Exploring data with graphs

Video lecture 1

Exploratory Data Analysis is the first step when looking at a new dataset, and it is a very important one too. In this video, we will focus on the graphical aspects.

The main purposes of exploratory data analysis are to:

- get insight into the dataset
- discover patterns
- test underlying assumptions
- detect outliers and anomalies
- uncover underlying structure
- extract important variables
- determine optimal factor settings

Load the packages

This is where we load the packages we need before we start analysing data. If you don't have the NHANES package yet, you can install it in the same way as you installed tidyverse using the **install.packages()** function.

Here we will use the **library()** function to load the tidyverse package, and a health data set called NHANES.

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```r
library(NHANES)
```

The NHANES dataset

This is survey data collected by the US National Center for Health Statistics (NCHS) which has conducted a series of health and nutrition surveys since the early 1960's. Since 1999, approximately 5,000 individuals of all ages are interviewed in their homes every year and complete the health examination component of the survey. Find out more here, in particular check the NHANES.pdf reference manual.

Distribution of variables

When looking at a new dataset, it is useful to look at the distribution of the continuous variables. This can tell us what kind of distribution is reasonable to assume when we select our statistical tests and models, and also about potential outliers. A variable is continuous if it can take any of an infinite set of ordered values (e.g. numbers or date-times).

Create a histogram

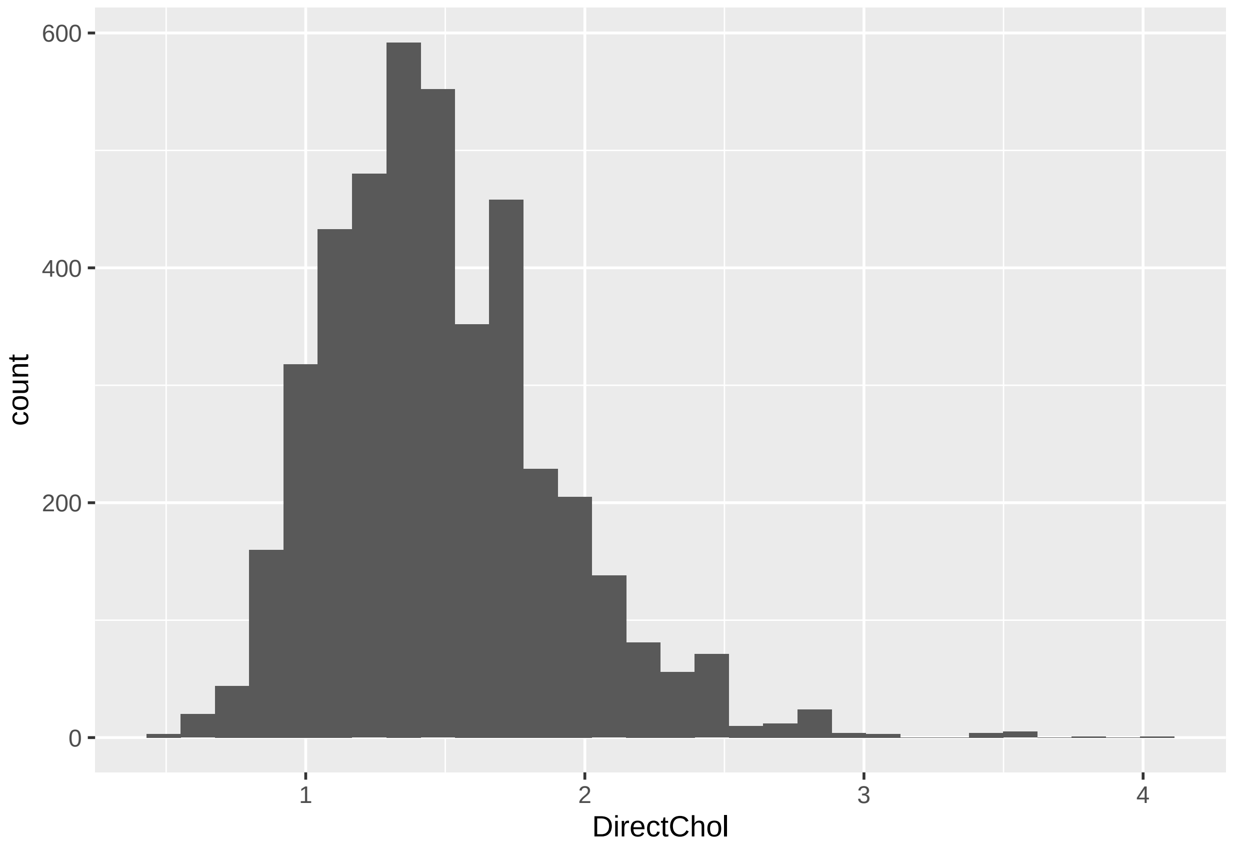
We can look at the distribution of the continuous DirectChol variable, which is the direct HDL cholesterol in mmol/L, using a histogram. First, we take the NHANES dataset, pass it on to the next function using the piping operator %>%, and filter it using the **filter()** function to keep only the rows for female individuals. Then we remove any row where the DirectChol value is missing using the **drop_na()** function.

This filtered tibble is then used to set a **ggplot()** object indicating that the x-axis should represent the DirectChol variable. This is done within the aesthetic mappings **aes()** function, which describes how variables in the data are mapped to visual properties.

Finally, we plot a histogram using the **geom_histogram()** function. You can see that the various ggplot2 functions are separated by + signs. By default, the histogram has 30 equal bins by default and the height of the bars is the count of the number of observations in each bin.

```
NHANES %>%
  filter(Gender=="female") %>%
  drop_na(DirectChol) %>%
  ggplot(aes(x=DirectChol)) +
  geom_histogram()
```

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

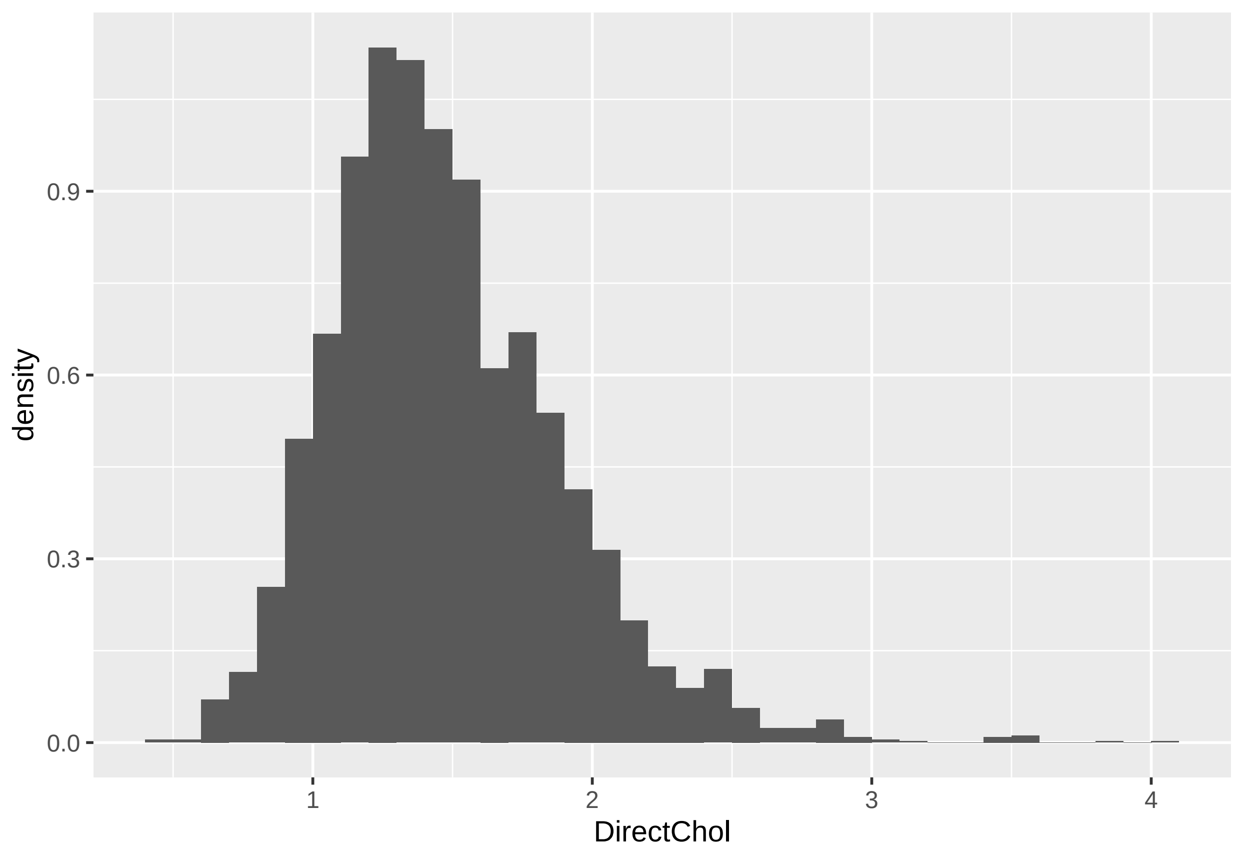


Customise our histogram

Now we want to change our histogram to make it easier to read. Taking the same code as for our basic histogram, we can now change the y-axis to be the density instead of the counts. This is done in the **geom_histogram** function, within the aesthetics parameters (**aes**). We used to use density surrounded by double dots (**..density..**), but now this has been replaced by **after_stat(density)**. This a shortcut to calculate the density and use it directly in the **geom_histogram()** function. So the height of the bars is now the frequency divided by the bin width (density).

Since R suggests that our bins are not ideal, we can also change the width of the bins, which can reveal different patterns about the data. Let's change the bin width to a smaller interval (0.1) by adding the **binwidth** argument to the **geom_histogram()** function. For easier interpretation, we can also change where the bin boundaries are by adding the **boundary** argument to the **geom_histogram()** function. Here we will have 1 as one of the boundaries, so that the bins will be 0.9-1, 1-1.1, 1.1-1.2, etc. where the first number is included and the second number excluded by default.

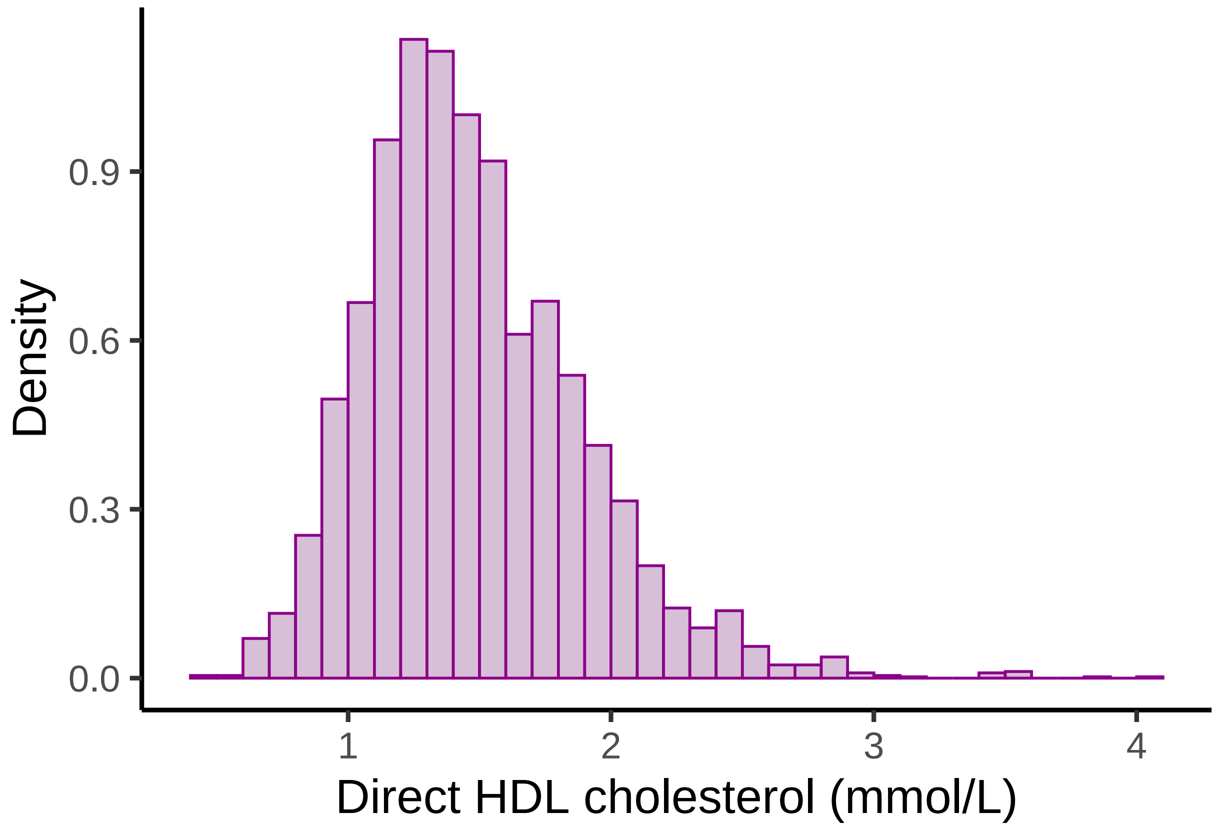
```r
NHANES %>%
  filter(Gender=="female") %>%
  drop_na(DirectChol) %>%
  ggplot(aes(x=DirectChol)) +
  geom_histogram(aes(y=after_stat(density)),
                 binwidth=0.1,
                 boundary = 1)
```



Finally, we can change the colours of our histogram bars, the labels, and the overall appearance of the plot. We add the **col** and **fill** arguments to the **geom_histogram()** function to change the colour of the bar borders and of the inside of the bars, respectively. You can find a helpful list of colours on that page: Colors in R (derekogle.com)

Then we use the **ylab()** and **xlab()** functions to change the labels for the y-axis and x-axis, respectively. Finally, you can change the theme of the ggplot object, which determines how the plot looks by using the function for your preferred theme, here we use the **theme_classic()** function. You can find information about these themes and what they look like on this page: Complete themes — ggtheme • ggplot2 (tidyverse.org). If you want to change an element of the theme, you can do so within the **theme_classic()** function. Here, I have changed the base size to 18 by adding the **base_size** argument to the **theme_classic()** function.

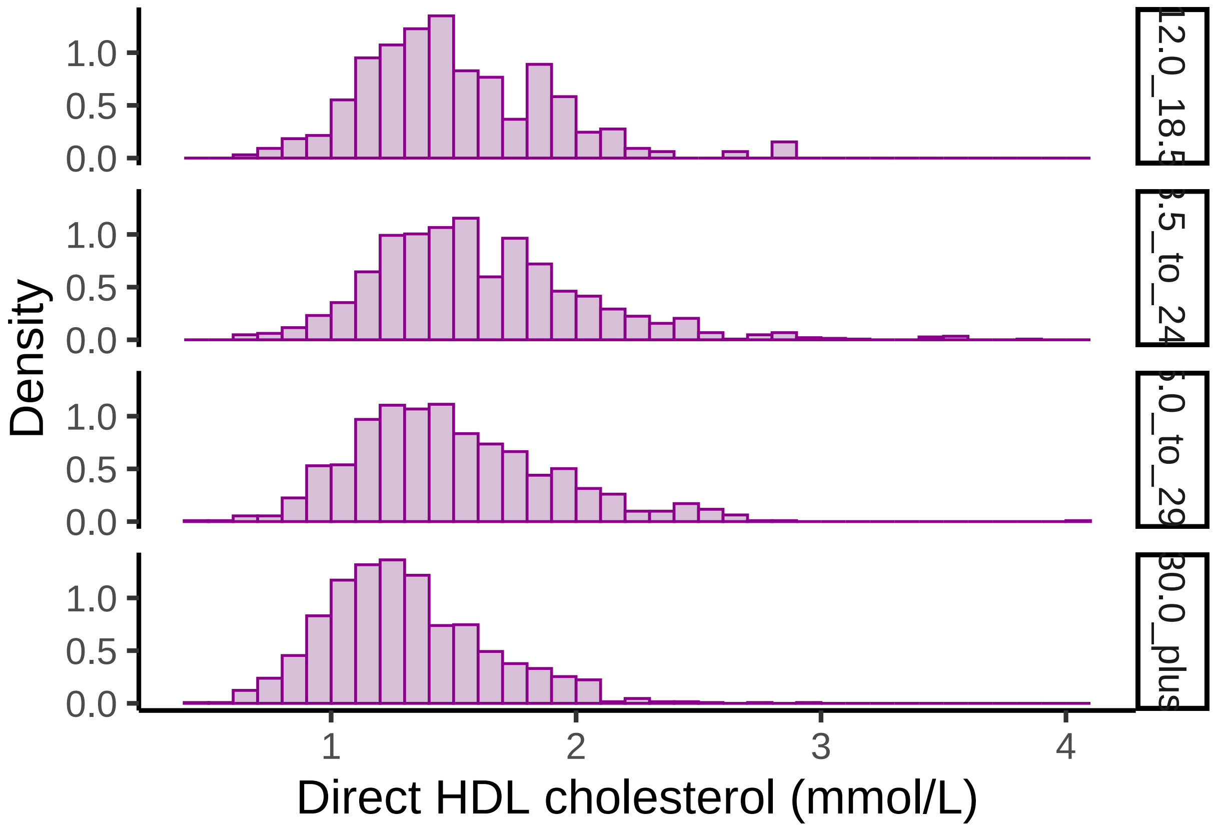
```
NHANES %>%
  filter(Gender == "female") %>%
  drop_na(DirectChol) %>%
  ggplot(aes(x=DirectChol)) +
  geom_histogram(aes(y = after_stat(density)),
                 binwidth = 0.1,
                 boundary = 1,
                 col = "darkmagenta",
                 fill = "thistle") +
  ylab("Density") +
  xlab("Direct HDL cholesterol (mmol/L)") +
  theme_classic(base_size = 18)
```



If you want to separate a histogram into multiple histograms, with each representing the observations for one category of a variable, you can use the **facet_grid()** function. Let's look at the distribution of DirectChol again, but separately for each BMI_WHO value.

First we will add BMI_WHO to the **drop_na()** function to remove any individual without a BMI_WHO category. Then we can add the **facet_grid()** function, specifying the **rows** argument as the BMI_WHO variable, so that the plots will be organised in a single column, one below the other.

```
NHANES %>%
  filter(Gender == "female") %>%
  drop_na(DirectChol, BMI_WHO) %>%
  ggplot(aes(x=DirectChol)) +
  geom_histogram(aes(y = after_stat(density)),
               binwidth = 0.1,
               boundary = 1,
               col = "darkmagenta",
               fill = "thistle") +
  facet_grid(rows = vars(BMI_WHO)) +
  ylab("Density") +
  xlab("Direct HDL cholesterol (mmol/L)") +
  theme_classic(base_size = 18)
```



Interpretation

When you have plotted a histogram you are satisfied with, you should make use of the information it provides. The main use of a histogram is to display the distribution of the sample data. You can comment on whether the distribution looks normally distributed, skewed, or uniform. It's also possible to comment on where the centre of the data is and how spread out the data is. If you are comparing the distribution of a variable between different categories, you can compare the shape of the distributions, and their centre and spread.

Video lecture 2

Create a boxplot

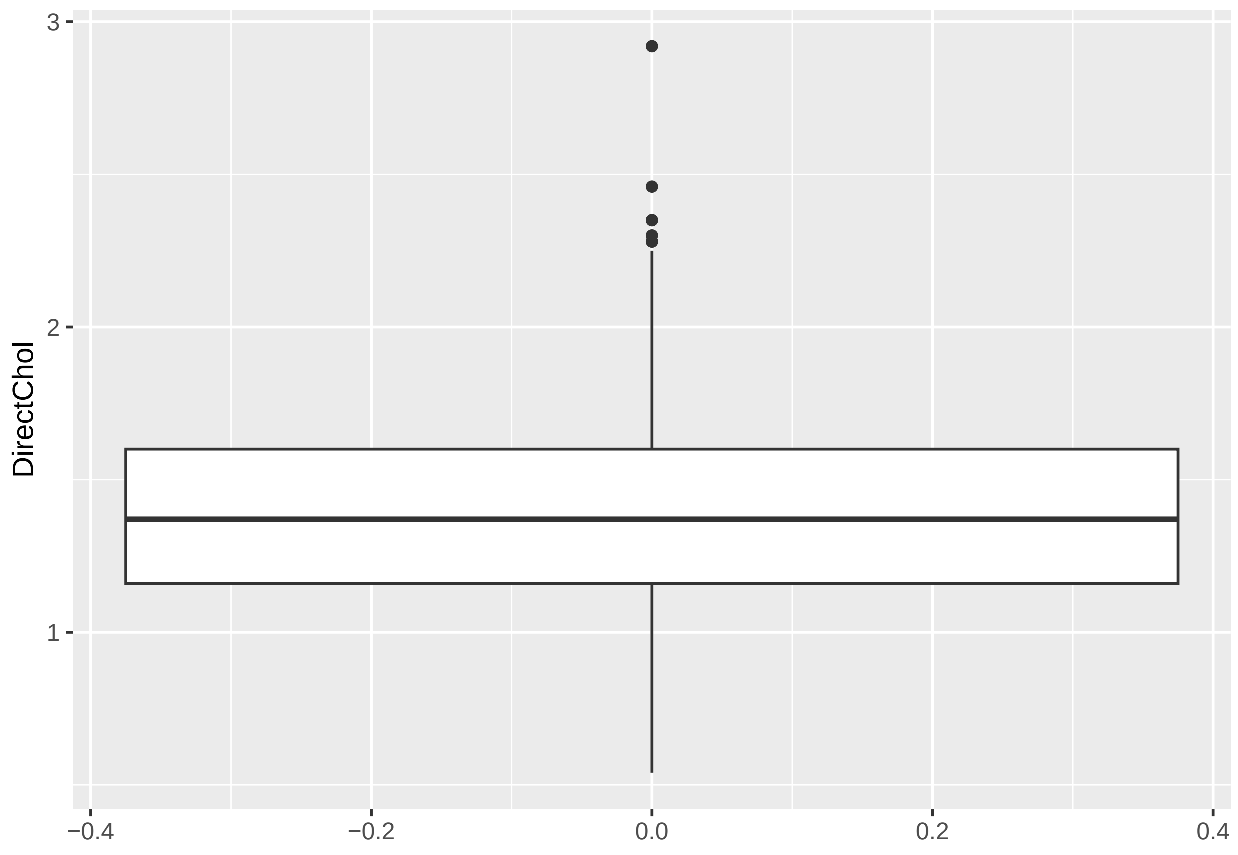
In this video, we will look at another graph that summarises the distribution of a continuous variable: the boxplot. Let's look at the distribution of DirectChol for women between 20 and 25 years old.

First, we take the NHANES dataset, pass it on to the next function using the piping operator **%>%**, and filter it using the **filter()** function to keep only the rows for female individuals whose Age is greater than or equal to 20 and smaller than or equal to 25 years old. Then we remove any row where the DirectChol value is missing using the **drop_na()** function.

This filtered tibble is then used to set a **ggplot()** object indicating that the y-axis represents the DirectChol variable. This is done within the aesthetic mappings **aes()** function, which describes how variables in the data are mapped to visual properties.

Finally, we plot a boxplot using the **geom_boxplot()** function. The default configuration of the ggplot2 boxplot is to show the median, 25th and 75th percentiles in the box part of the plot. The whiskers go to the largest and smallest values, but no further than 1.5 * IQR (the inter-quartile range). Data beyond the whiskers are the outliers and are plotted individually.

```r
NHANES %>%
  filter(Gender=="female" & Age >= 20 & Age <= 25) %>%
  drop_na(DirectChol) %>%
  ggplot(aes(y=DirectChol)) +
  geom_boxplot()
```



Customise our boxplot

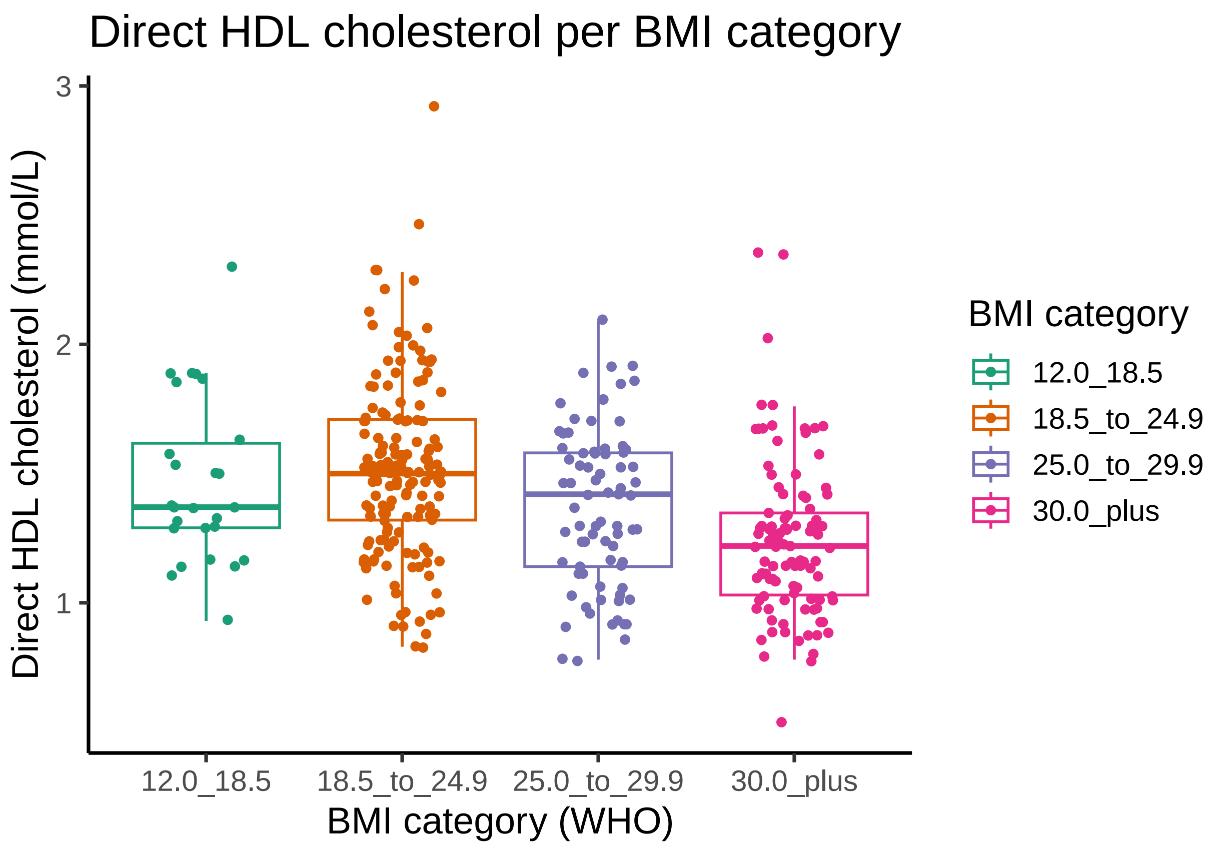
Now let's split this boxplot to look at the distribution separately for each category of BMI_WHO. First we will add BMI_WHO to the **drop_na()** function to remove any individual without a BMI_WHO category. Then, we add the x argument to the **aes()** function inside the **geom_boxplot()** function to indicate that the x-axis should represent the BMI_WHO variable. This will plot the boxplots for each BMI category along the x axis.

To differentiate the categories using colours, we add the **color** argument to the **aes()** function inside the **geom_boxplot()** function and set it to BMI_WHO. We pick a colour palette called "Dark2" using the **scale_color_brewer()** function instead of the default colours. You can explore the different palettes available on this page: ColorBrewer: Color Advice for Maps (colorbrewer2.org).

To have more information on the distribution of the variable, we can add the data points on the boxplots with some jitter to reduce overlap between points using the **geom_jitter()** function. The **shape** argument determines the shape of the dots, as described in this page (Point Shape Options in ggplot - Albert's Blog (albertkuo.me)), and the **position** argument determines how spread out the data points will be. Since we are now displaying all the data points, we should remove the outliers, which are displayed by default, to avoid plotting the outliers twice. We can do this by adding the **outlier.shape** argument to the **geom_boxplot()** function and setting it to NA. Before plotting the data points of a dataset, you should check how many data points will be plotted, as it might take a very long time, become unreadable or produce a very large file if you have a very large number of data points.

We can then use the **labs()** function to change the labels of the x- and y-axes, to add a title and a legend title. Finally, we can use the same theme as for our histogram, by adding the **theme_classic()** function with a **base_size** argument of 16.

```
NHANES %>%
  filter(Gender=="female" & Age >= 20 & Age <= 25) %>%
  drop_na(DirectChol, BMI_WHO) %>%
  ggplot(aes(x=BMI_WHO, y=DirectChol, color=BMI_WHO)) +
  geom_boxplot(outlier.shape = NA) +
  scale_color_brewer(palette="Dark2") +
  geom_jitter(shape=16, position=position_jitter(0.2))+
  labs(title = 'Direct HDL cholesterol per BMI category',
       y="Direct HDL cholesterol (mmol/L)",
       x="BMI category (WHO)",
       col="BMI category") +
  theme_classic(base_size = 14)
```



Interpretation

A boxplot summarises the distribution of the data, but doesn't give the shape of the distribution. You can comment on where the centre is, how spread out the data is, and if the spread is even on either side of the median, and mention any outliers.

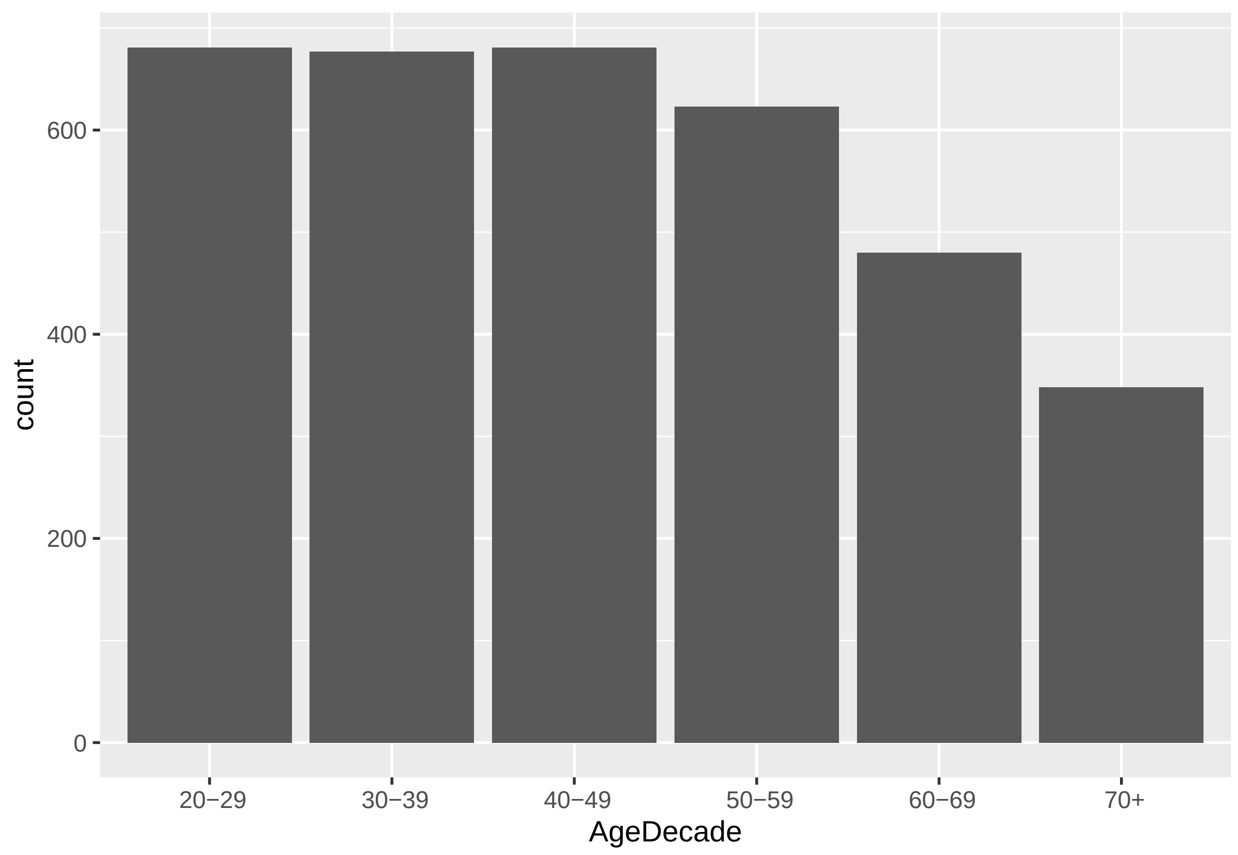
Create a bar chart

A variable is categorical if it can take only one of a small set of values. In R, categorical variables are usually saved as factors or character vectors. If you have a categorical variable in your dataset, a bar chart can be used to look at its distribution. We will look at the distribution of AgeDecade for women who are 20 years old or more.

First, we take the NHANES dataset, pass it on to the next function using the piping operator **%>%**, and filter it using the **filter()** function to keep only the rows for female individuals whose Age is greater than or equal to 20 years old. Then we remove any row where the AgeDecade value is missing using the **drop_na()** function.

This filtered tibble is then used to set a **ggplot()** object indicating that the x-axis represents the AgeDecade variable. This is done within the aesthetic mappings **aes()** function. Finally, we plot a bar plot using the **geom_bar()** function. The height of the bars represents the number of observations for each category of the variable displayed on the x-axis.

```r
NHANES %>%
  filter(Gender=="female" & Age >= 20) %>%
  drop_na(AgeDecade) %>%
  ggplot(aes(x = AgeDecade)) +
  geom_bar()
```



Customise our bar chart

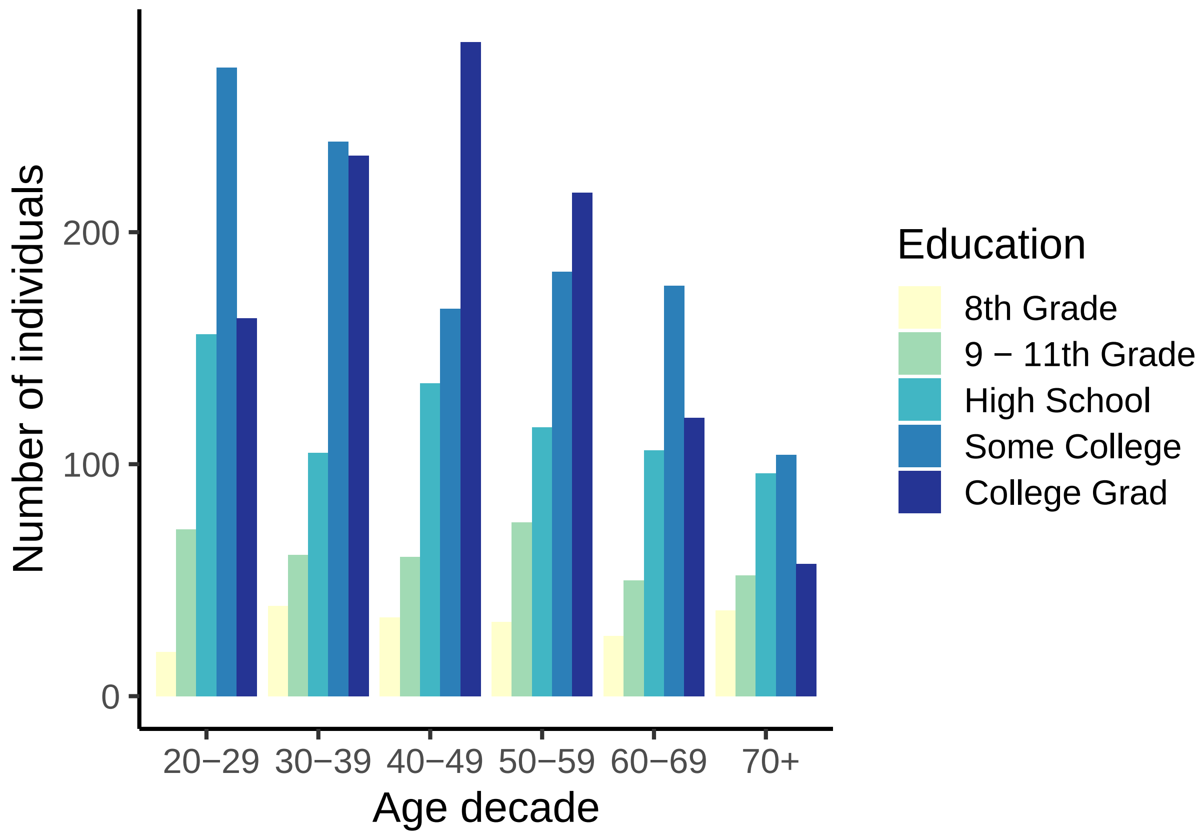
We can add more information to this plot by making adjacent coloured bar charts. Let's add the Education categorical variable here. First we will add Education to the **drop_na()** function to remove any individual without an Education category.

To differentiate the Education categories using colours, we add the **fill** argument to the **aes()** function inside the **ggplot()** function and set it to Education. We pick a colour palette called "YlGnBu" using the **scale_fill_brewer()** function instead of the default colours. You can explore the different palettes available on this page: ColorBrewer: Color Advice for Maps (colorbrewer2.org).

Then we use the position argument inside the **geom_bar()** function and set it to "dodge". This will plot the bars for each Education category side-by-side.

We can then use the **labs()** function to change the labels of the x- and y-axes, to add a title and a legend title. Finally, we can use the same theme as for our histogram, by adding the **theme_classic()** function with a **base_size** argument of 16.

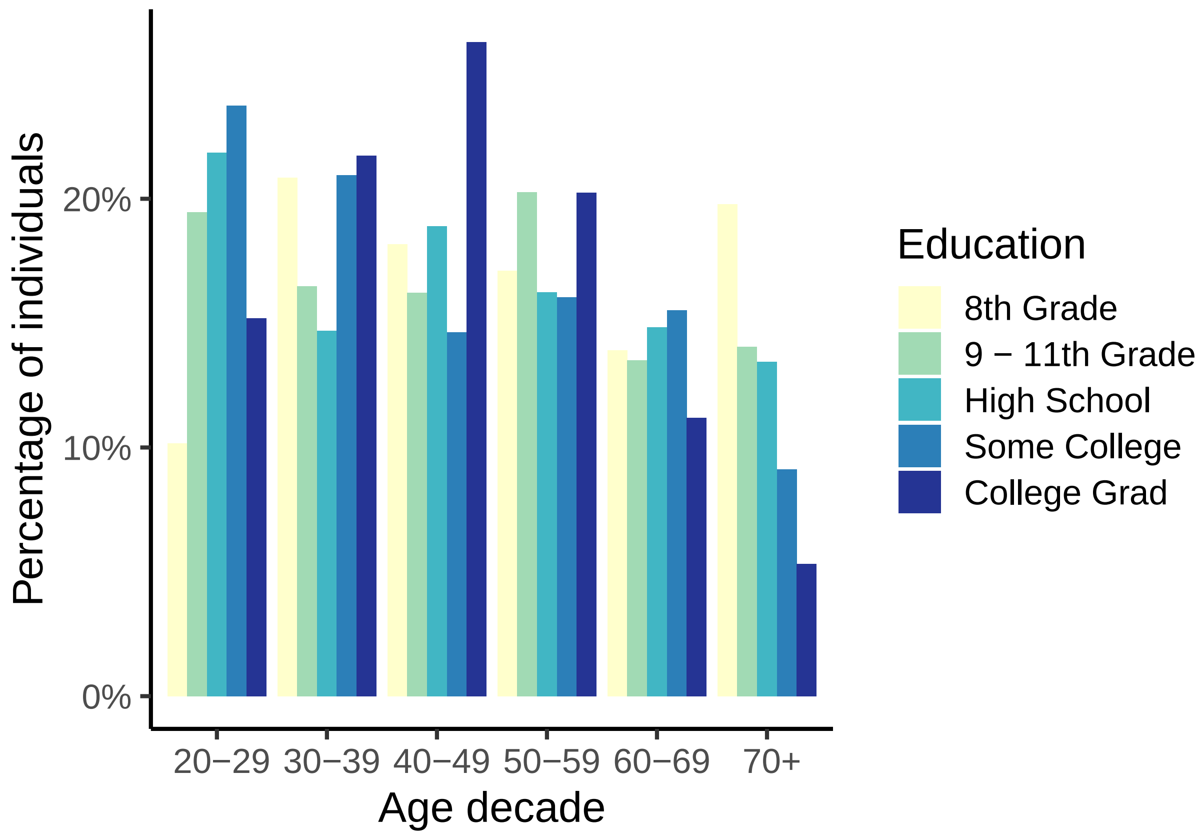
```r
NHANES %>%
  filter(Gender=="female" & Age >= 20) %>%
  drop_na(AgeDecade,Education) %>%
  ggplot(aes(x = AgeDecade, fill = Education)) +
  geom_bar(position='dodge') +
  scale_fill_brewer(palette="YlGnBu") +
  labs(x = "Age decade",
       y = "Number of individuals") +
  theme_classic(base_size = 16)
```



Finally, we can transform this bar chart into a percentage bar chart. To do this, we add the **group** argument inside the **ggplot()** function and set it to Education. This tells ggplot to calculate the percentages separately for each Education category. We tell ggplot to use percentages by adding the **y** argument to the **aes()** function within the **geom_bar()** function and setting it to ..prop.. , this is a shortcut to calculate the proportions and use them directly in the **geom_bar()** function. We also change the scale of the y-axis to a percentage scale by adding the **scale_y_continuous()** function, with the **labels** argument set to scales::percent.

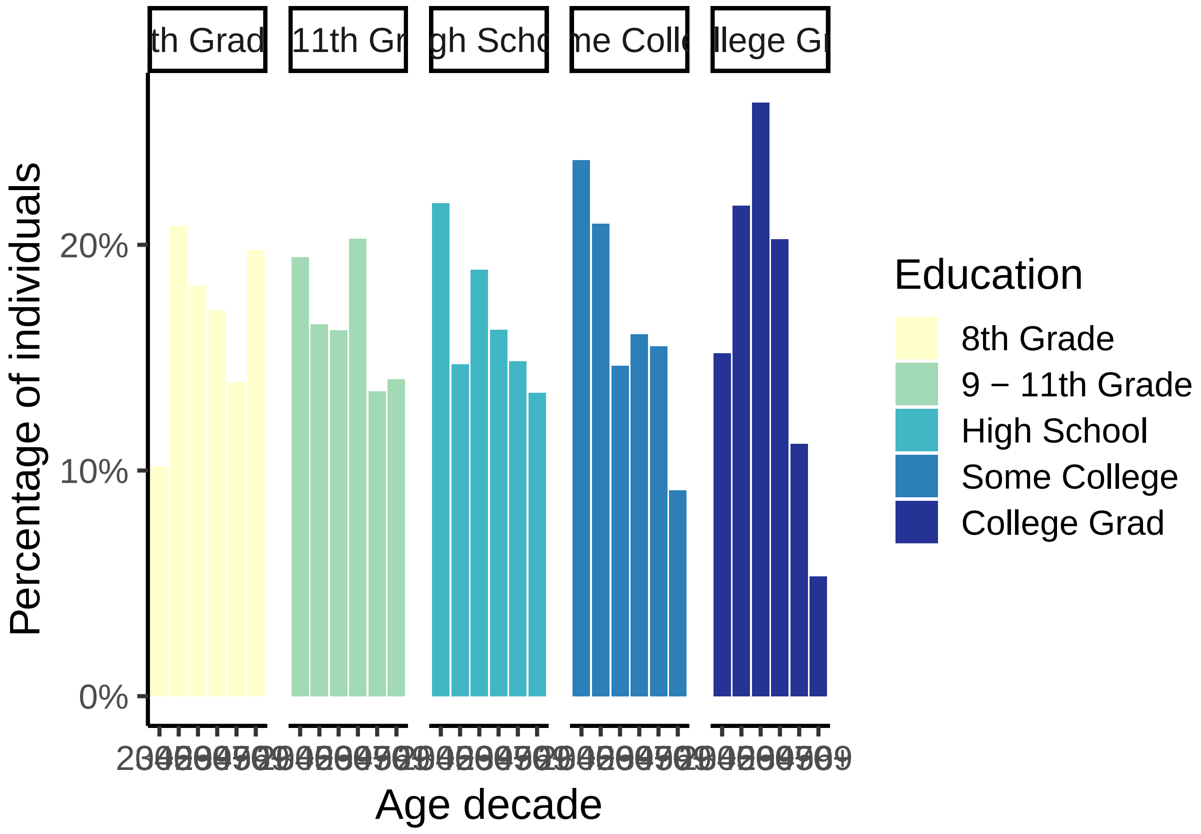
```
NHANES %>%
  filter(Gender=="female" & Age >= 20) %>%
  drop_na(AgeDecade,Education) %>%
  ggplot(aes(x = AgeDecade, group = Education, fill = Education)) +
  geom_bar(aes(y = ..prop.. ), position='dodge') +
  scale_fill_brewer(palette="YlGnBu") +
  labs(x = "Age decade",
       y = "Percentage of individuals") +
  theme_classic(base_size = 16) +
  scale_y_continuous(labels = scales::percent)
```

```
## Warning: The dot-dot notation (`..prop..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(prop)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



We can also display the bars in different side-by-side plots by adding the **facet_grid()** function and setting the **cols** argument to Education.

```r
NHANES %>%
  filter(Gender=="female" & Age >= 20) %>%
  drop_na(AgeDecade,Education) %>%
  ggplot(aes(x = AgeDecade, group = Education, fill = Education)) +
  geom_bar(aes(y = ..prop.. )) +
  scale_fill_brewer(palette="YlGnBu") +
  labs(x = "Age decade",
       y = "Percentage of individuals") +
  theme_classic(base_size = 16) +
  scale_y_continuous(labels = scales::percent) +
  facet_grid(cols = vars(Education))
```



Interpretation

A bar chart shows the frequency distribution of a categorical variable. You can describe any pattern along the categories of your categorical variable. If you have several bars, you can also compare this pattern between the categories of the second variable. Percentage bar charts can show patterns that might be obscured by low numbers in some categories when using the counts of observations.

Video lecture 3

Covariation between variables

Once we have a good idea of the distribution of our variables, we can start looking at the relationship between variables. You might have noticed that we already did this when we added an extra categorical variable to our histogram, boxplot and bar chart.

Categorical variables

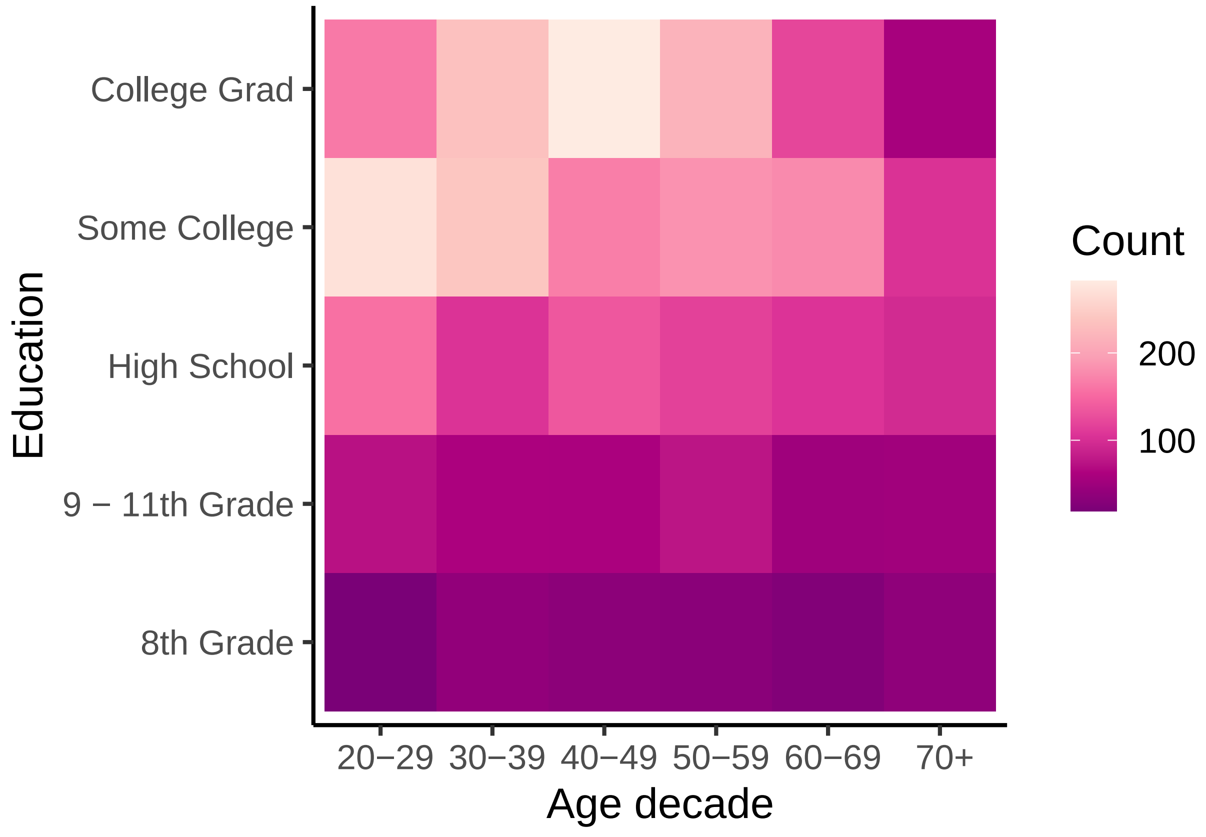
To visualise the covariation between two categorical variables, it is best to use contingency tables to display the number of observations for each combination of the levels of the two variables. These will be covered later. But you can also use the **geom_tile()** function to create a tile plot.

Let's look at the covariation between AgeDecade and Education. First, we take the NHANES dataset, pass it on to the next function using the piping operator **%>%**, and filter it using the **filter()** function to keep only the rows for female individuals. Then we remove any row where the AgeDecade or Education value is missing using the **drop_na()** function.

This filtered tibble is then used in the **count()** function to calculate the number of individuals in each combination of Education and AgeDecade values.

We then set a **ggplot()** object indicating that the x-axis represents the AgeDecade variable and the y-axis represents the Education variable. This is done within the aesthetic mappings **aes()** function. Finally, we plot a tile plot using the **geom_tile()** function, setting an additional argument called **fill** within the **aes()** function. This is set to n, which is the output of the **count()** function containing the counts for each combination. We pick a colour palette called "RdPu" using the **scale_fill_distiller()** function instead of the default colours. The resulting plot has coloured tiles for each combination, where a lighter colour indicates a higher count.

```r
NHANES %>%
  filter(Gender=="female") %>%
  drop_na(AgeDecade, Education) %>%
  count(AgeDecade, Education) %>%
  ggplot(aes(x = AgeDecade, y = Education)) +
  geom_tile(aes(fill=n)) +
  labs(x = "Age decade",
       y = "Education",
       fill = "Count") +
  scale_fill_distiller(palette = "RdPu") +
  theme_classic(base_size = 16)
```



Interpretation

The tile plot gives an indication of the frequency of different combinations of variable categories. The differences in frequencies can indicate patterns of association between two categorical variables, it is helpful to comment on these patterns using the tile plot (or contingency tables).

Continuous variables

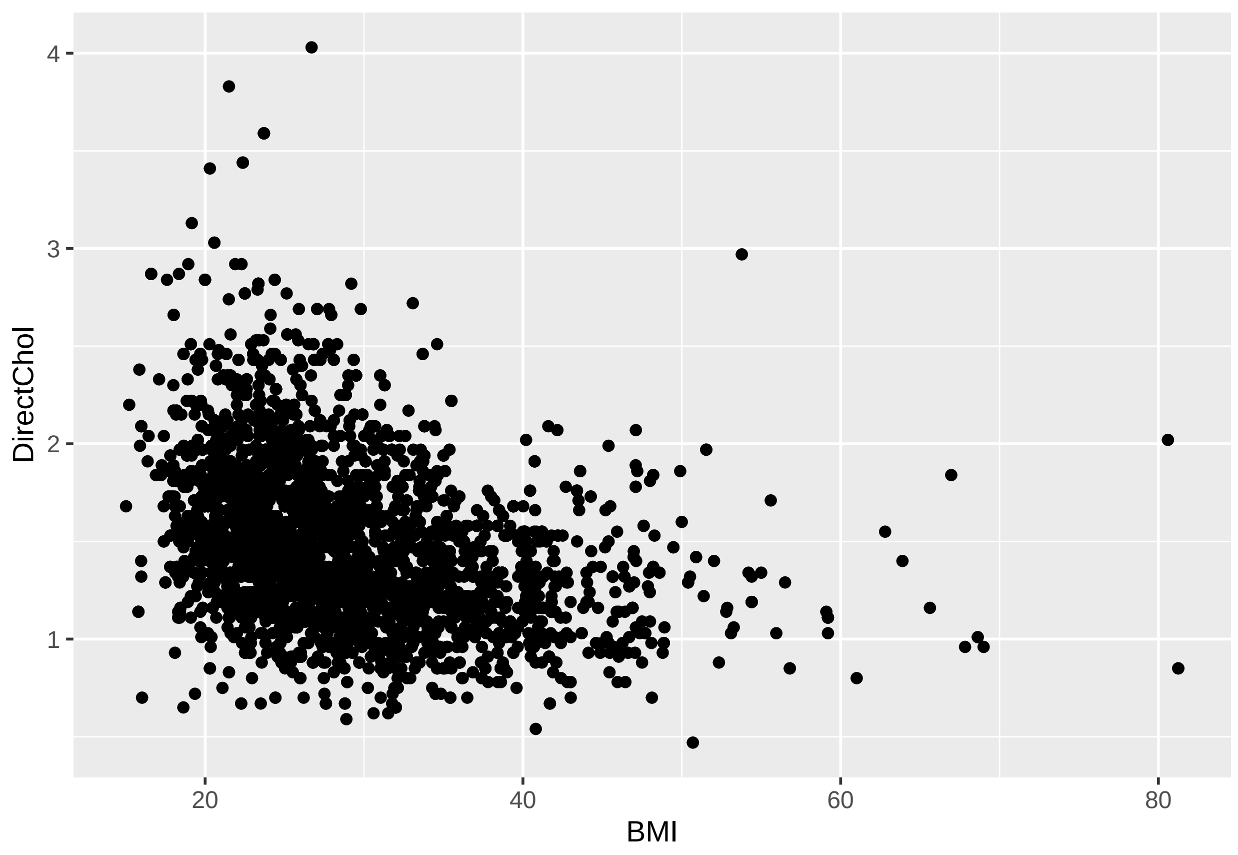
Initialise a scatter plot Now we will look at the covariation between two continuous variables from the NHANES dataset, BMI and DirectChol. This time, we will initialise a plot using the ggplot2 package. This code creates an object named **init**, then takes the NHANES dataset, filters it to keep only female individuals aged 18 and over, removes any row where either variable is missing, and then sets a **ggplot()** object with plot attributes within the **aes()** function indicating the x (BMI) and y (DirectChol) variables.

```
init <- NHANES %>%
      drop_na(BMI, DirectChol) %>%
      filter(Gender=="female" & Age >= 18) %>%
      ggplot(aes(x = BMI, y = DirectChol))
```

The **<-** symbol is an assignment operator, so the output of the code to the right-hand side of it gets saved into an object named after the left-hand side of the **<-**. Saving this initial **ggplot()** object saves typing the same code multiple times when trying different ways of plotting the data.

Create a scatterplot Now we can create the actual scatter plot. This code takes the initialised **ggplot()** object we just created and creates a scatter plot using the **geom_point()** function to plot the paired values of BMI and DirectChol for each individual.

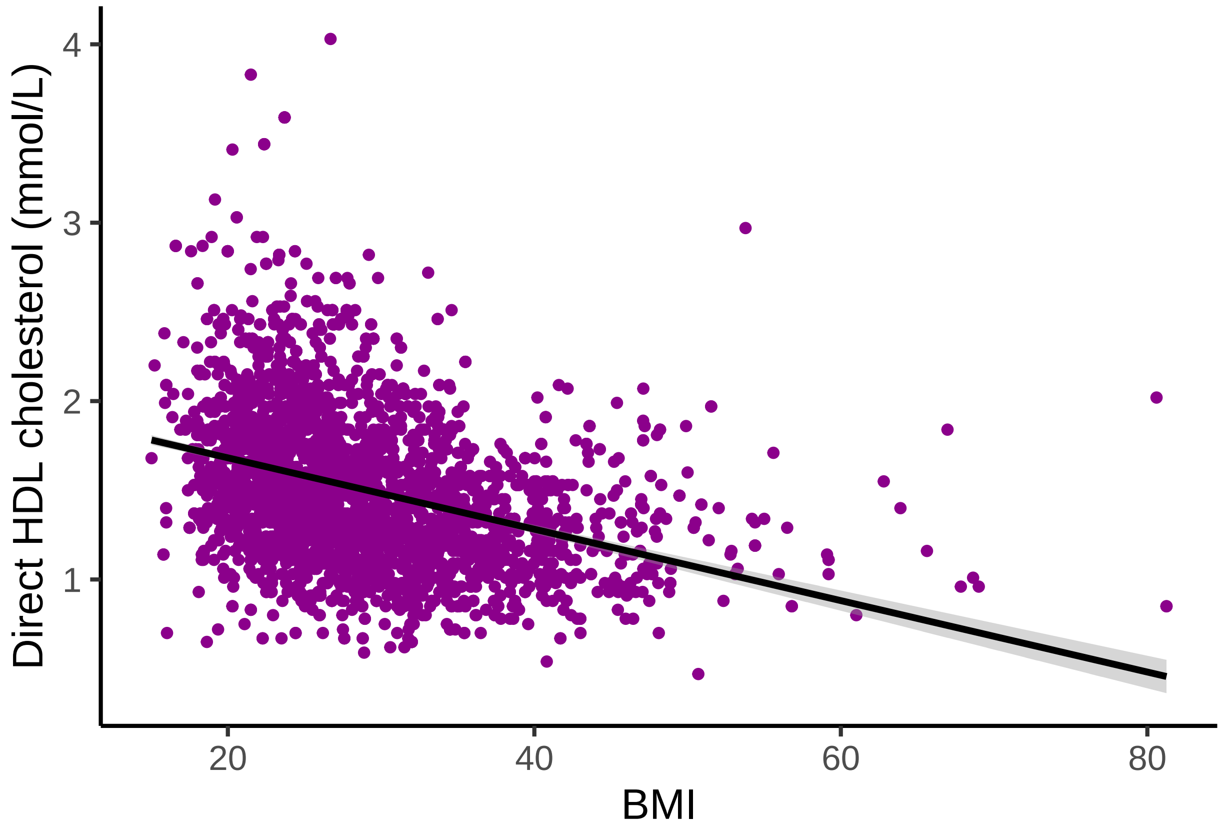
```
init +
geom_point()
```



Customise our scatterplot Now we can customise our plot! We add the **col** argument within the **geom_point()** function to colour the dots in darkmagenta. We add a trend line with 95% confidence region using the **geom_smooth()** function, adding the **col** argument to make the line black, and the **lwd** to increase the line width. Finally, we add axis labels using the **xlab()** and **ylab()** functions, and use the same **theme_classic()** function and **base_size** argument as in previous plots.

```
init +
geom_point(col = "darkmagenta") +
geom_smooth(method=lm, col = "black", lwd = 1.2) +
xlab("BMI") +
ylab("Direct HDL cholesterol (mmol/L)") +
theme_classic(base_size = 16)
```

```
## ‘geom_smooth()‘ using formula = ’y ~ x’
```



Interpretation

When looking at a scatterplot, you can comment on the direction and the strength of the relationship between two continuous variables. You can also describe the relationship as linear or non-linear, as this affects subsequent analysis. There may be data points that don't follow the general trend, these are worth mentioning and investigating further.

Optional material

Visualising missing data

An important thing to explore in a new dataset is the amount of missing values. Missing values should be represented by **NA** in a clean dataset. We can look at this in different ways. Please note that we don't expect you to understand the following R code at this stage as it is quite complex. Let's make a bar chart with the percentage of missing data for each of the variables we looked at so far.

```
missing.values <- NHANES %>%
  select(Age, Weight, Height, BMI, PhysActive, DirectChol, BMI_WHO, AgeDecade, Education, Gender) %>%
  gather(key = "key", value = "val") %>%
  mutate(isna = is.na(val)) %>%
  group_by(key) %>%
  mutate(total = n()) %>%
  group_by(key, total, isna) %>%
```

```r
  summarise(num.isna = n()) %>%
  mutate(pct = num.isna / total * 100)
```

```
## Warning: attributes are not identical across measure variables; they will be
## dropped
```

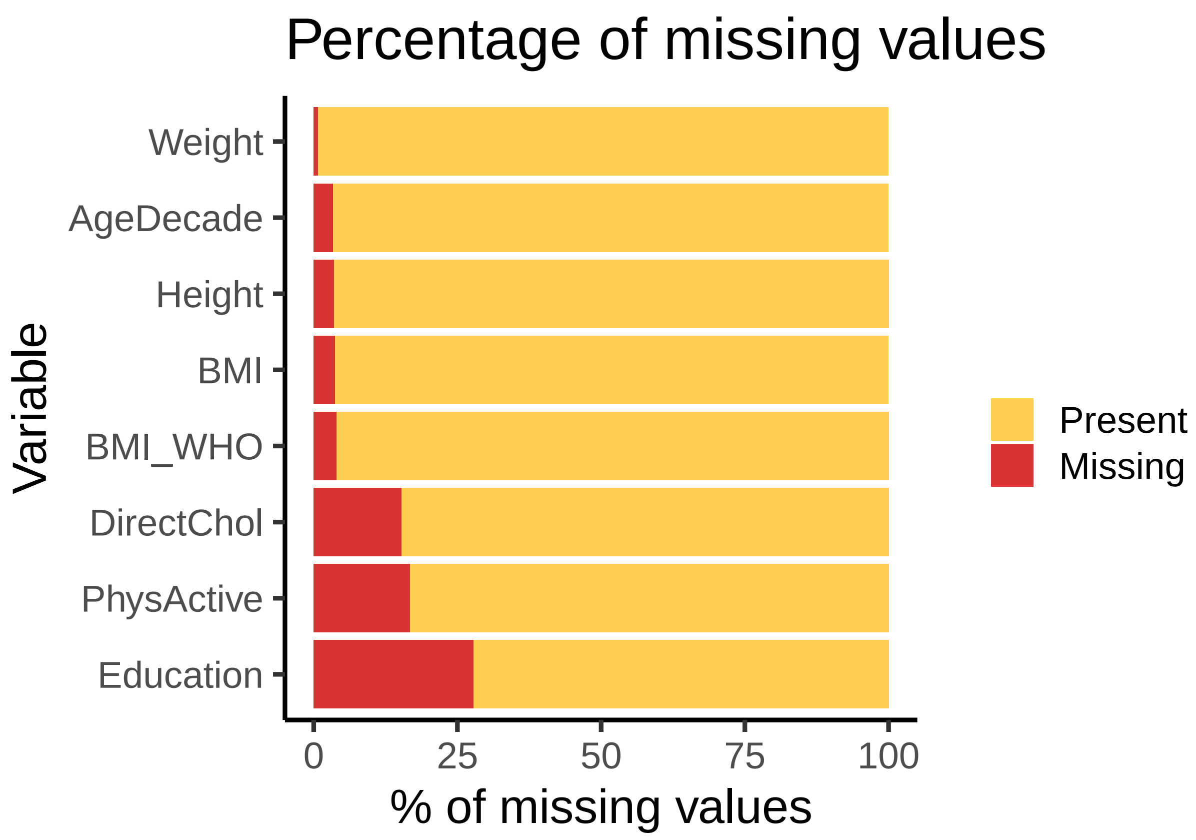
```
## `summarise()` has grouped output by 'key', 'total'. You can override using the
## `.groups` argument.
```

```r
levels <-
    (missing.values  %>% filter(isna == T) %>% arrange(desc(pct)))$key

percentage.plot <- missing.values %>%
      ggplot() +
        geom_bar(aes(x = reorder(key, desc(pct)),
                     y = pct, fill=isna),
                 stat = 'identity', alpha=0.8) +
      scale_x_discrete(limits = levels) +
      scale_fill_manual(name = "",
                        values = c('goldenrod1', 'red3'), labels = c("Present", "Missing")) +
      coord_flip() +
      labs(title = "Percentage of missing values", x =
             'Variable', y = "% of missing values") +
  theme_classic(base_size = 18)

percentage.plot
```

```
## Warning: Removed 2 rows containing missing values (`position_stack()`).
```



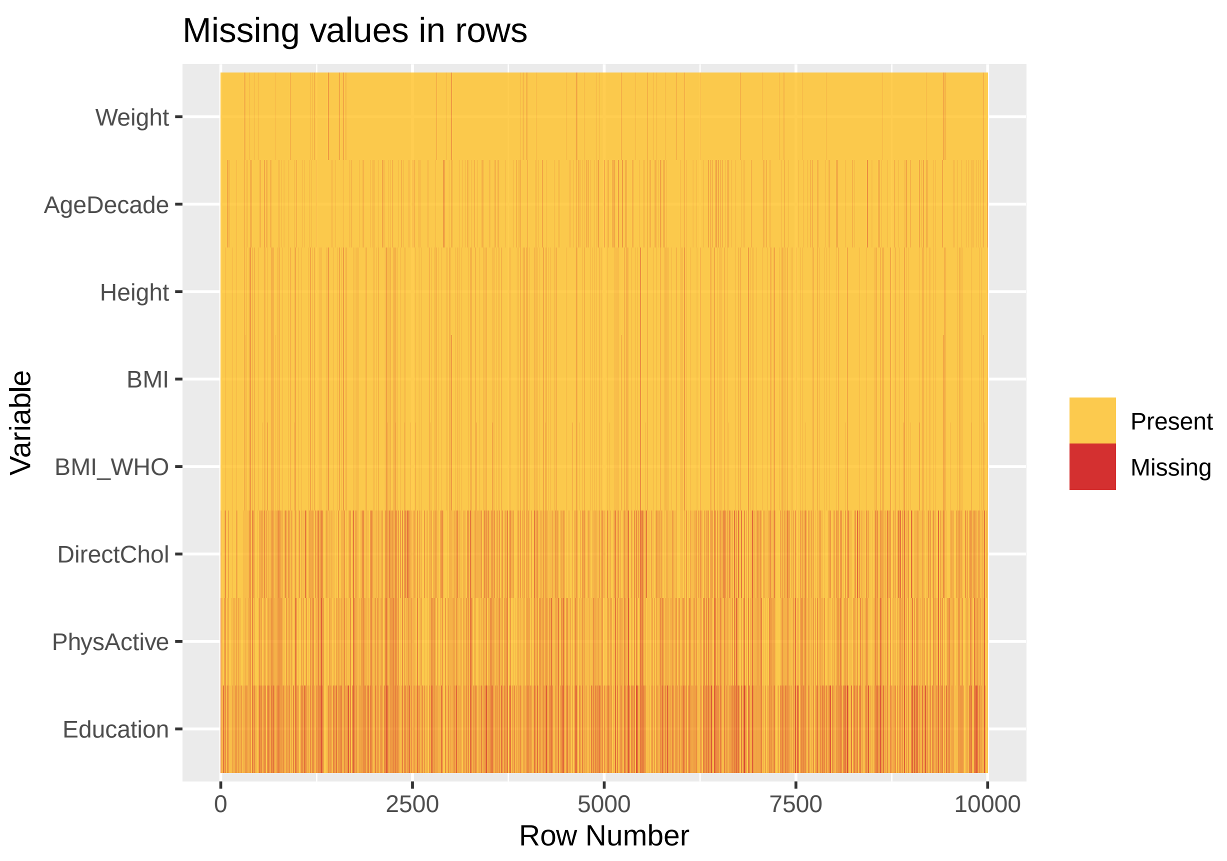
Now for each variable, we can also look at which rows are missing values by creating a row plot using the **geom_raster()** function.

```
row.plot <- NHANES %>%
  select(Age, Weight, Height, BMI, PhysActive, DirectChol, BMI_WHO, AgeDecade, Education, Gender) %>%
  mutate(id = row_number()) %>%
  gather(-id, key = "key", value = "val") %>%
  mutate(isna = is.na(val)) %>%
  ggplot(aes(key, id, fill = isna)) +
    geom_raster(alpha=0.8) +
    scale_fill_manual(name = "",
        values = c('goldenrod1', 'red3'),
        labels = c("Present", "Missing")) +
    scale_x_discrete(limits = levels) +
    labs(x = "Variable",
           y = "Row Number", title = "Missing values in rows") +
    coord_flip()
```

```
## Warning: attributes are not identical across measure variables; they will be
## dropped
```

```
row.plot
```

```
## Warning: Removed 20000 rows containing missing values (‘geom_raster()‘).
```


Missing values in rows
Weight
AgeDecade
Height
BMI
BMI_WHO
DirectChol
PhysActive
Education
Variable
0
2500
5000
7500
10000
Row Number
Present
Missing