Data manipulation

Contents

Introduction

In this practical, we will learn how to visualise data after we have cleaned up our datasets using the `dplyr` verbs from the previous practical. `filter()`, `arrange()`, `mutate()`, `select()`, `summarise()`. For the visualisations, we will be using a package that implements the grammar of graphics: `ggplot2`.

Don't forget to open the project file `03_Data_visualisation.Rproj` and to create your `.Rmd` or `.R` file to work in.

```
library(ISLR)
library(tidyverse)
```

An excellent reference manual for `ggplot` can be found on the tidyverse website: https://ggplot2.tidyverse.org/reference/

What is ggplot

Plots can be made in `R` without the use of `ggplot` using `plot()`, `hist()` or `barplot()` and related functions. Here is an example of each on the `Hitters` dataset from `ISLR`:

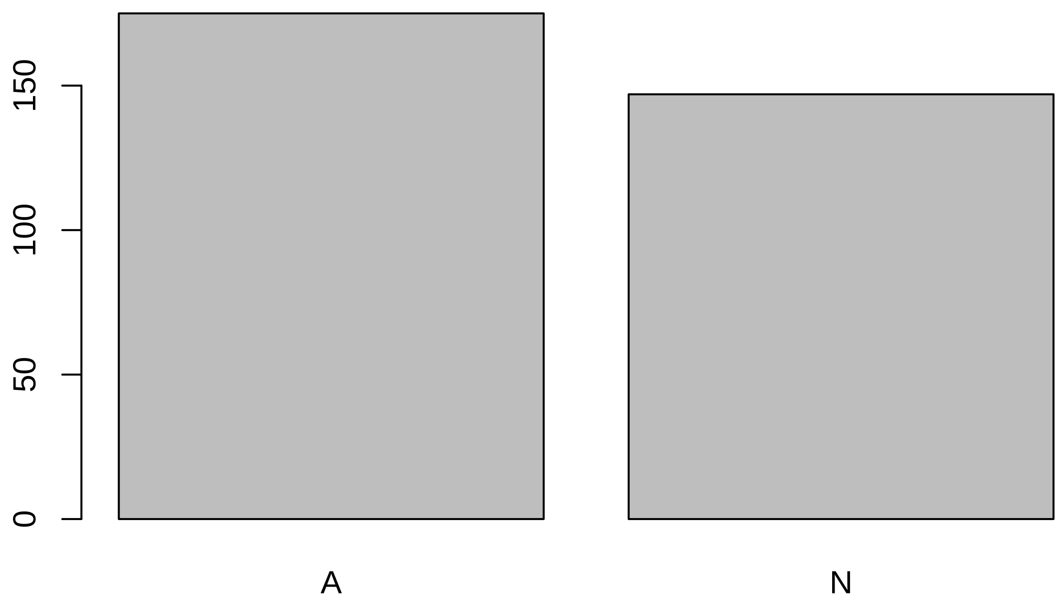
```
# Get an idea of what the Hitters dataset looks like
head(Hitters)
```

```
##                   AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits
## -Andy Allanson      293   66     1   30  29    14     1    293    66
## -Alan Ashby         315   81     7   24  38    39    14   3449   835
## -Alvin Davis        479  130    18   66  72    76     3   1624   457
## -Andre Dawson       496  141    20   65  78    37    11   5628  1575
## -Andres Galarraga   321   87    10   39  42    30     2    396   101
```

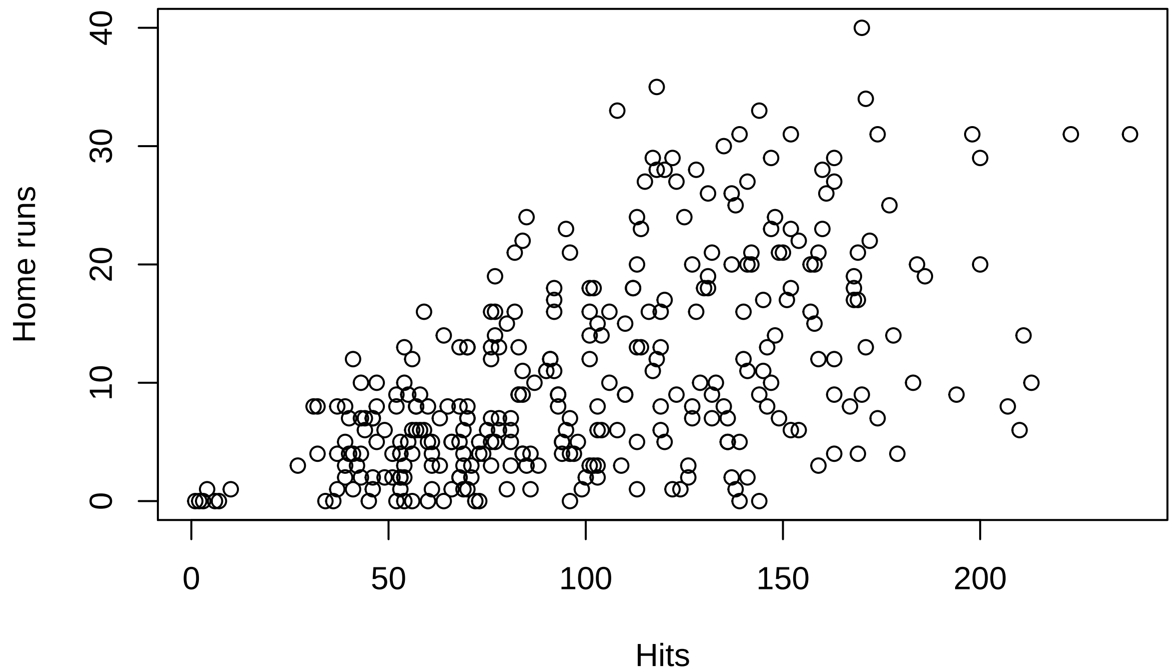
```
## -Alfredo Griffin    594  169    4   74  51    35    11   4408  1133
##                    CHmRun CRuns CRBI CWalks League Division PutOuts Assists
## -Andy Allanson          1    30   29     14      A        E     446      33
## -Alan Ashby            69   321  414    375      N        W     632      43
## -Alvin Davis           63   224  266    263      A        W     880      82
## -Andre Dawson         225   828  838    354      N        E     200      11
## -Andres Galarraga      12    48   46     33      N        E     805      40
## -Alfredo Griffin       19   501  336    194      A        W     282     421
##                    Errors Salary NewLeague
## -Andy Allanson         20     NA         A
## -Alan Ashby            10  475.0         N
## -Alvin Davis           14  480.0         A
## -Andre Dawson           3  500.0         N
## -Andres Galarraga       4   91.5         N
## -Alfredo Griffin       25  750.0         A
```

```r
# histogram of the distribution of salary
hist(Hitters$Salary, xlab = "Salary in thousands of dollars")
```


```r
# barplot of how many members in each league
barplot(table(Hitters$League))
```




```r
# number of career home runs vs 1986 home runs
plot(x = Hitters$Hits, y = Hitters$HmRun,
     xlab = "Hits", ylab = "Home runs")
```



These plots are informative and useful for visually inspecting the dataset, and they each have a specific syntax associated with them. `ggplot` has a more unified approach to plotting, where you build up a plot layer by layer using the + operator:

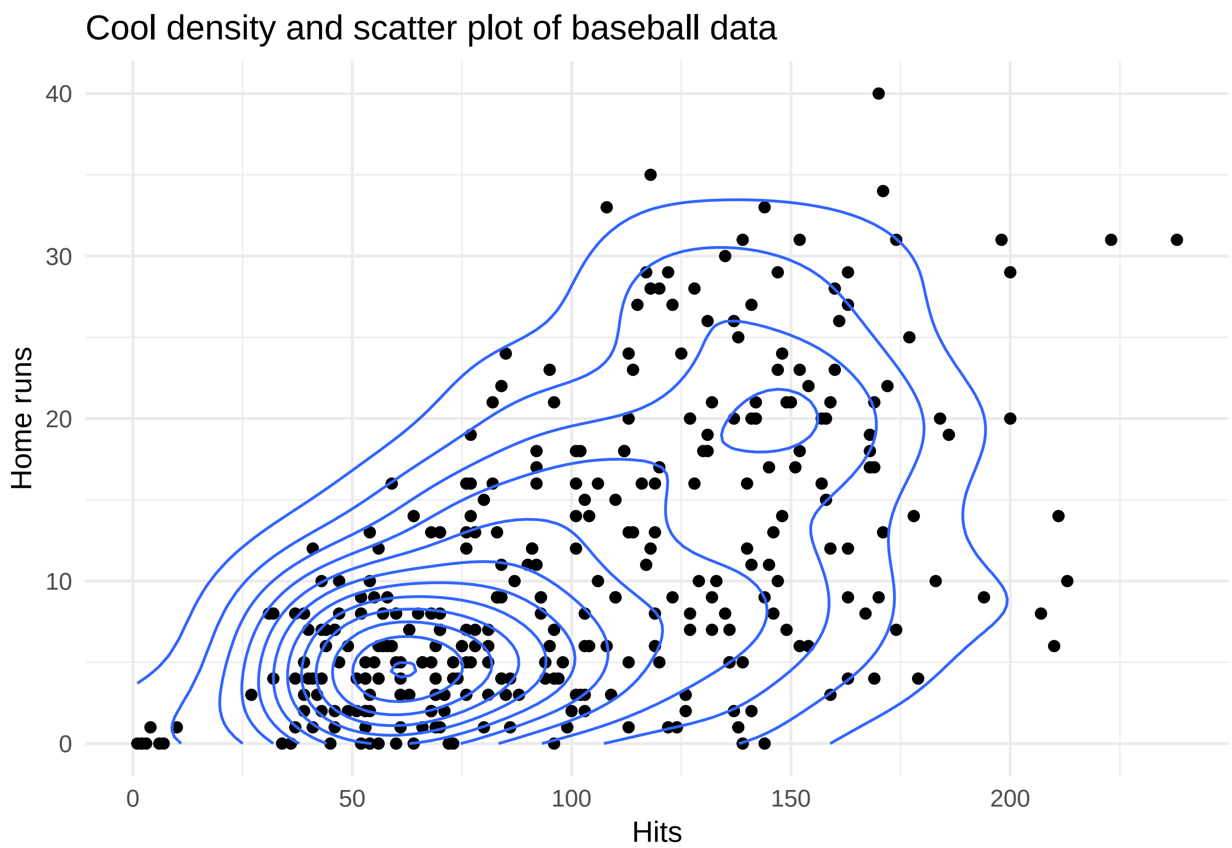
```
homeruns_plot <-
  ggplot(Hitters, aes(x = Hits, y = HmRun)) +
  geom_point() +
  labs(x = "Hits", y = "Home runs")

homeruns_plot
```

As introduced in the lectures, a `ggplot` object is built up in different layers:

1. input the dataset to a `ggplot()` function call
2. construct aesthetic mappings
3. add (geometric) components to your plot that use these mappings
4. add labels, themes, visuals.

Because of this layered syntax, it is then easy to add elements like these fancy density lines, a title, and a different theme:

```
homeruns_plot +
  geom_density_2d() +
  labs(title = "Cool density and scatter plot of baseball data") +
  theme_minimal()
```



In conclusion, `ggplot` objects are easy to manipulate and they force a principled approach to data visualisation. In this practical, we will learn how to construct them.

Name the aesthetics, geoms, scales, and facets of the above visualisation. Also name any statistical transformations or special coordinate systems.

```
# Aesthetics:
#   number of hits mapped to x-position
#   number of home runs mapped to y-position
# Geoms: points and contour lines
# Scales:
#   x-axis: continuous
#   y-axis: continuous
# Facets: None
# Statistical transformations: None
# Special coordinate system: None (just cartesian)
```

Aesthetics and data preparation

The first step in constructing a ggplot is the preparation of your data and the mapping of variables to aesthetics. In the homeruns_plot, we used an existing data frame, the Hitters dataset.

The data frame needs to have proper column names and the types of the variables in the data frame need to be correctly specified. Numbers should be numerics, categories should be factors, and names or identifiers should be character variables. ggplot() *always* expects a data frame, which may feel awfully strict, but it allows for excellent flexibility in the remaining plotting steps.

Run the code below to generate data. There will be three vectors in your environment. Put them in a data frame for entering it in a ggplot() call using either the data.frame() or the tibble() function. Give informative names and make sure the types are correct (use the as.<type>() functions). Name the result gg_students

---

```
set.seed(1234)
student_grade  <- rnorm(32, 7)
student_number <- round(runif(32) * 2e6 + 5e6)
programme      <- sample(c("Science", "Social Science"), 32, replace = TRUE)
```

```
gg_students <- tibble(
  number = as.character(student_number), # an identifier
  grade  = student_grade,                # already the correct type.
  prog   = as.factor(programme)          # categories should be factors.
)

head(gg_students)
```

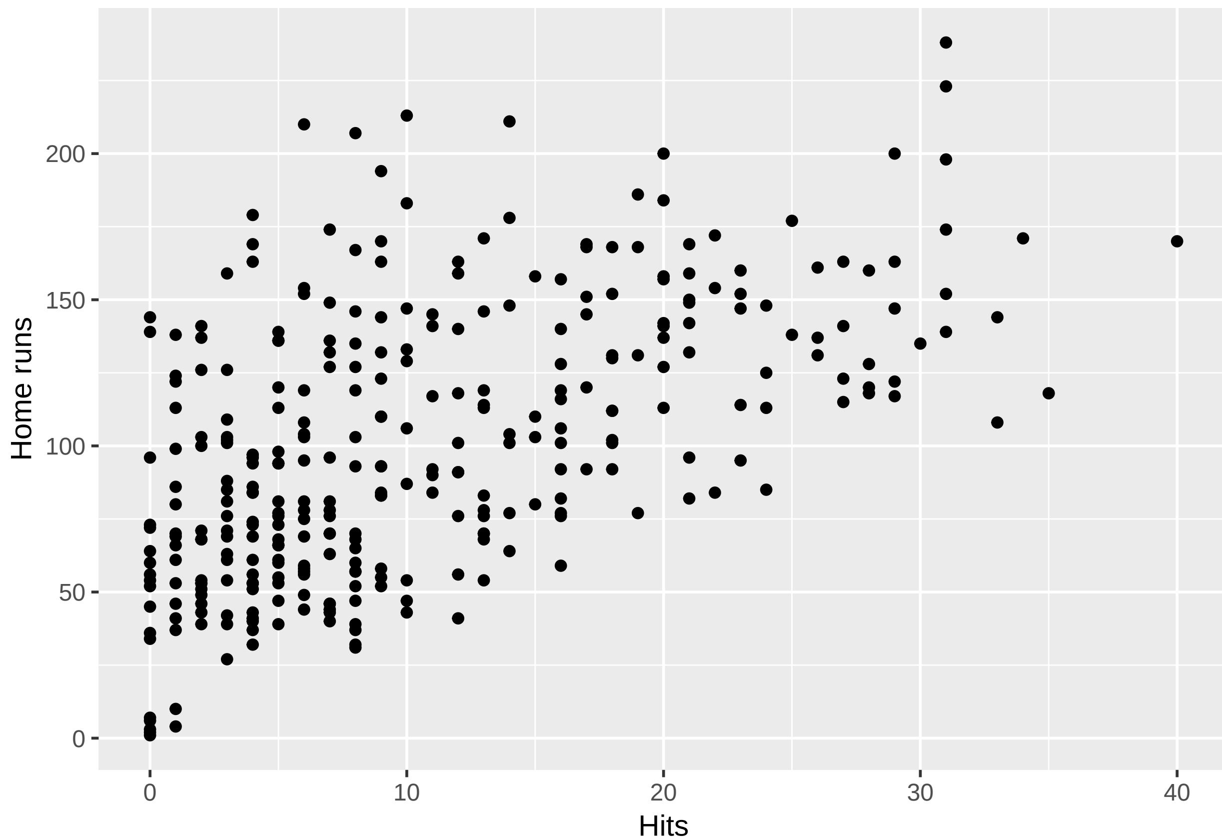
```
## # A tibble: 6 x 3
##   number  grade prog
##   <chr>   <dbl> <fct>
## 1 5478051  5.79 Science
## 2 6412989  7.28 Science
## 3 5616190  8.08 Science
## 4 6017095  4.65 Social Science
## 5 5103293  7.43 Science
## 6 6129140  7.51 Social Science
```

```
# note that if you use data.frame(), you need to set the argument
# stringsAsFactors to FALSE to get student number to be a character.
# tibble() does this by default.
```

Mapping aesthetics is usually done in the main ggplot() call. Aesthetic mappings are the second argument to the function, after the data frame.

Replicate the first `homeruns_plot`, but map the `Hits` to the y-axis and the `HmRun` to the x-axis instead.

```
ggplot(Hitters, aes(x = HmRun, y = Hits)) +
  geom_point() +
  labs(x = "Hits", y = "Home runs")
```

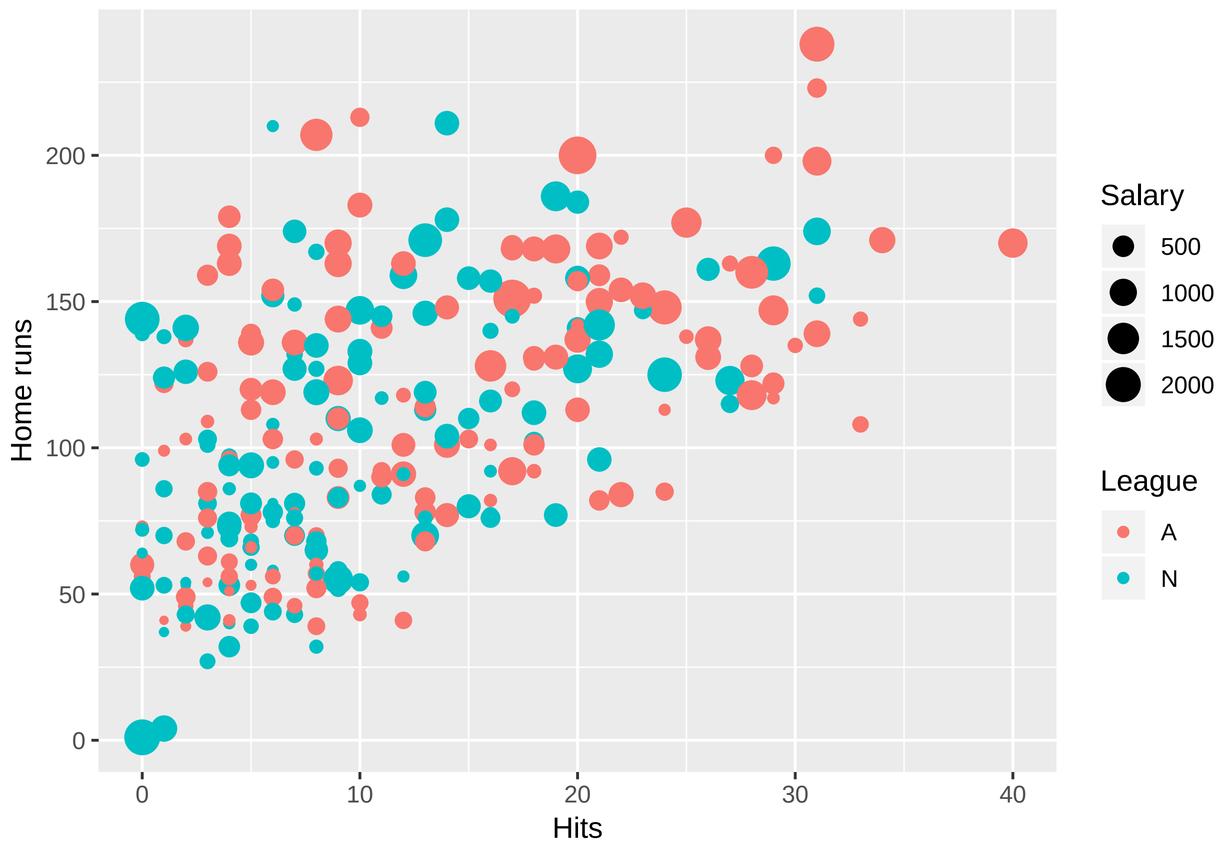


Recreate the same plot once more, but now also map the variable `League` to the `colour` aesthetic and the variable `Salary` to the `size` aesthetic

---

```
ggplot(Hitters, aes(x = HmRun, y = Hits, colour = League, size = Salary)) +
  geom_point() +
  labs(x = "Hits", y = "Home runs")
```

```
## Warning: Removed 59 rows containing missing values (geom_point).
```



Examples of aesthetics are:

- x
- y
- alpha (transparency)
- colour
- fill
- group
- shape
- size
- stroke

Geoms

Up until now we have used two geoms: contour lines and points. The geoms in `ggplot2` are added via the `geom_<geomtype>()` functions. Each geom has a required aesthetic mapping to work. For example, `geom_point()` needs at least and x and y position mapping, as you can read here.

Look at the many different geoms on the reference website.

There are two types of geoms:

- geoms which perform a transformation of the data beforehand, such as `geom_density_2d()` which calculates contour lines from x and y positions.
- geoms which do not transform data beforehand, but use the aesthetic mapping directly, such as `geom_point()`.

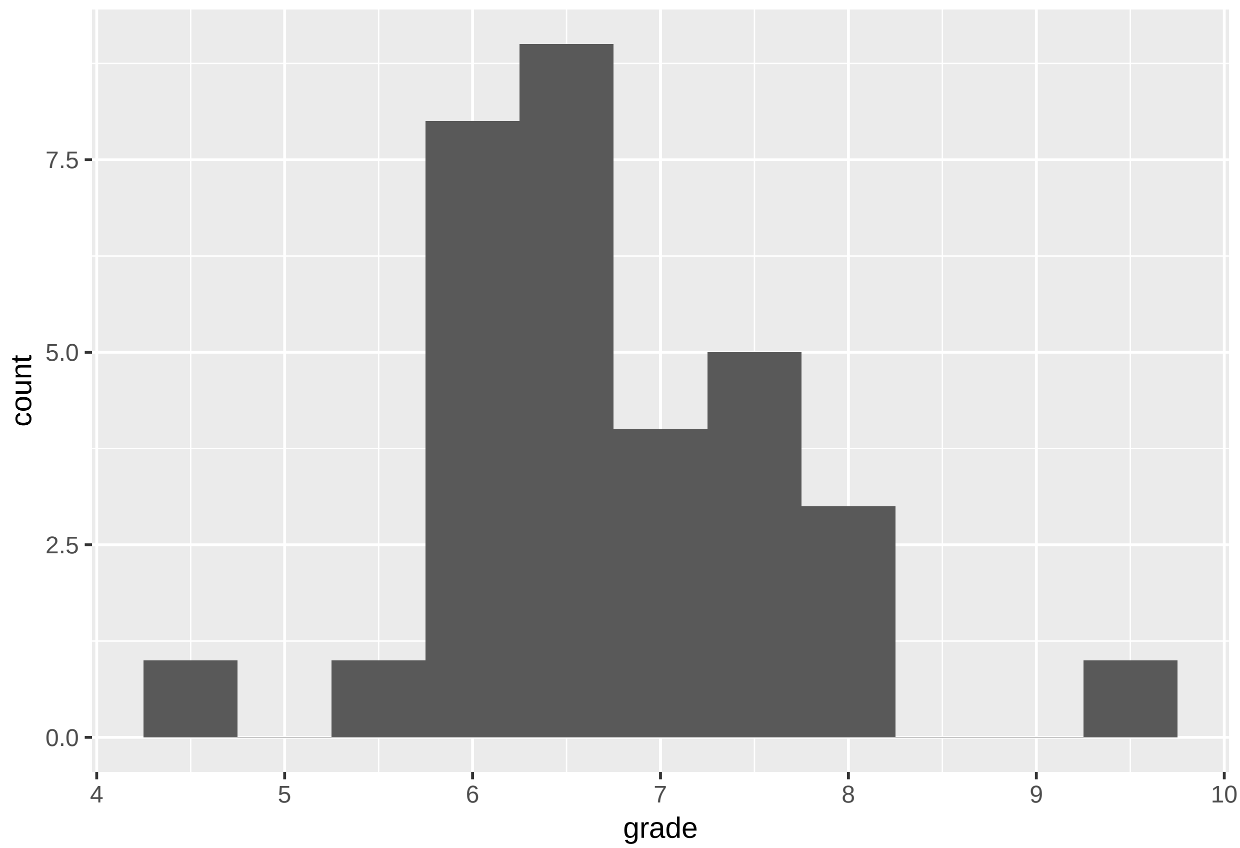
Visual exploratory data analysis

Several types of plots are useful for exploratory data analysis. In this section, you will construct different plots to get a feel for the two datasets we use in this practical: `Hitters` and `gg_students`. One of the most common tasks is to look at the distributions of variables in your dataset.

Histogram

Use `geom_histogram()` to create a histogram of the grades of the students in the `gg_students` dataset. Play around with the `binwidth` argument of the `geom_histogram()` function.

```
gg_students %>%
  ggplot(aes(x = grade)) +
  geom_histogram(binwidth = .5)
```

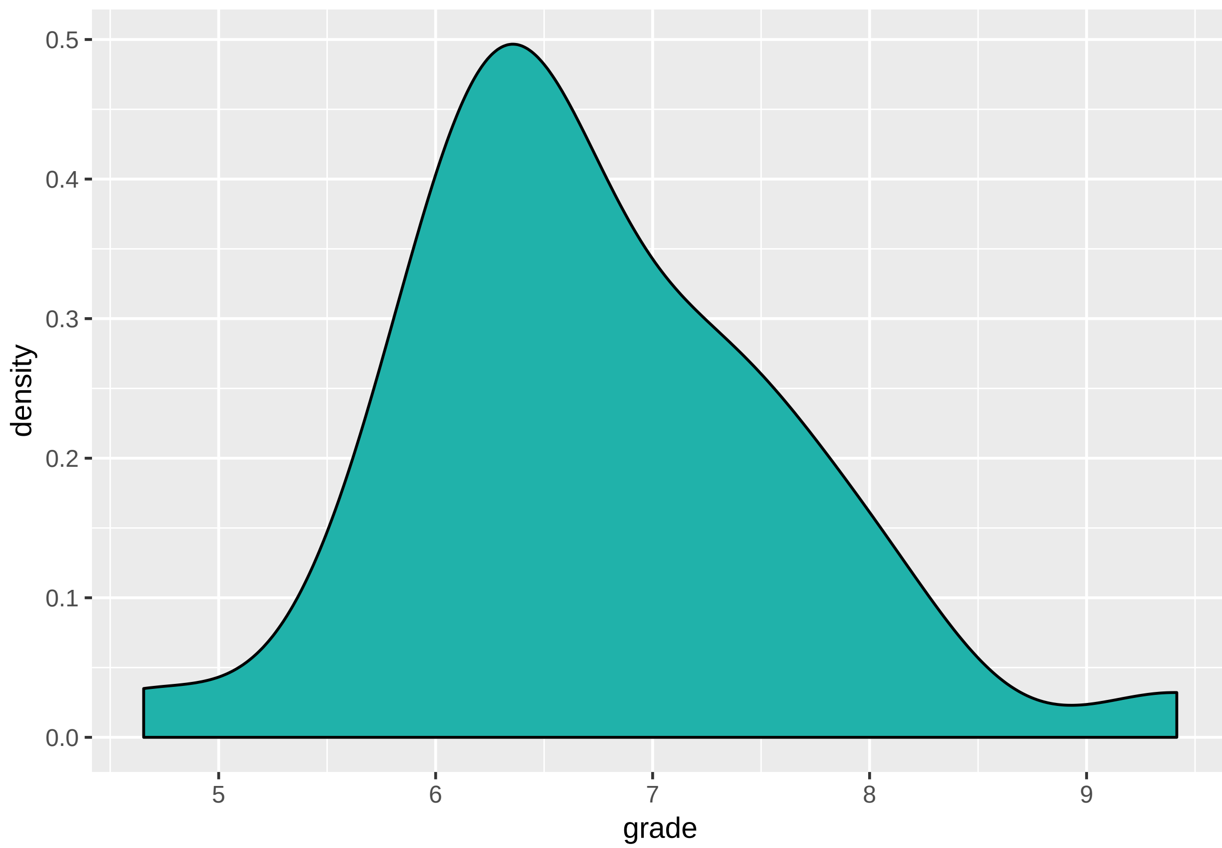


Density

The continuous equivalent of the histogram is the density geom.

Use `geom_density()` to create a density plot of the grades of the students in the `gg_students` dataset. Add the argument `fill = "light seagreen"` to `geom_density()`.

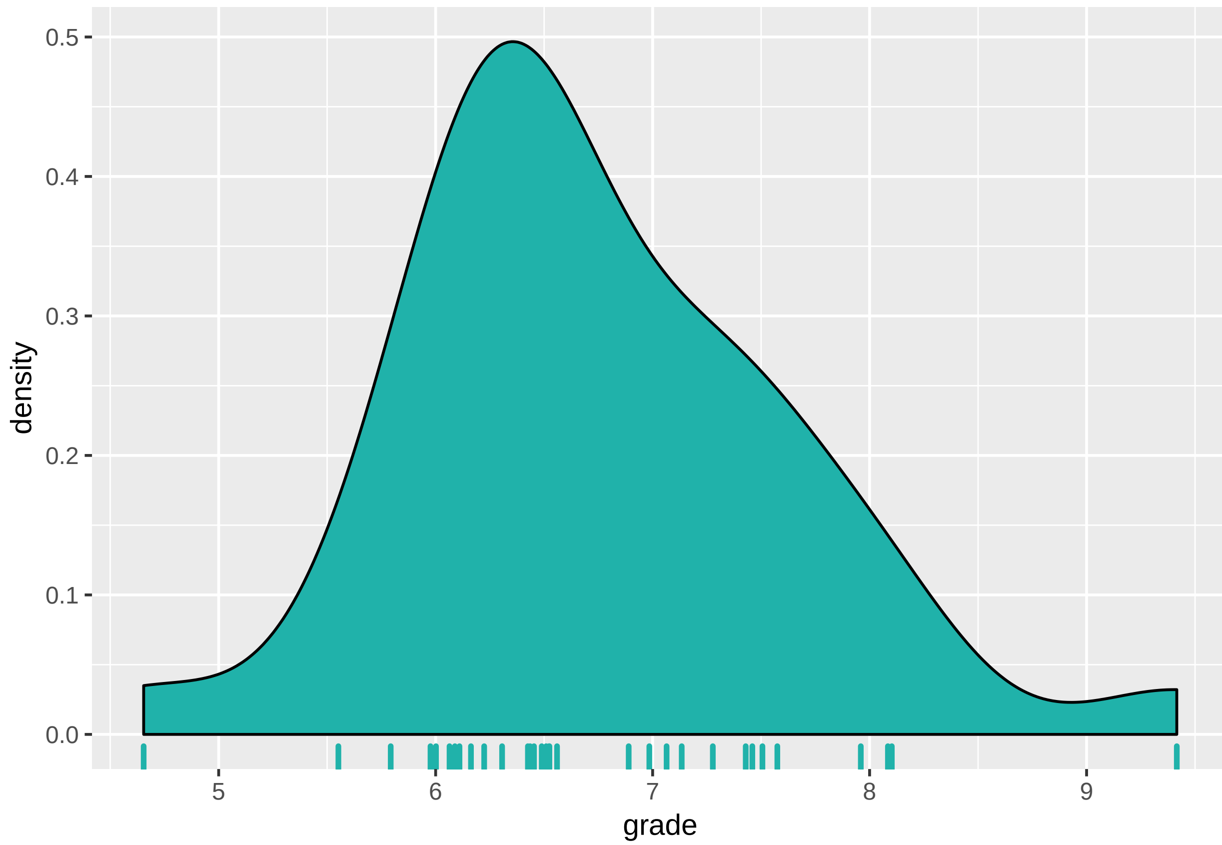
```
gg_students %>%
  ggplot(aes(x = grade)) +
  geom_density(fill = "light seagreen")
```



The downside of only looking at the density is that it is an abstraction from the raw data. We can add a raw data display in the form of rug marks.

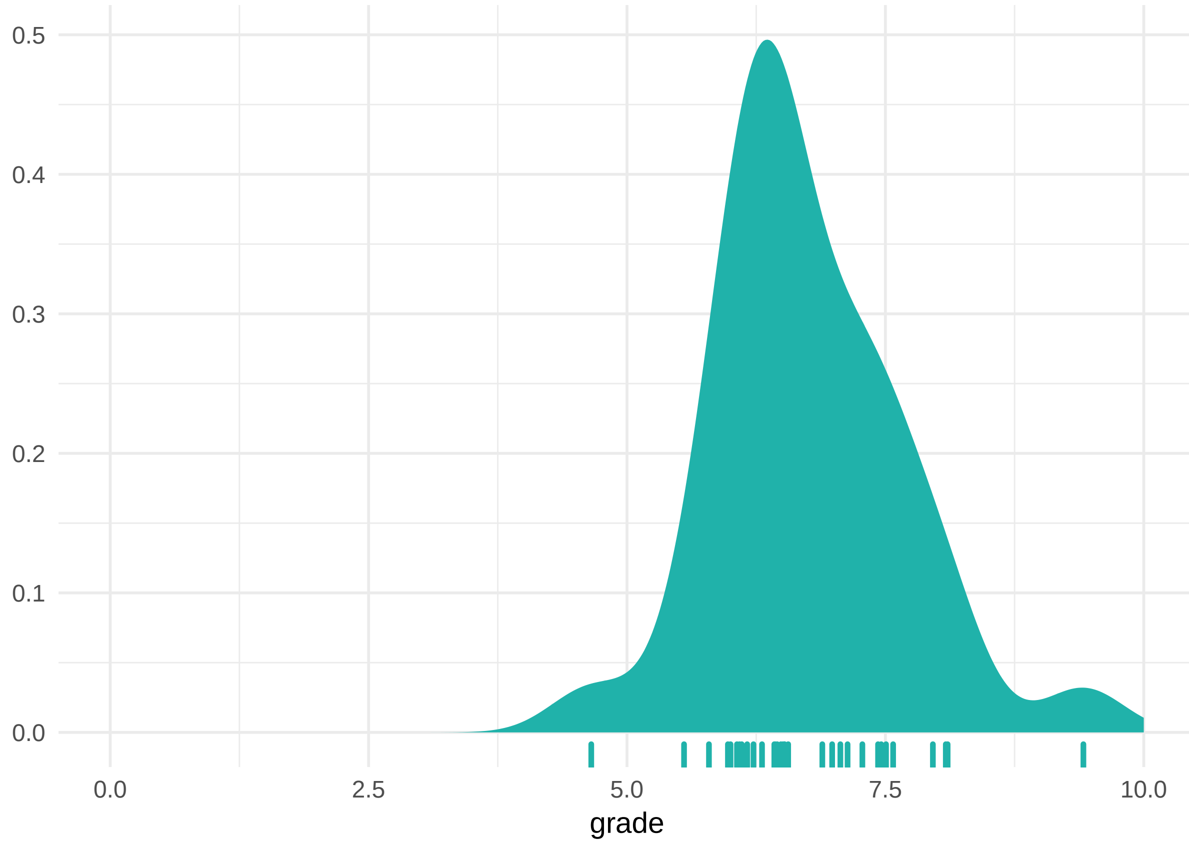
Add rug marks to the density plot through `geom_rug()`. You can edit the `colour` and `size` of the rug marks using those arguments within the `geom_rug()` function.

```
gg_students %>%
  ggplot(aes(x = grade)) +
  geom_density(fill = "light seagreen") +
  geom_rug(size = 1, colour = "light seagreen")
```



Increase the data to ink ratio by removing the y axis label, setting the theme to `theme_minimal()`, and removing the border of the density polygon. Also set the limits of the x-axis to go from 0 to 10 using the `xlim()` function, because those are the plausible values for a student grade.

```
gg_students %>%
  ggplot(aes(x = grade)) +
  geom_density(fill = "light seagreen", colour = NA) +
  geom_rug(size = 1, colour = "light seagreen") +
  theme_minimal() +
  labs(y = "") +
  xlim(0, 10)
```

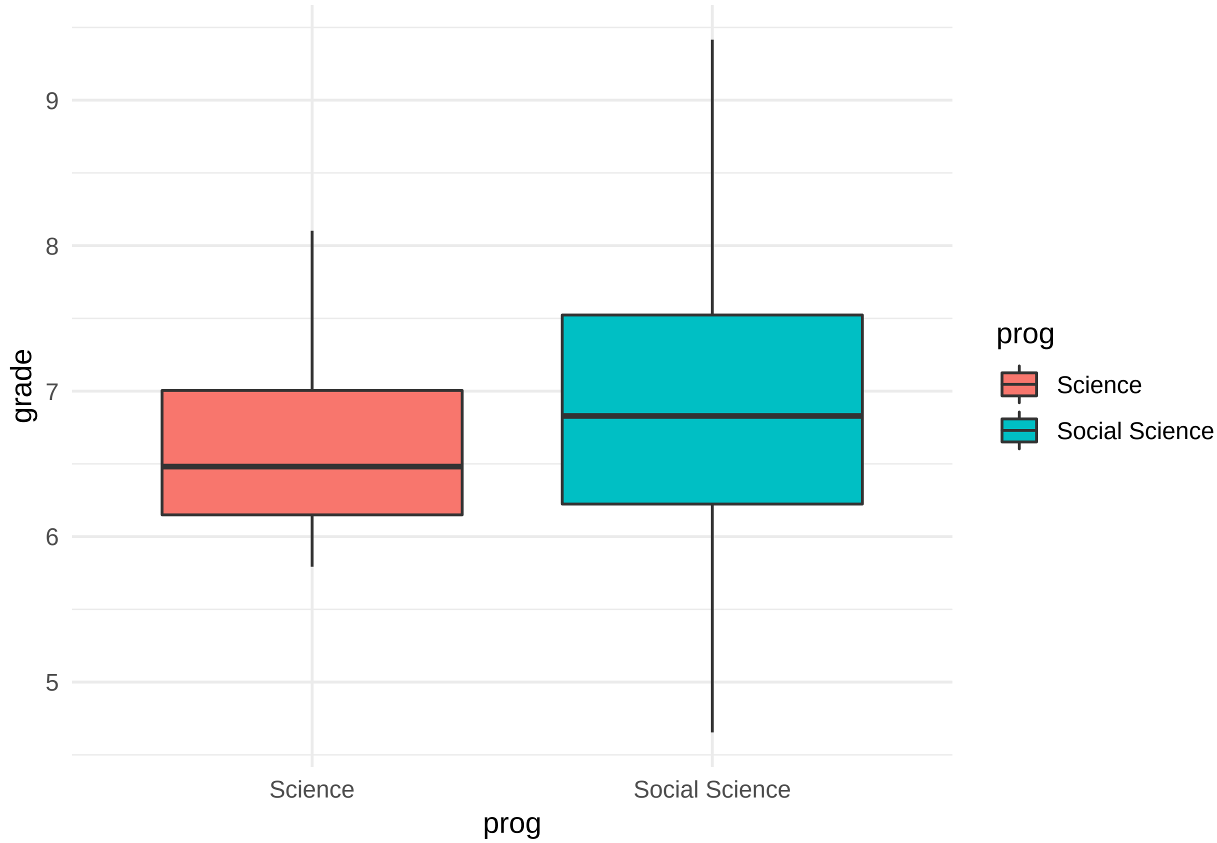


One of the most useful geoms is `geom_boxplot()`. It allows for visual comparison of the distribution of two groups.

Create a boxplot of student grades per programme in the `gg_students` dataset you made earlier: map the programme variable to the x position and the grade to the y position. For extra visual aid, you can additionally map the programme variable to the fill aesthetic

---

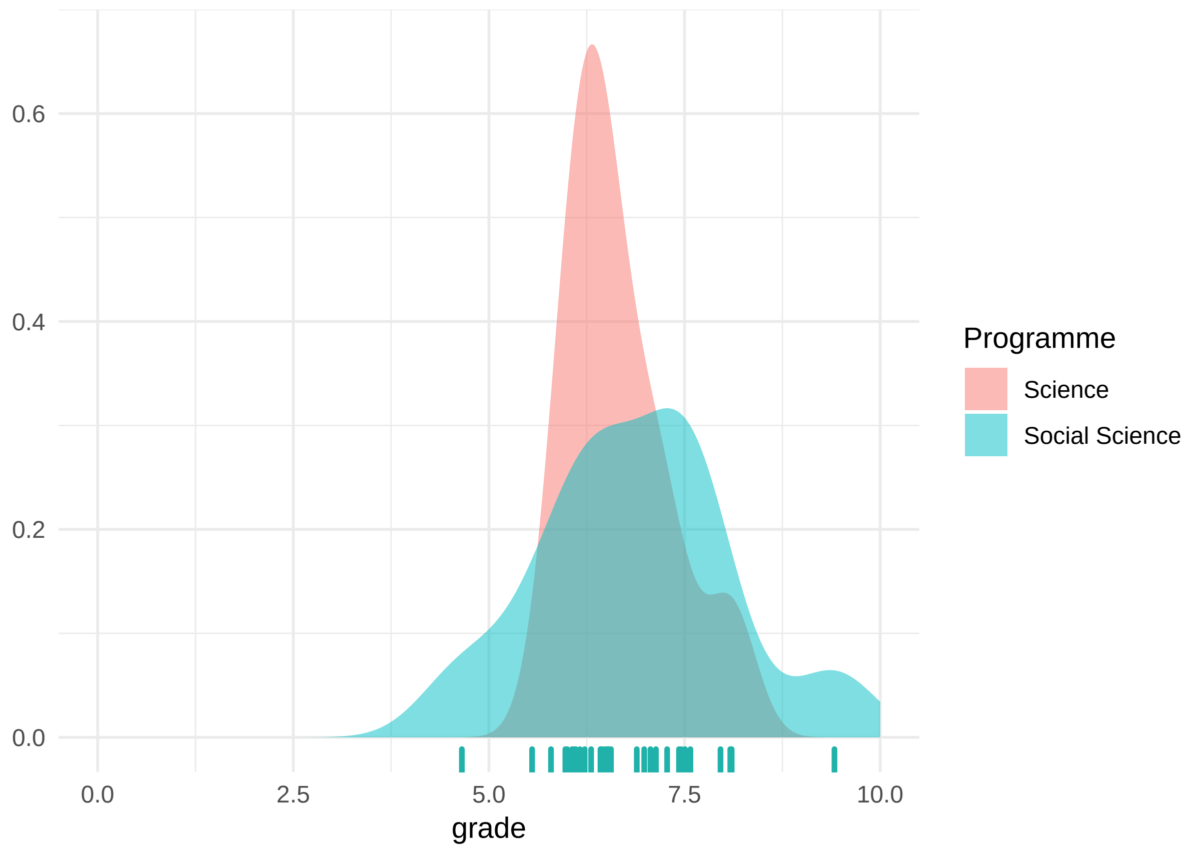
```
gg_students %>%
  ggplot(aes(x = prog, y = grade, fill = prog)) +
  geom_boxplot() +
  theme_minimal()
```



Comparison of distributions across categories can also be done by adding a fill aesthetic to the density plot you made earlier. Try this out. To take care of the overlap, you might want to add some transparency in the `geom_density()` function using the `alpha` argument

---

```
gg_students %>%
  ggplot(aes(x = grade, fill = prog)) +
  geom_density(alpha = .5, colour = NA) +
  geom_rug(size = 1, colour = "light seagreen") +
  theme_minimal() +
  labs(y = "", fill = "Programme") +
  xlim(0, 10)
```



0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
grade
Programme
Science
Social Science