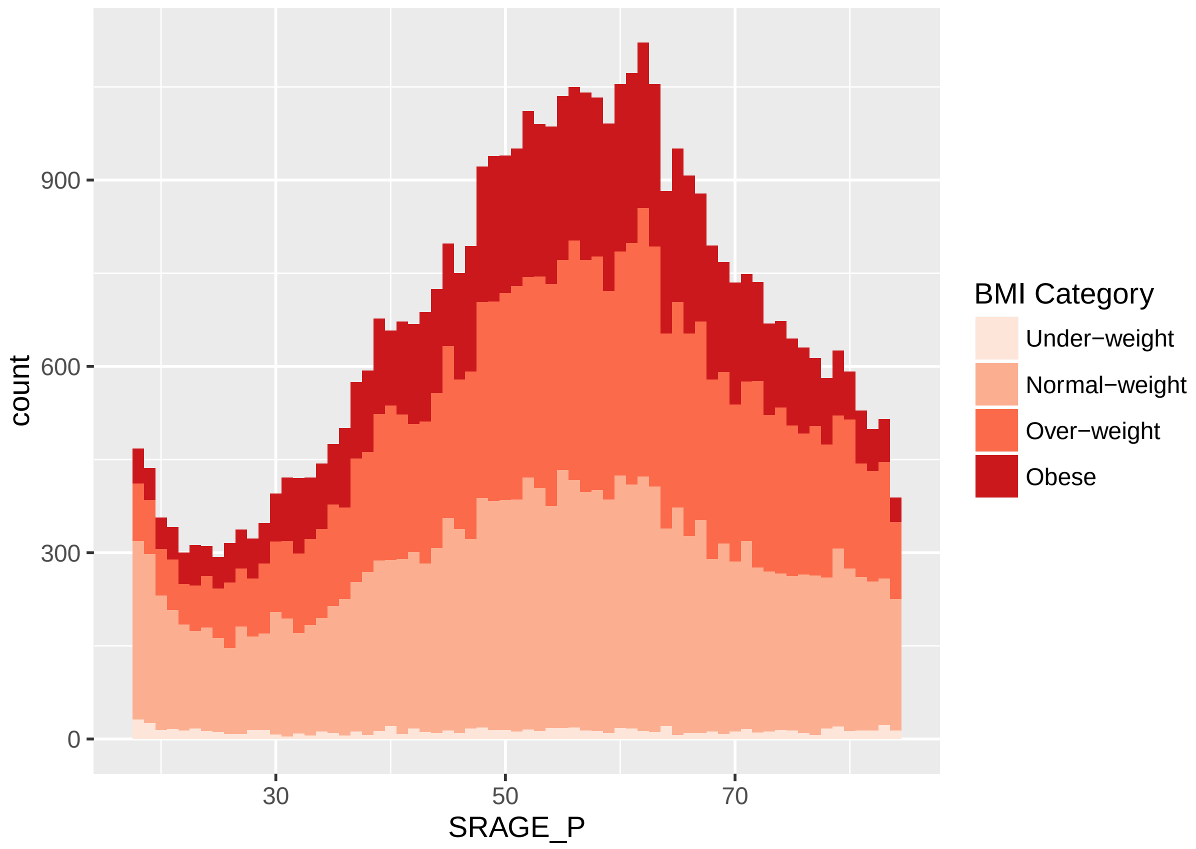


Alternatives

In the previous exercise we looked at different ways of showing the absolute count of multiple histograms. This is fine, but density would be more useful measure if we wanted to see how the frequency of one variable changes accross another. However, there are some difficulties here, so let's take a closer look at different plots.

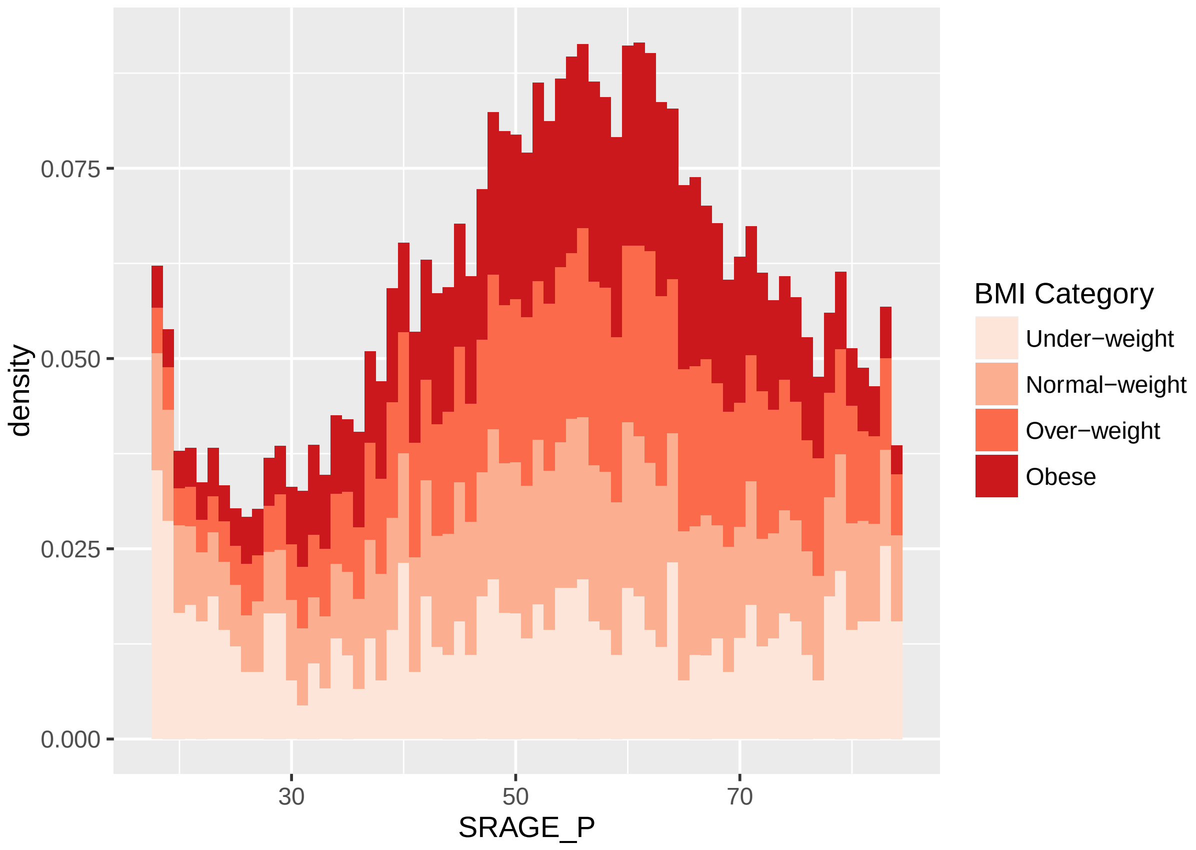
The clean adult dataset is available, as is the BMI_fill color palette. The first plot simply shows a histogram of counts, without facets, without modified themes. It's denoted Plot 1.

```r
# Plot 1 - Count histogram
ggplot(adult, aes (x = SRAGE_P, fill= factor(RBMI))) +
  geom_histogram(binwidth = 1) +
  BMI_fill
```



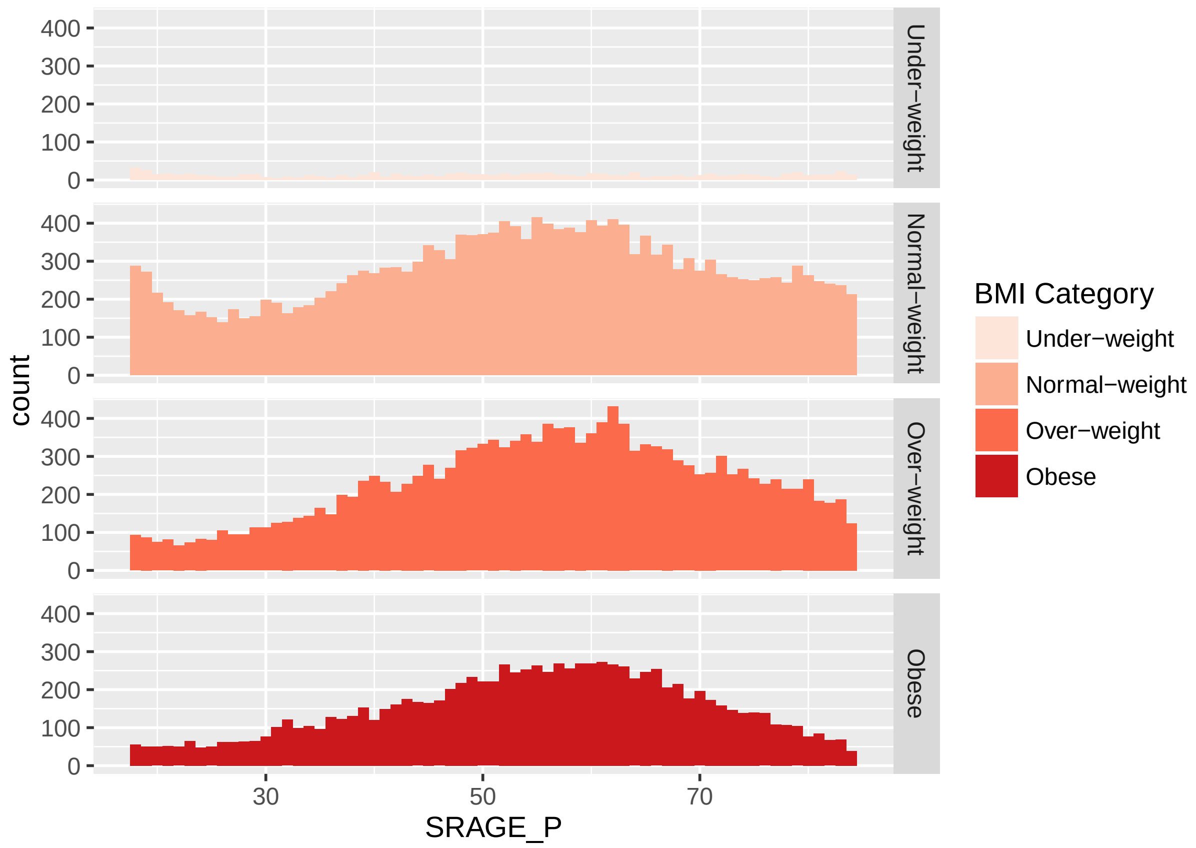
Plot 2 - Copy, paste and adapt the code for plot 1 so that it shows density. Do this by adding aes(y = ..density..) inside the geom_histogram() function. This plot looks really strange, because we get the density within each BMI category, not within each age group!

```r
# Plot 2 - Density histogram
ggplot(adult, aes (x = SRAGE_P, fill= factor(RBMI))) +
  geom_histogram(binwidth = 1, aes(y = ..density..)) +
  BMI_fill
```



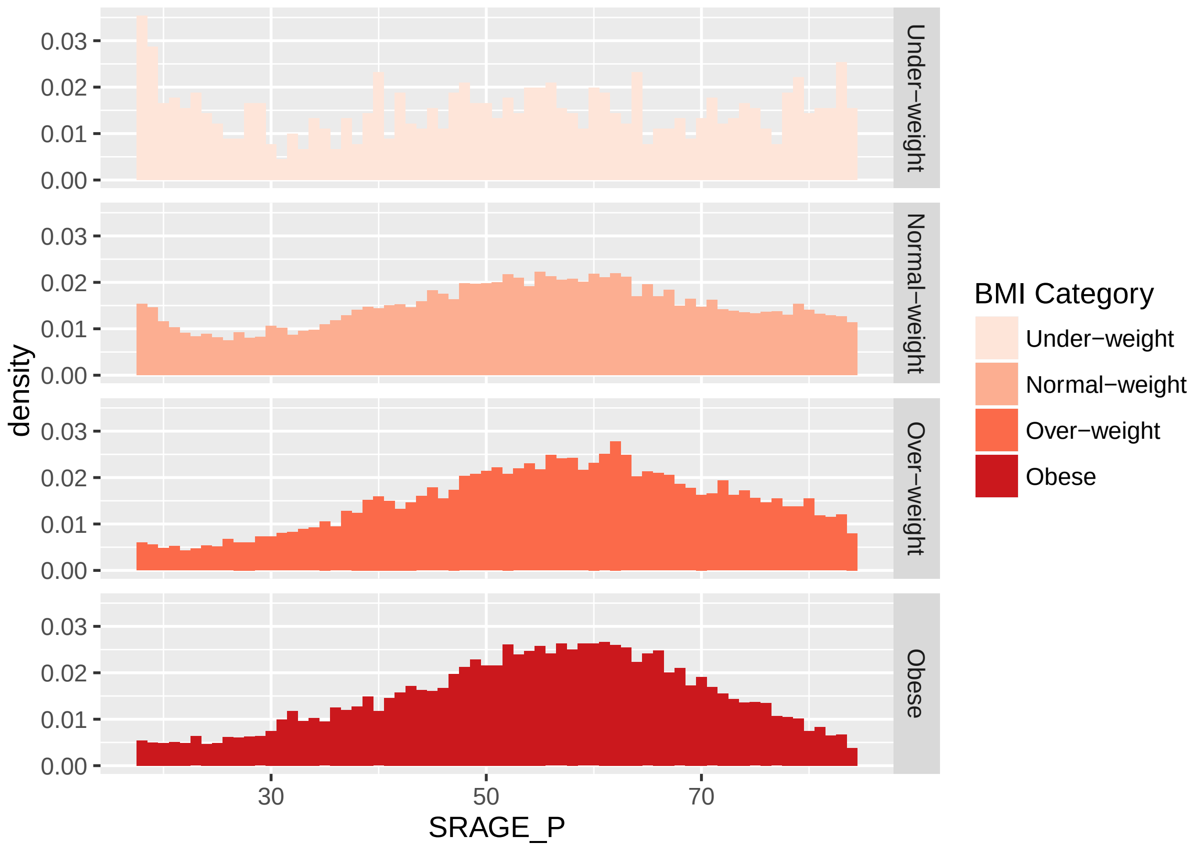
Plot 3 - starting from plot 1, create a faceted histogram. Use facet_grid() with the formula: RBMI ~ ..

```
# Plot 3 - Faceted count histogram
ggplot(adult, aes (x = SRAGE_P, fill= factor(RBMI))) +
  geom_histogram(binwidth = 1) +
  BMI_fill +
  facet_grid(RBMI ~ .)
```



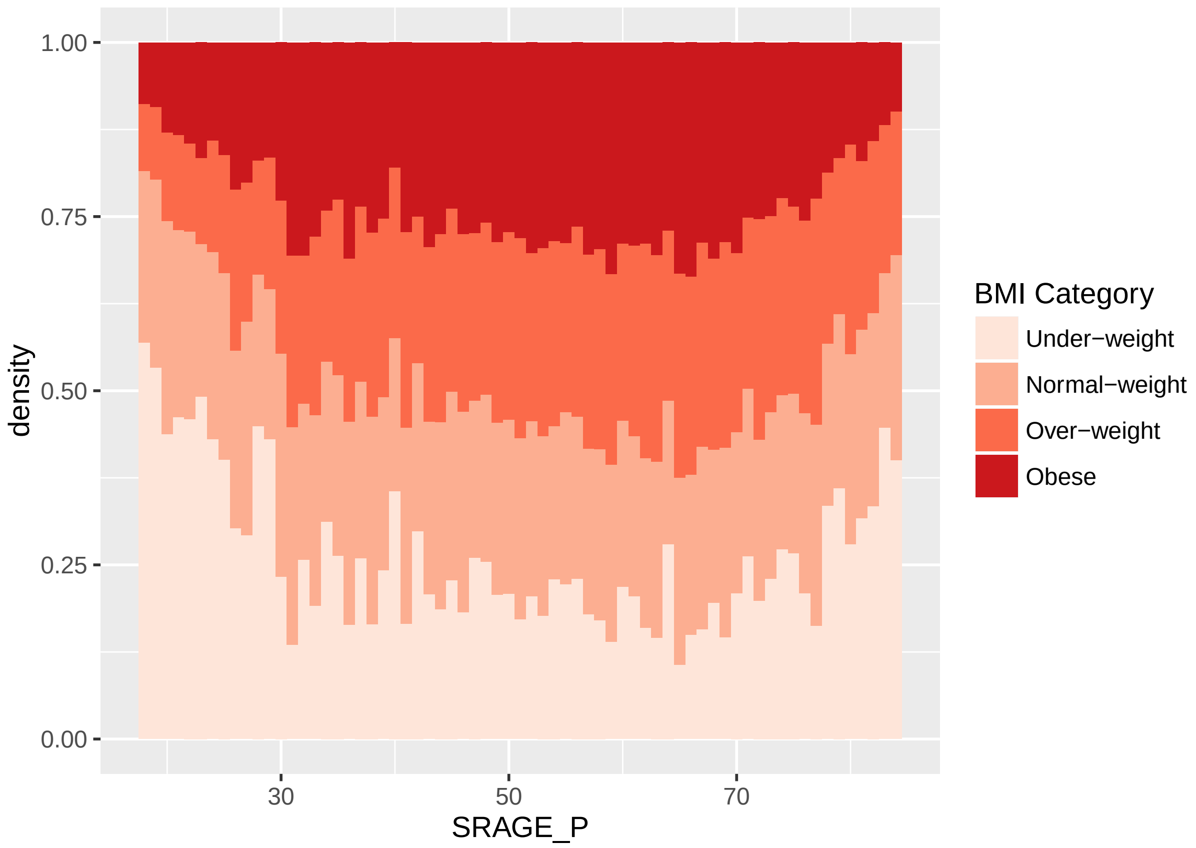
Plot 4 - starting from plot 2, create a faceted histogram showing density. Use facet_grid() with the formula RBMI ~ .. Plots 3 and 4 can be useful if we are interested in the frequency distribution within each BMI category.

```
# Plot 4 - Faceted density histogram
ggplot(adult, aes (x = SRAGE_P, fill= factor(RBMI))) +
  geom_histogram(binwidth = 1, aes(y = ..density..)) +
  BMI_fill +
  facet_grid(RBMI ~ .)
```



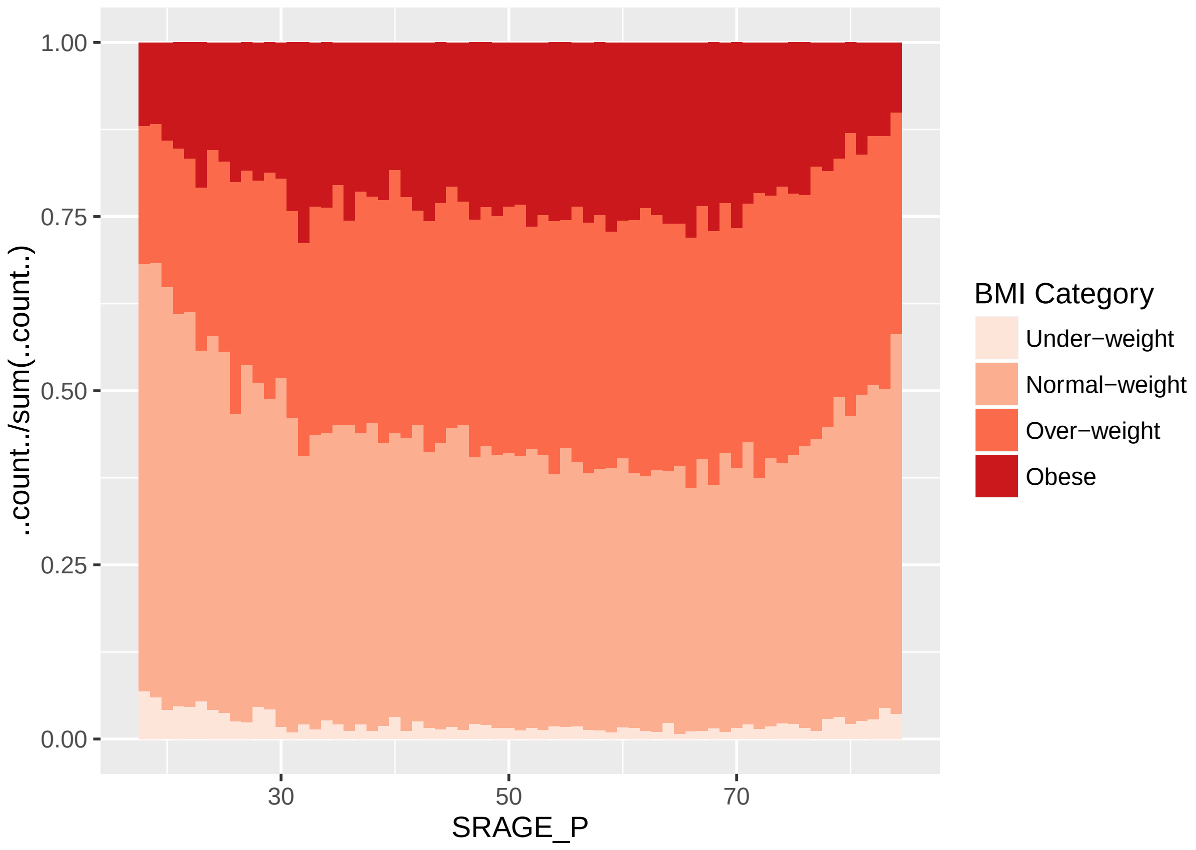
Plot 5 - Change the second plot to have position = "fill". This is not an accurate representation, as density calculates the proportion across category, and not across bin.

```
# Plot 5 - Density histogram with position = "fill"
ggplot(adult, aes (x = SRAGE_P, fill= factor(RBMI))) +
  geom_histogram(binwidth = 1, aes(y = ..density..), position = "fill") +
  BMI_fill
```



Plot 6 - To get an accurate visualization, change Plot 5, but this time, instead of ..density.., set the y aesthetic to ..count../sum(..count..).

```r
# Plot 6 - The accurate histogram
ggplot(adult, aes (x = SRAGE_P, fill= factor(RBMI))) +
  geom_histogram(binwidth = 1, aes(y = ..count../sum(..count..)), position = "fill") +
  BMI_fill
```



Do Things Manually

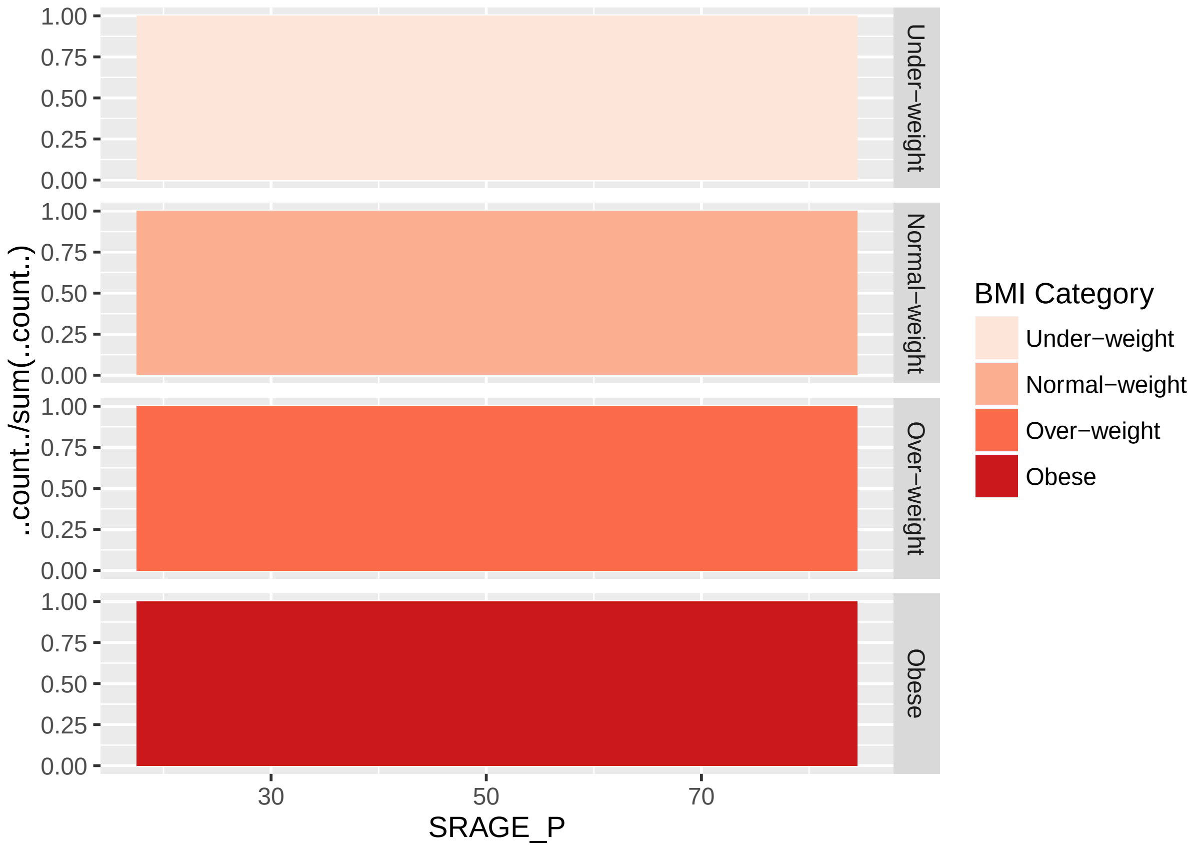
In the previous exercise we looked at how to produce a frequency histogram when we have many sub-categories. The problem here is that this can't be facetted because the calculations occur on the fly inside ggplot2.

To overcome this we're going to calculate the proportions outside ggplot2. This is the beginning of our flexible script for a mosaic plot.

The dataset adult and the BMI_fill object from the previous exercise have been carried over for you. Code that tries to make the accurate frequency histogram faceted is available. Execute it to understand what is happening.

Use adult$RBMI and adult$SRAGE_P as arguments in table() to create a contingency table of the two variables. Save this as DF.

```
# An attempt to facet the accurate frequency histogram from before (failed)
ggplot(adult, aes (x = SRAGE_P, fill= factor(RBMI))) +
  geom_histogram(aes(y = ..count../sum(..count..)), binwidth = 1, position = "fill") +
  BMI_fill +
  facet_grid(RBMI ~ .)
```



```r
# Create DF with table()
DF <- table(adult$RBMI, adult$SRAGE_P)
summary(DF)
```

```
## Number of cases in table: 45360
## Number of factors: 2
## Test for independence of all factors:
##  Chisq = 1120.8, df = 198, p-value = 1.203e-128
```

Use apply() To get the frequency of each group. The first argument is DF, the second argument 2, because you want to do calculations on each column. The third argument should be function(x) x/sum(x). Store the result as DF_freq.

```r
# Use apply on DF to get frequency of each group
DF_freq <- apply(DF, 2, function(x) x/sum(x))
```

Load the reshape2 package and use the melt() function on DF_freq. Store the result as DF_melted. Examine the structure of DF_freq and DF_melted if you are not familiar with this operation.

```r
# Load reshape2 and use melt on DF to create DF_melted
library(reshape2)
DF_melted <- melt(DF_freq)
str(DF_freq)
```

```
##  num [1:4, 1:67] 0.0684 0.6132 0.1987 0.1197 0.0596 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:4] "Under-weight" "Normal-weight" "Over-weight" "Obese"
##   ..$ : chr [1:67] "18" "19" "20" "21" ...
```

```r
str(DF_melted)
```

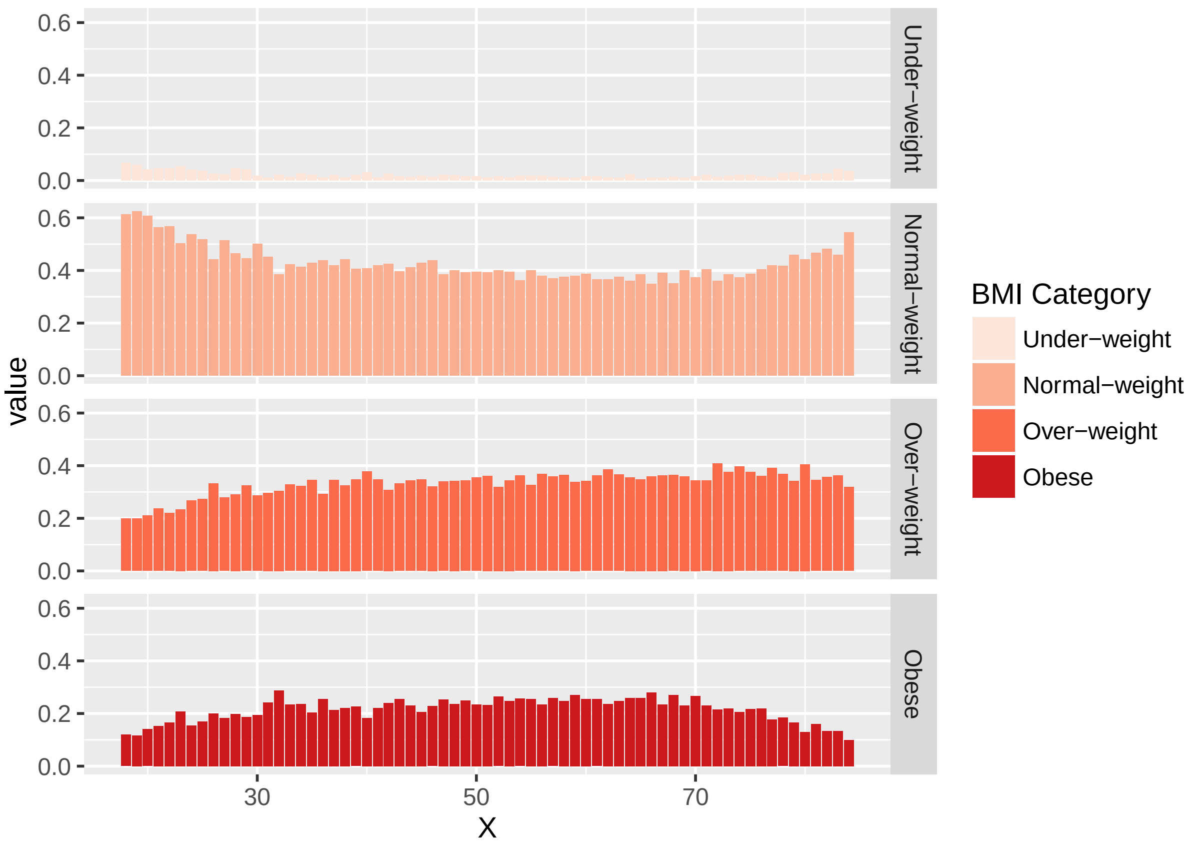
```
## 'data.frame':    268 obs. of  3 variables:
##  $ Var1 : Factor w/ 4 levels "Under-weight",..: 1 2 3 4 1 2 3 4 1 2 ...
##  $ Var2 : int  18 18 18 18 19 19 19 19 20 20 ...
##  $ value: num  0.0684 0.6132 0.1987 0.1197 0.0596 ...
```

Use names() to rename the variables in DF_melted to be c("FILL", "X", "value"), with the prospect of making this a generalized function later on.

```r
# Change names of DF_melted
names(DF_melted) <- c("FILL", "X", "value")
```

The plotting call at the end uses DF_melted. Add code to make it faceted. Use the formula FILL ~ .. Note that we use geom_bar() now!

```r
# Add code to make this a faceted plot
ggplot(DF_melted, aes(x = X, y = value, fill = FILL)) +
  geom_bar(stat = "identity", position = "stack") +
  BMI_fill +
  facet_grid(FILL ~ .)
```



Merimeko/Mosaic Plot

In the previous exercise we looked at different ways of showing the frequency distribution within each BMI category. This is all well and good, but the absolute number of each age group also has an influence on if we will consider something as over-represented or not. Here, we will proceed to change the widths of the bars to show us something about the n in each group.

This will get a bit more involved, because the aim is not to draw bars, but rather rectangles, for which we can control the widths. You may have already realized that bars are simply rectangles, but we don't have easy access to the xmin and xmax aesthetics, but in geom_rect() we do! Likewise, we also have access to ymin and ymax. So we're going to draw a box for every one of our 268 distinct groups of BMI category and age.

The clean adult dataset, as well as BMI_fill, are already available. Instead of running apply() like in the previous exercise, the contingency table has already been transformed to a data frame using as.data.frame.matrix().

To build the rectangle plot, we'll add several variables to DF:

- groupSum, containing the sum of each row in the DF. Use rowSums() to calculate this. groupSum represents the total number of individuals in each age group.

```r
# The initial contingency table
DF <- as.data.frame.matrix(table(adult$SRAGE_P, adult$RBMI))
```

- xmax: the xmax value for each rectangle, calculated as cumsum(DF$groupSum)
- xmin: the xmin value for each rectangle, calculated by subtracting the groupSum column from the xmax column.

```r
# Add the columns groupsSum, xmax and xmin. Remove groupSum again.
DF$groupSum <- rowSums(DF)
DF$xmax <- cumsum(DF$groupSum)
DF$xmin <- DF$xmax - DF$groupSum
# The groupSum column needs to be removed, don't remove this line
DF$groupSum <- NULL
```

- The names of the x axis groups are stored in the row names, which is pretty bad style, so make a new variable, X, that stores the values of row.names() for DF.

```r
# Copy row names to variable X
DF$X <- row.names(DF)
```

Now we are ready to melt the dataset. Load reshape2 and use melt() on DF. Specify the id.vars variables as c("X", "xmin", "xmax") and the variable.name argument as "FILL". Store the result as DF_melted.

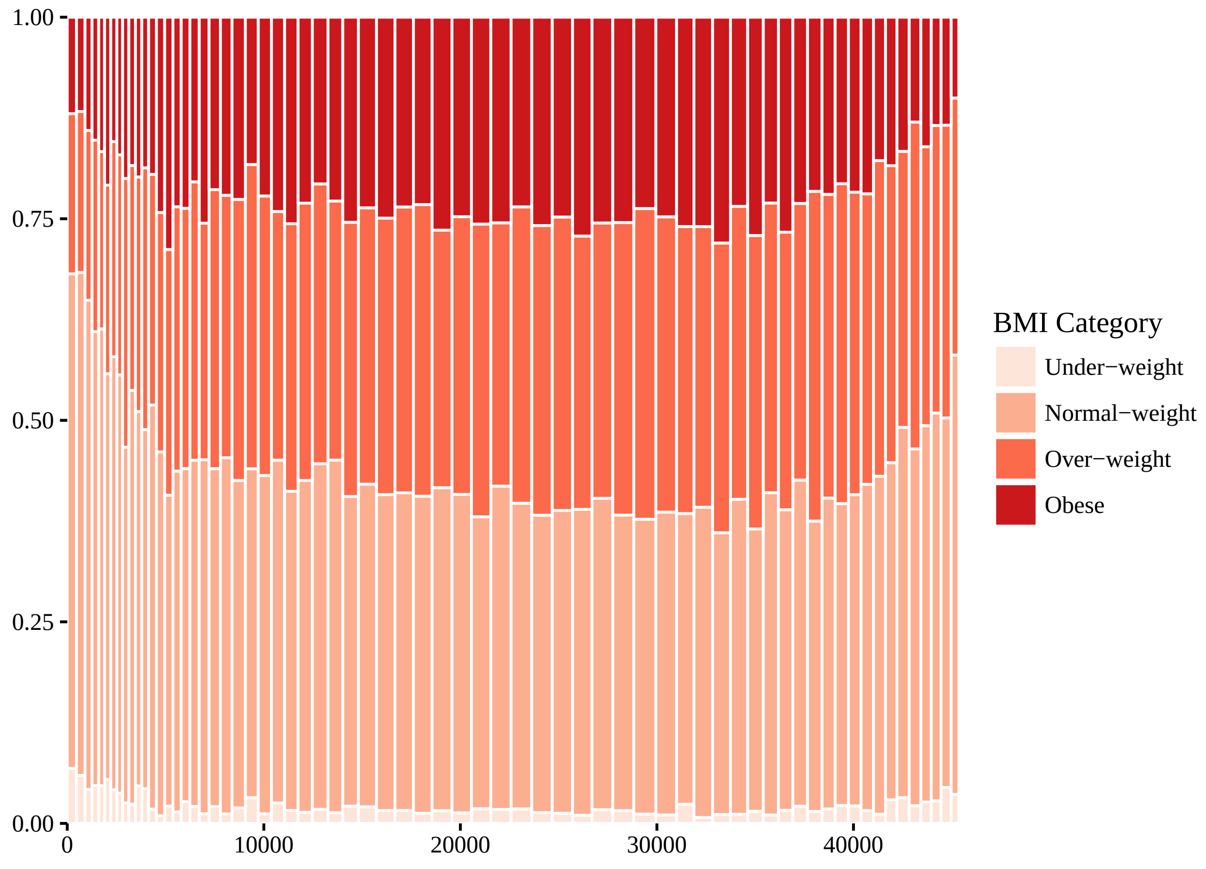
```r
# Melt the dataset
library(reshape2)
DF_melted <- melt(DF, id.vars = c("X", "xmin", "xmax"), variable.name = "FILL")
```

Have a look at the dplyr call that calculates the ymax and ymin columns of DF_melted. It first groups by X and then calculates cumulative proportions. The result is stored as DF_melted again.

```r
# dplyr call to calculate ymin and ymax - don't change
library(dplyr)
DF_melted <- DF_melted %>%
  group_by(X) %>%
  mutate(ymax = cumsum(value/sum(value)),
         ymin = ymax - value/sum(value))
```

If all goes well you should see the plot in the viewer when you execute the plotting function at the bottom of the script.

```r
# Plot rectangles - don't change.
library(ggthemes)
ggplot(DF_melted, aes(ymin = ymin,
                      ymax = ymax,
                      xmin = xmin,
                      xmax = xmax,
                      fill = FILL)) +
  geom_rect(colour = "white") +
  scale_x_continuous(expand = c(0,0)) +
  scale_y_continuous(expand = c(0,0)) +
  BMI_fill +
  theme_tufte()
```



Adding Statistics

In the previous exercise we generated a plot where each individual bar was plotted separately using rectangles (shown in the viewer). This means we have access to each piece and we can apply different fill parameters.

So let's make some new parameters. To get the Pearson residuals, we'll use the chisq.test().

The data frames adult and DF_melted, as well as the object BMI_fill that you created throughout this chapter, are all still available. The reshape2 package is already loaded.

Use the adult*RBMI*(*correspondingtoFILL*)*andadult*SRAGE_P (corresponding to X) columns inside the table() function that's inside the chisq.test() function. Store the result as results.

```r
# Perform chi.sq test (RBMI and SRAGE_P)
results <- chisq.test(table(adult$RBMI, adult$SRAGE_P))
```

The residuals can be accessed through results$residuals. Apply the melt() function on them with no further arguments. Store the resulting data frame as resid.

```r
# Melt results$residuals and store as resid
resid <- melt(results$residuals)
```

Change the names of resid to c("FILL", "X", "residual"). This is so that we have a consistent naming convention similar to how we called our variables in the previous exercises.

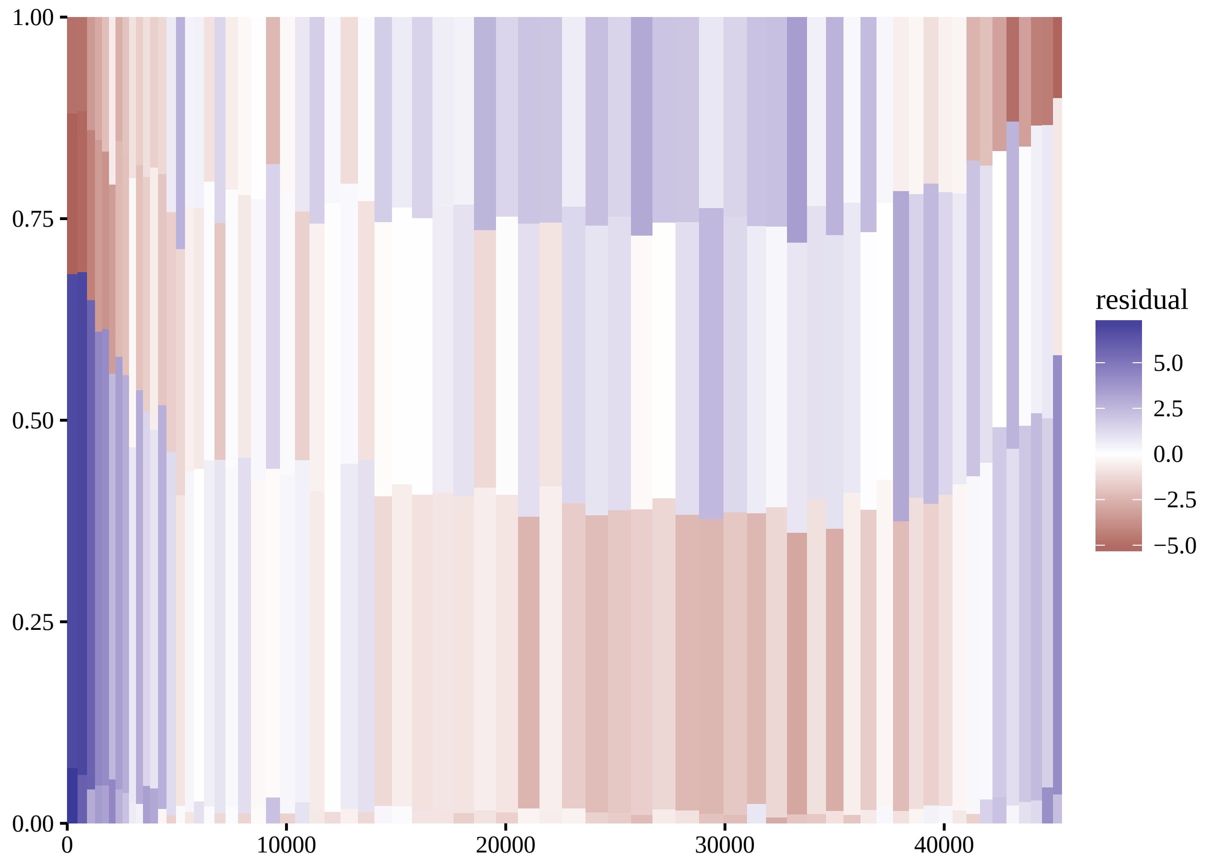
```r
# Change names of resid
names(resid) <- c("FILL", "X", "residual")
```

The data frame from the previous exercise, DF_melted is already available. Use the merge() function to bring the two data frames together. Store the result as DF_all.

```r
# merge the two datasets:
DF_all <- merge(DF_melted, resid)
```

Adapt the code in the ggplot command to use DF_all instead of DF_melted. Also, map residual onto fill instead of FILL.

```r
# Update plot command
library(ggthemes)
ggplot(DF_all, aes(ymin = ymin,
                   ymax = ymax,
                   xmin = xmin,
                   xmax = xmax,
                   fill = residual)) +
  geom_rect() +
  scale_fill_gradient2() +
  scale_x_continuous(expand = c(0,0)) +
  scale_y_continuous(expand = c(0,0)) +
  theme_tufte()
```



Adding Text

Now that we are not coloring according to BMI category, we have to add the group labels manually. Also, we neglected to label the x-axis properly!

Here we'll use the label aesthetic inside geom_text(). The actual labels will be the FILL and X columns in the DF data frame. Since we have axes on the left and bottom of our plot, we'll add information to the top and right inner edges of the plot. We could have also added margin text, but that is a more advanced topic. This will be a suitable solution for the moment.

To position our labels correctly, we need to calculate the midpoint between each xmax and xmin value. To get this, calculate the half difference between each pair of xmax and xmin then add this value to xmin.

```
# Position for labels on x axis
DF_all$xtext <- DF_all$xmin + (DF_all$xmax - DF_all$xmin)/2
```

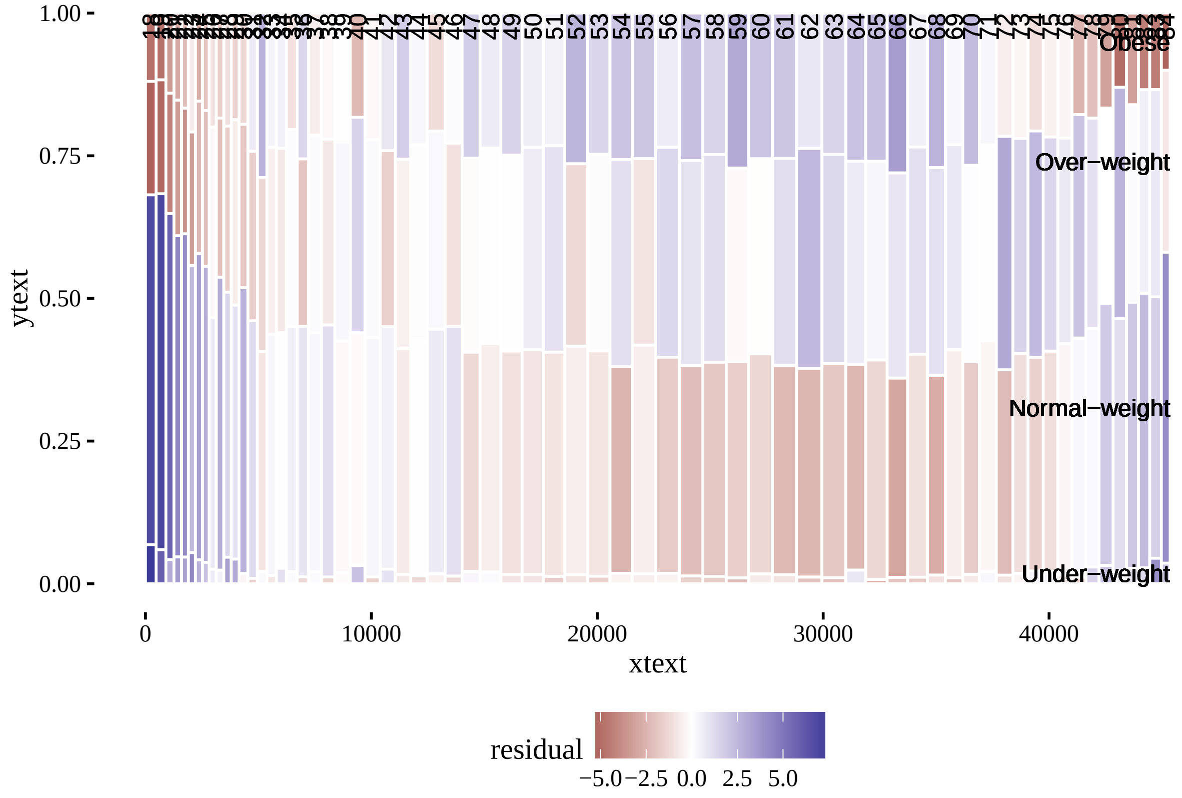
For this instruction, you don't have to write code yourself. For the y label positions, we only want to work with the values at the maximum xmax, i.e. at the very end. The code to calculate the ytext column is already available. Try to understand it.

```
# Position for labels on y axis (don't change)
index <- DF_all$xmax == max(DF_all$xmax)
DF_all$ytext <- DF_all$ymin[index] + (DF_all$ymax[index] - DF_all$ymin[index])/2
```

Now that xtext and ytext are available, we can add the labels to our plot. In the two geom_text() functions, separate aesthetics are defined that control the x and y positioning of the labels.

- For the age groups, set the x position with xtext. The y position is fixed since our y axis is always going to end at 1.
- For the labeling of the y axis, the second geom_text() has three aesthetics since the position on the right will depend on the size of our dataset in the future. Fill in the _____ accordingly. x depends on the max() of xmax. You will have to use the predefind ytext column for y. The correct label is available in the FILL column.
- Some additional attributes have been set inside geom_text(), outside the aes() function. This is just some fine tweaking to get the positioning and angle correct. It's not perfect, but since this is an exploratory plot, it does a pretty good job.

```
# Plot
ggplot(DF_all, aes(ymin = ymin, ymax = ymax, xmin = xmin,
                   xmax = xmax, fill = residual)) +
  geom_rect(col = "white") +
  # geom_text for ages (i.e. the x axis)
  geom_text(aes(x = xtext,
                label = X),
            y = 1,
            size = 3,
            angle = 90,
            hjust = 1,
            show.legend = FALSE) +
  # geom_text for BMI (i.e. the fill axis)
  geom_text(aes(x = max(xmax),
                y = ytext,
                label = FILL),
            size = 3,
            hjust = 1,
            show.legend  = FALSE) +
  scale_fill_gradient2() +
  theme_tufte() +
  theme(legend.position = "bottom")
```



Generalizations

Now that you've done all the steps necessary to make our mosaic plot, you can wrap all the steps into a single function that we can use to examine any two variables of interest in our data frame (or in any other data frame for that matter). For example, we can use it to examine the Vocab data frame we saw earlier in this course.

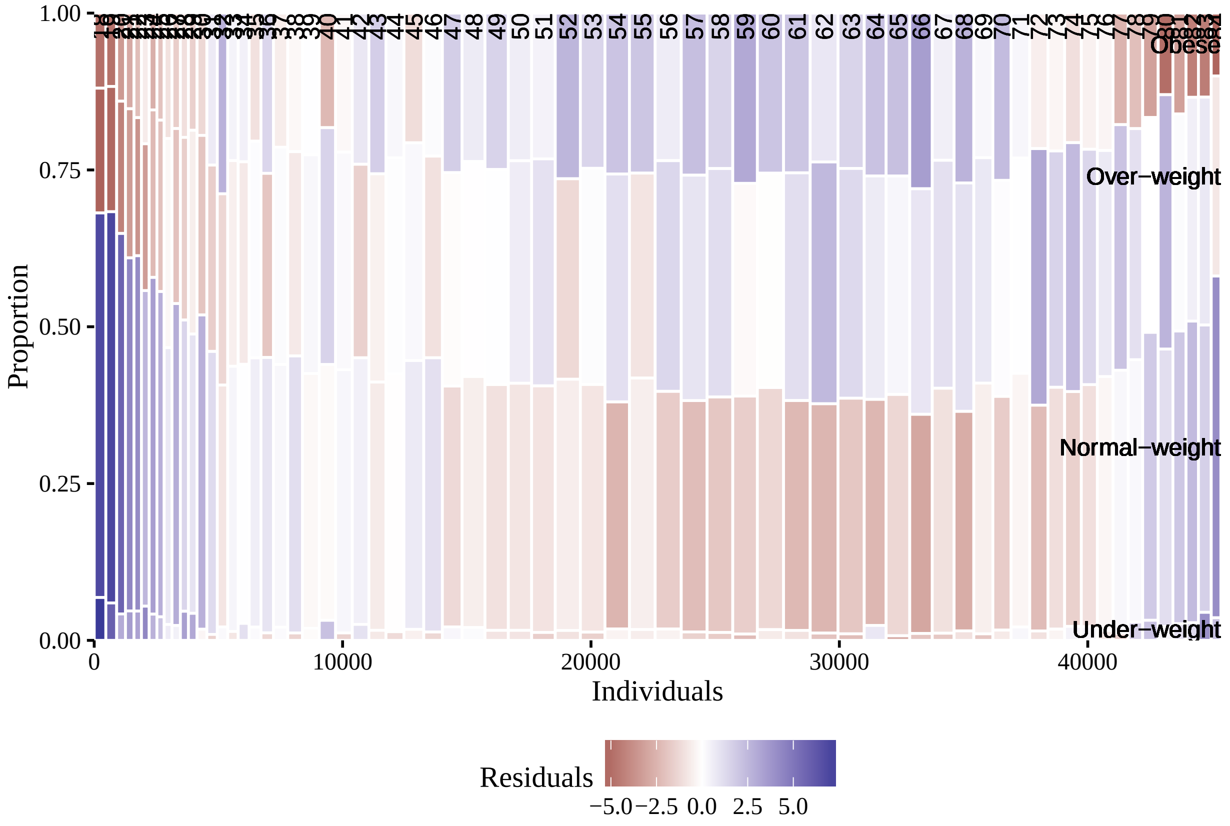
You've seen all the code in our function, so there shouldn't be anything surprising there. Notice that the function takes multiple arguments, such as the data frame of interest and the variables that you want to create the mosaic plot for. None of the arguments have default values, so you'll have to specify all three if you want the mosaicGG() function to work.

Start by going through the code and see if you understand the function's implementation.

```r
# Load all packages
library(ggplot2)
library(reshape2)
library(dplyr)
library(ggthemes)

# Script generalized into a function
mosaicGG <- function(data, X, FILL) {

  # Proportions in raw data
  DF <- as.data.frame.matrix(table(data[[X]], data[[FILL]]))
  DF$groupSum <- rowSums(DF)
  DF$xmax <- cumsum(DF$groupSum)
  DF$xmin <- DF$xmax - DF$groupSum
  DF$X <- row.names(DF)
  DF$groupSum <- NULL
  DF_melted <- melt(DF, id = c("X", "xmin", "xmax"), variable.name = "FILL")
  library(dplyr)
  DF_melted <- DF_melted %>%
    group_by(X) %>%
    mutate(ymax = cumsum(value/sum(value)),
           ymin = ymax - value/sum(value))

  # Chi-sq test
  results <- chisq.test(table(data[[FILL]], data[[X]])) # fill and then x
  resid <- melt(results$residuals)
  names(resid) <- c("FILL", "X", "residual")

  # Merge data
  DF_all <- merge(DF_melted, resid)

  # Positions for labels
  DF_all$xtext <- DF_all$xmin + (DF_all$xmax - DF_all$xmin)/2
  index <- DF_all$xmax == max(DF_all$xmax)
  DF_all$ytext <- DF_all$ymin[index] + (DF_all$ymax[index] - DF_all$ymin[index])/2

  # plot:
  g <- ggplot(DF_all, aes(ymin = ymin,  ymax = ymax, xmin = xmin,
                          xmax = xmax, fill = residual)) +
  geom_rect(col = "white") +
  geom_text(aes(x = xtext, label = X),
            y = 1, size = 3, angle = 90, hjust = 1, show.legend = FALSE) +
  geom_text(aes(x = max(xmax),  y = ytext, label = FILL),
            size = 3, hjust = 1, show.legend = FALSE) +
  scale_fill_gradient2("Residuals") +
  scale_x_continuous("Individuals", expand = c(0,0)) +
  scale_y_continuous("Proportion", expand = c(0,0)) +
  theme_tufte() +
  theme(legend.position = "bottom")
  print(g)
}
```

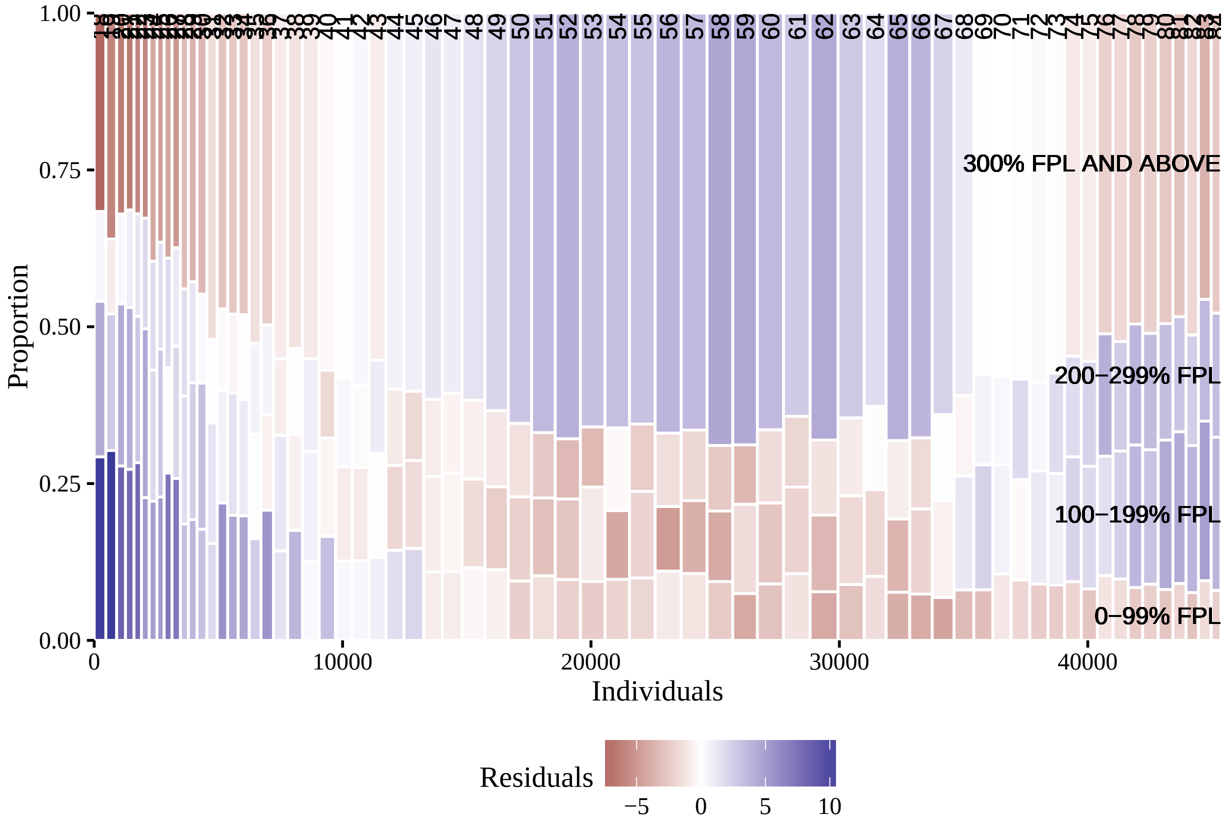
Calling mosaicGG(adult, “SRAGE_P”,“RBMI”) will result in the plot you’ve been working on so far. Try this out. This gives you a mosaic plot where BMI is described by age.

```r
# BMI described by age
mosaicGG(adult, "SRAGE_P", "RBMI")
```



Test out another combination of variables in the adult data frame: Poverty (POVLL) described by Age (SRAGE_P).

```
# Poverty described by age
mosaicGG(adult, "SRAGE_P", "POVLL")
```

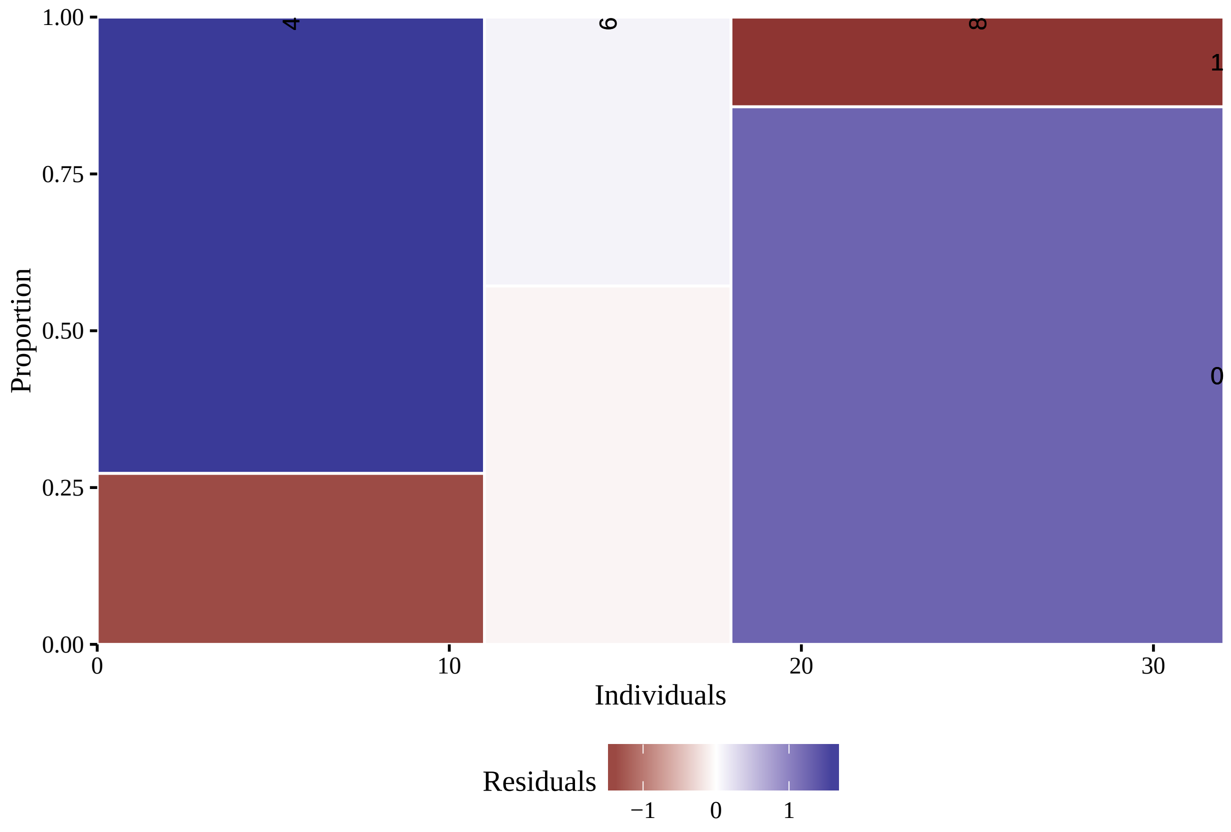


Try the function on other datasets we've worked with throughout this course:

- mtcars dataset: am described by cyl

```
# mtcars: am described by cyl
mosaicGG(mtcars, "cyl", "am")
```

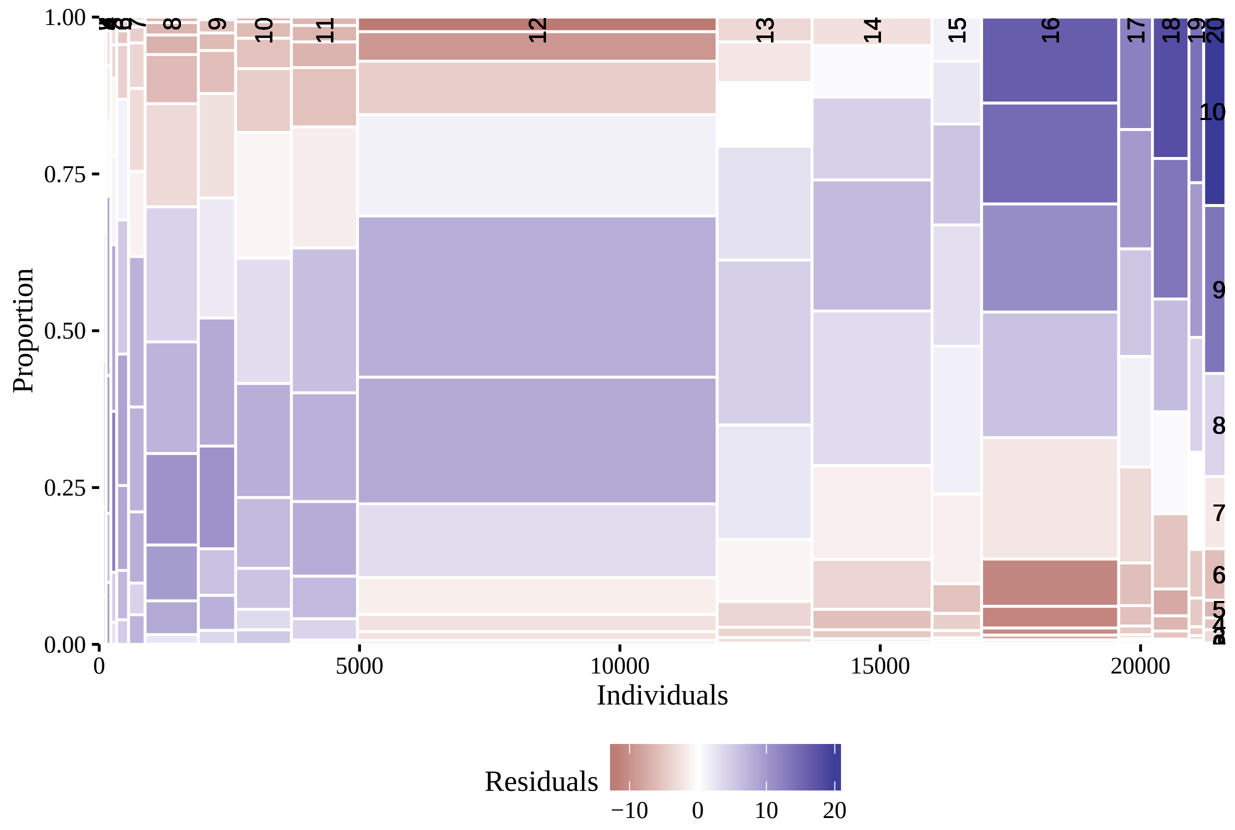
```
## Warning in chisq.test(table(data[[FILL]], data[[X]])): Chi-squared
## approximation may be incorrect
```



- Vocab dataset: vocabulary described by education.

```
library(car)
mosaicGG(Vocab, "education", "vocabulary")
```

```
## Warning in chisq.test(table(data[[FILL]], data[[X]])): Chi-squared
## approximation may be incorrect
```



1.00
0.75
0.50
0.25
0.00
Proportion
0
5000
10000
15000
20000
Individuals
7
8
9
10
11
12
13
14
15
16
17
18
19
20
10
9
8
7
6
5
4
Residuals
−10
0
10
20