Data visualization in R

Zoltan Kekecs, Marton Kovacs

November 04, 2020

Contents

1 Abstract

During this exercise we will learn how to create graphs to better understand the data and to understand the underlying relationships. This exercise demonstrates plotting the distribution of variables using the ggplot2 package.

2 Basics of data visualization using ggplot2

During this exercise we will work with a datase called "movies". This dataset contains data accessed from Rotten Tomato and IMDB about different movies.This dataset can be loaded using the following code.

Note that after running this code the "movies" data shows up in the workspace.

```
movies <- read.csv("https://raw.githubusercontent.com/kekecsz/PSYP14-Advanced-Scientific-Methods/main/m
```

Lets do a basic overview of the dataset usign the methods we learned earlier.

```
movies

View(movies)

str(movies)
```

We will graph data from this dataset using the ggplot2 package. As always, we need to load this package (and install if you don't have this package yet). The **tidyverse** package already contains **ggplot2**, so if you have tidyverse isntalled and loaded, you should already have ggplot2 as well. Since ggplot2 is a part of the tidy family, it is compatible with the %>% (pipe) operator.

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 3.6.3
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

The ggplot2 package contains a lot of functions which cna be customized in almost infinite ways. If you master the use of ggplot2 you will be able to create extremely informative and nice looking graphs, but this takes a lot of practice.

Initially it might help to use the ggplot2 cheatsheet to get a basic overview and reminder of which functions to use in which cases.

https://rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

Lets say that we would like to visualize the relationship of critics ratings with the ratings of viewers. This can be done using a simple scatterplot, so we will use this to demonstrate how ggplot works.

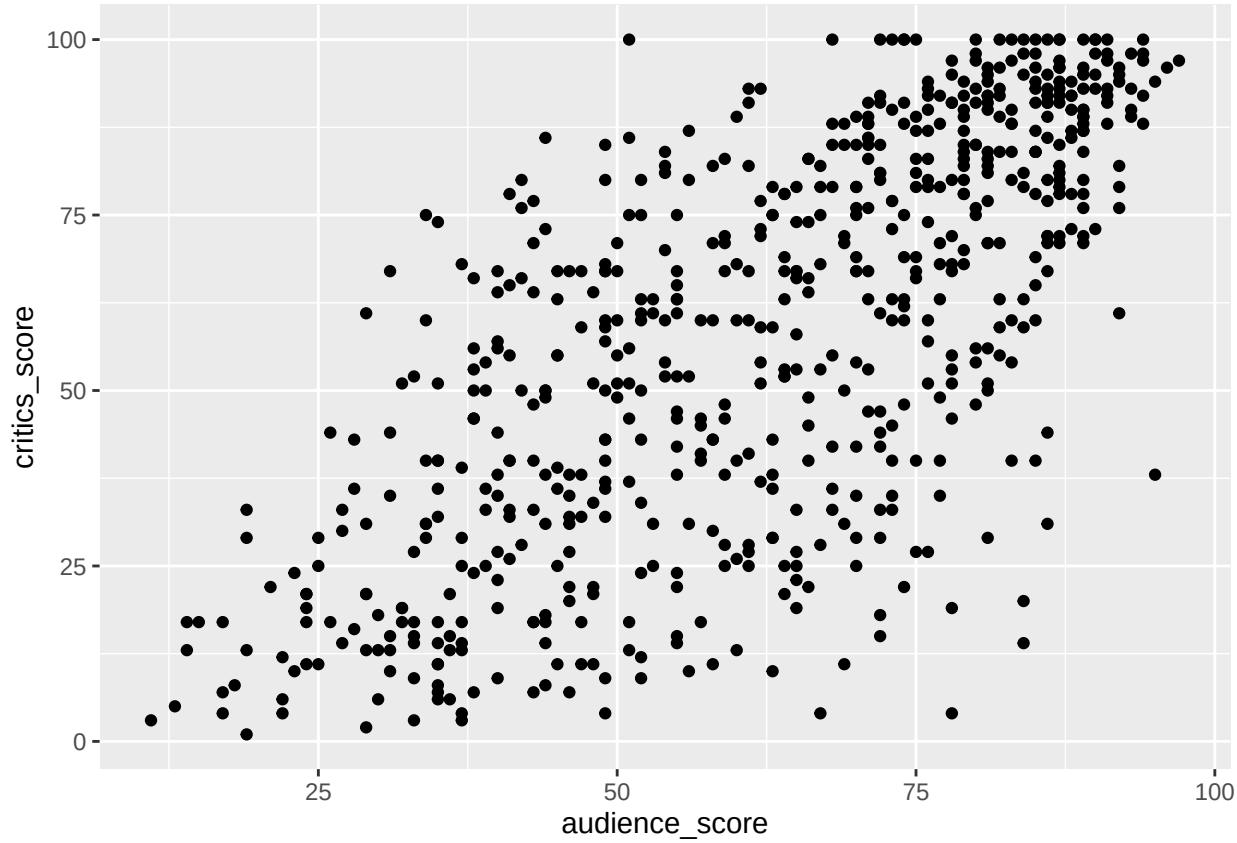
In the code below you can see how the **pipe operator** is used to pass the object "movies" into the ggplot function. (Alternatively we could write ggplot(movies)+...)

After each ggplot function we use **the + sign** to indicate that we have some additional functions we want to applz to our plot, until the final line of the code is finished.

The **aes()** (aesthetics) function determines the data elements to be added to the graph. In the graph below we want to visualize the data from the variable audicance_scores on the x axis, and data from the critics_scores on the y axis, so we specify this in the aes() function.

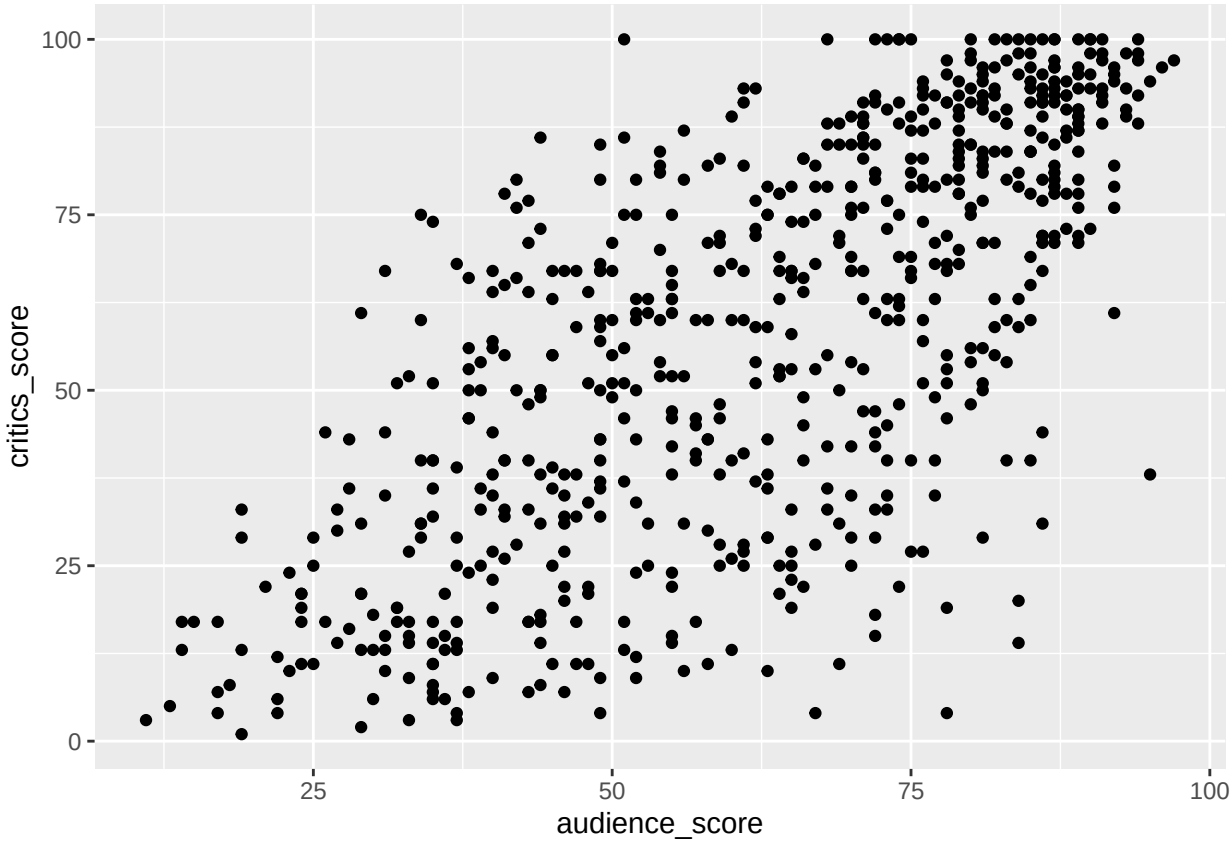
The **geom_...()** functions determine the visualization type or visualization element we are going to use on the graph. We want to create a scatterplot, which we can use via the **geom_point()** function in ggplot.

```
movies %>%
  ggplot() +
    aes(x = audience_score,
      y = critics_score) +
    geom_point()
```



Just like other types of outputs, graphs can also be **saved as objects** in R, and later called and displayed by just using the object name:

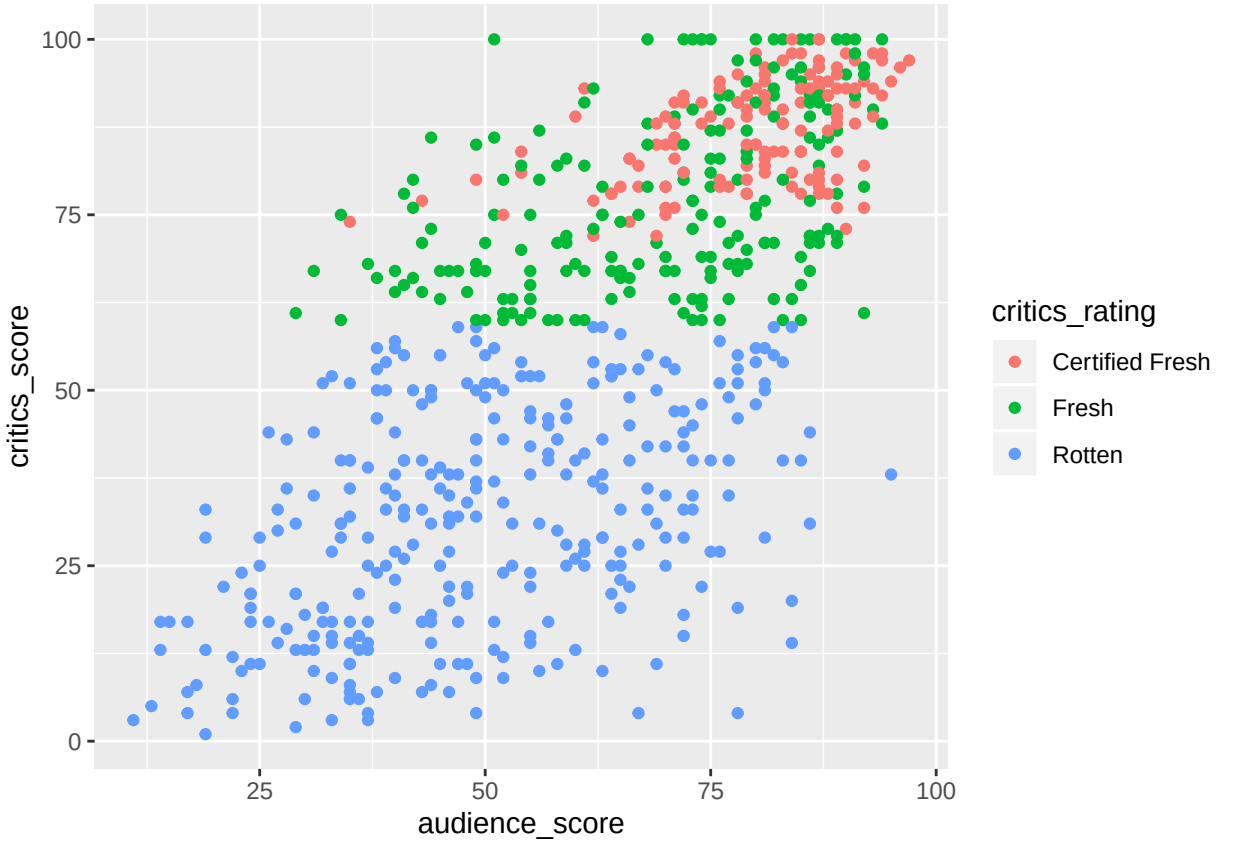
```
plot1 <- movies %>%
          ggplot() +
            aes(x = audience_score,
                y = critics_score) +
            geom_point()

plot1
```



One of the great things about ggplot is that we can very easily **add new data elements and visualization elements** to already existing plots.

For example in this code **we add another variable** called "critics_rating" into the visualization, which is a categorical rating of the movies instead of a rating on a continuous scores. We can do this within the aes() function. Since the x and y axes are already taken, we can visualize this third variable as different **colors** of the points.

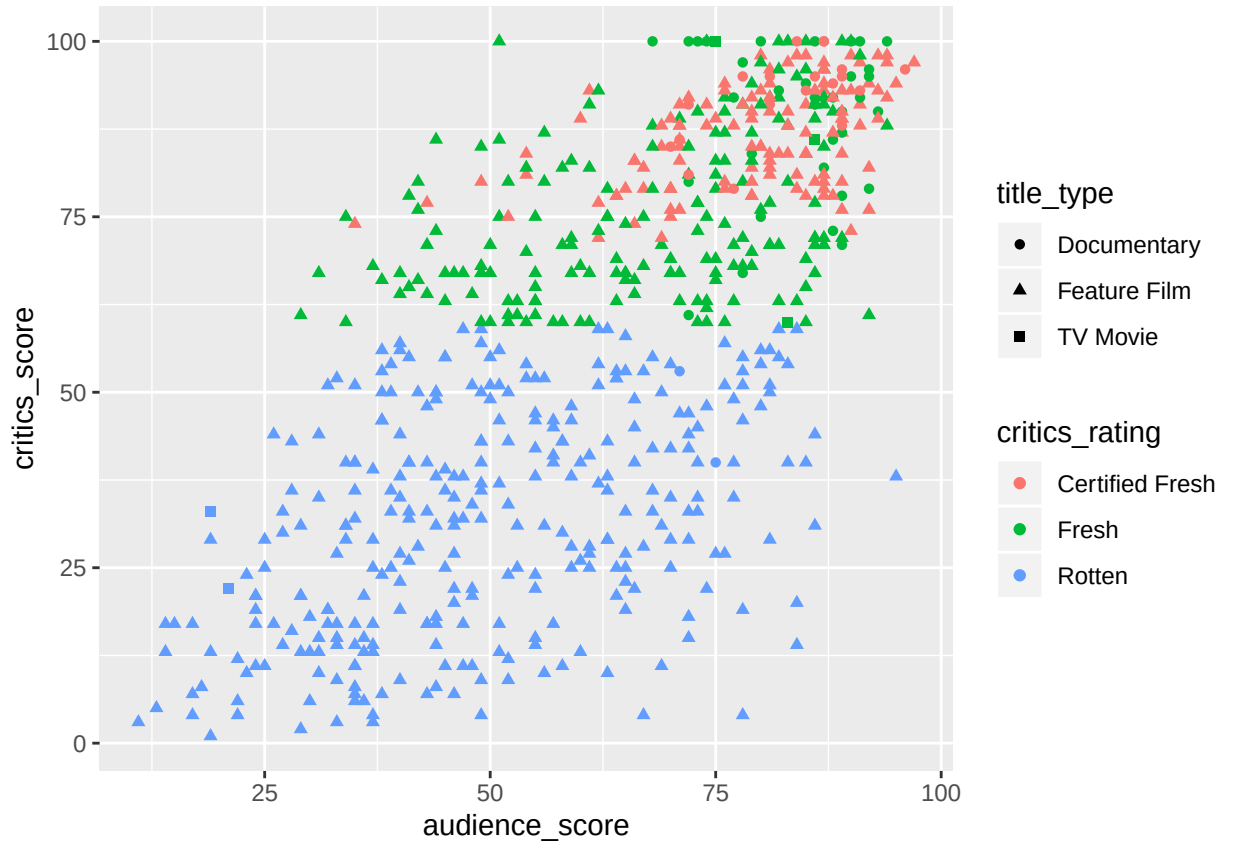
```
plot2 <- movies %>%
          ggplot() +
            aes(x = audience_score,
                y = critics_score,
                color = critics_rating) +
            geom_point()

plot2
```



We do not need to write out the whole code every time we want to add soemthing. One other feature of ggplot is that ti can be amended using the object name like in the code below. In this code we add a fourth variable to the visualization: title_type, which is about the type of the movie. Notice that we use the object name and add a new aes() with the + sign.

This plot shows us that a good proportion of the movies that got 100 as a critics score are documentaries.

```
plot2 + aes(shape = title_type)
```



We can include new visualization elements (called geoms in ggplot) to this plot as well.

By adding the geom_smooth geom we can see a projected underlying pattern (using a regression method) behind the relationship of critics scores and audience scores.

```
movies %>%
  ggplot() +
    aes(x = audience_score,
        y = critics_score,
            color = critics_rating) +
        geom_point() +
        geom_smooth()
```

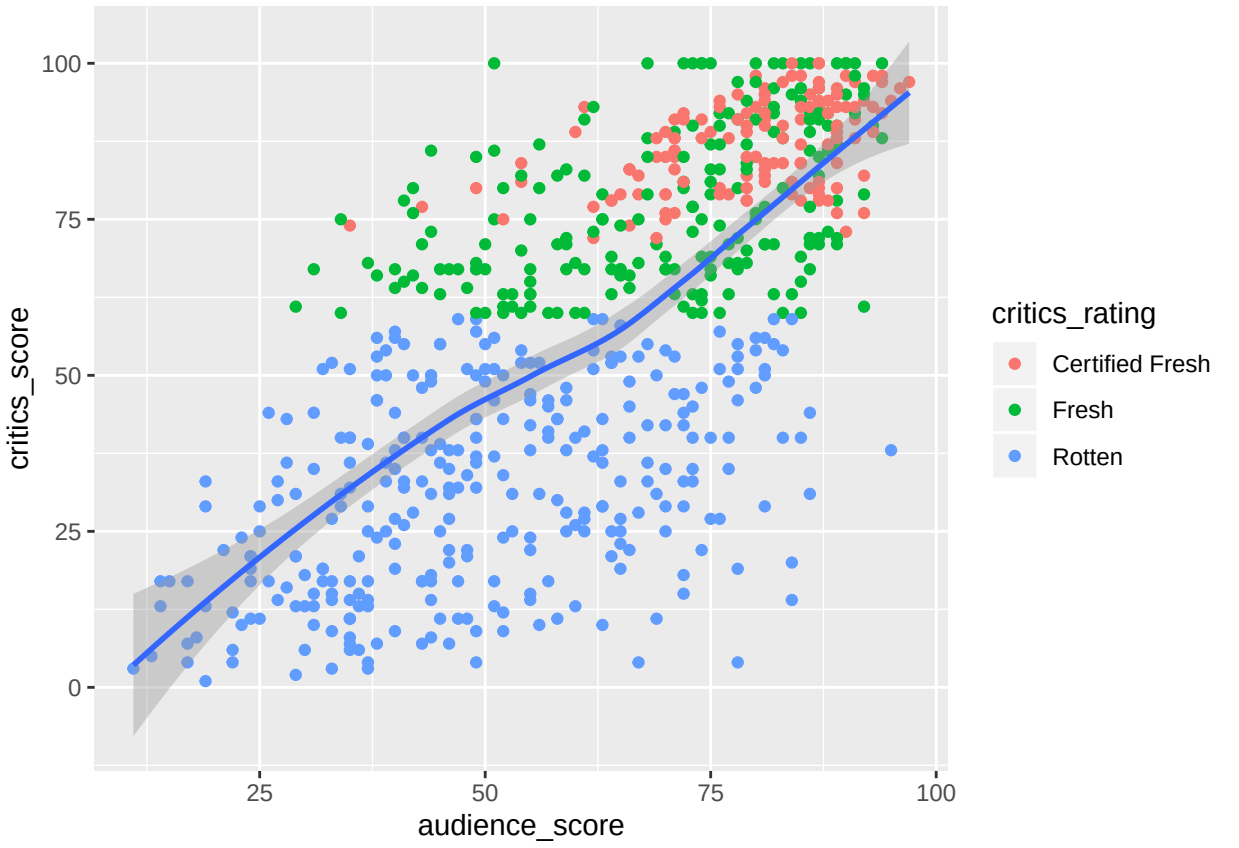
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

Since "color" is entered into the **global aes() function** that affects all geoms in the plot, the lines are plotted separately for the three critics ratings.

If we want to get a single line for the scatterplot (not to plot separately for critics rating), we need to take the critics_rating out of the global aes(), and only enter it as a **"local" aes()** within the geom_point() function, indicating that the critics_rating should only affect the color of point, and not be represented by other geoms. See the example below:

```
movies %>%
  ggplot() +
    aes(x = audience_score,
        y = critics_score) +
    geom_point(aes(color = critics_rating)) +
    geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Practice

We should continue using the movies dataset.

- Display the relationship between the variables imdb_rating and imdb_num_votes (number of votes based on which the imdb rating was determined)
- Add information abut the movie genre (variable "genre") to this graph

---

3 Customizing plots

3.1 Setting some plot parameters as constant (not linked to variables)

If we want to set one of the parameters of the plot to be set as constant without depending on data, for example if we want to set the color of the geom_smooth line to red, we need to specify this **within the given geom_...() function outside of any potential local aes() functions**.

In the examlpe below we set the geom_smooth **color and the fill and shape** of the geom_point points to constant values:

To determine the shampe of points we use numbers. A guide to which number refers to which shape can be found here:

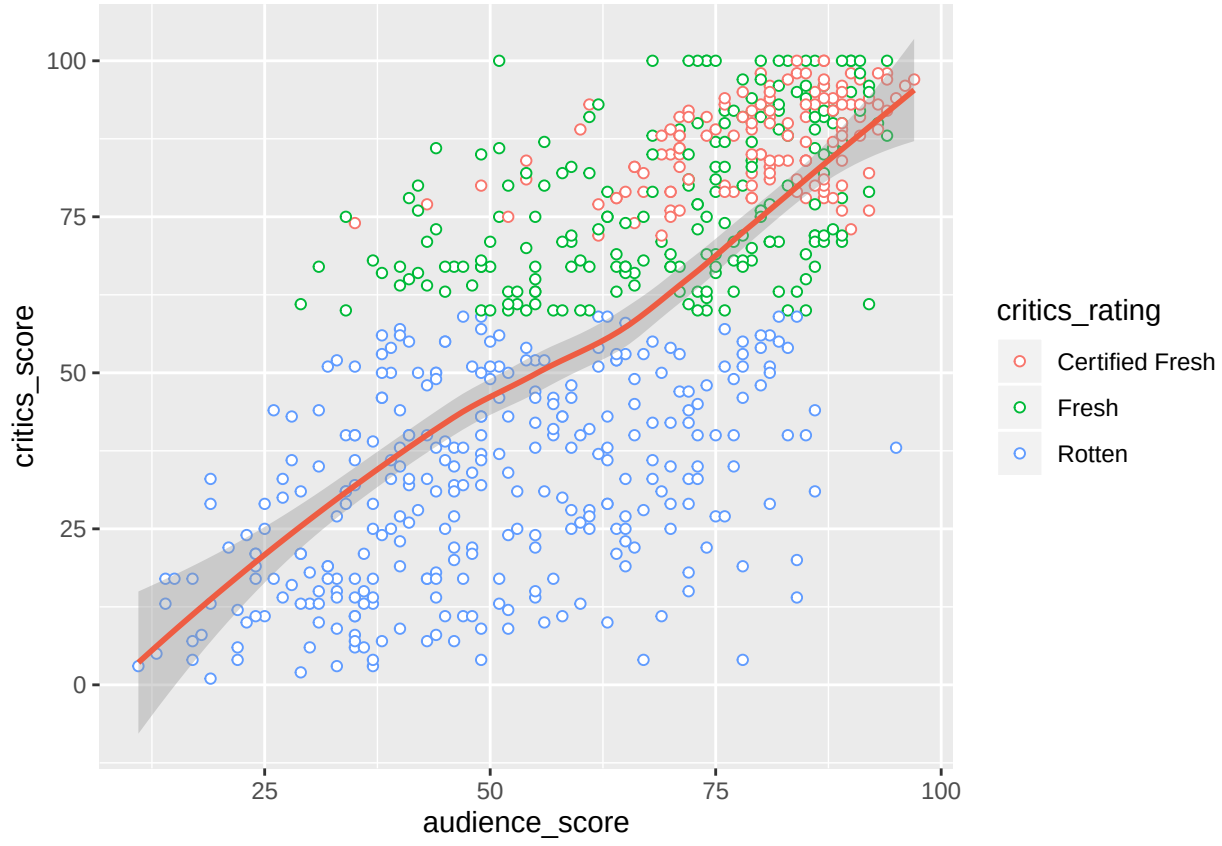
http://www.sthda.com/english/wiki/ggplot2-point-shapes

A lot of color names can be entered in English (see the example code below), but if more control is desired over the exact color, here is a guide about the color codes:

http://sape.inf.usi.ch/quick-reference/ggplot2/colour

```
movies %>%
  ggplot() +
    aes(x = audience_score,
        y = critics_score) +
    geom_point(aes(color = critics_rating), shape = 21, fill = "white") +
    geom_smooth(color = "tomato2")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

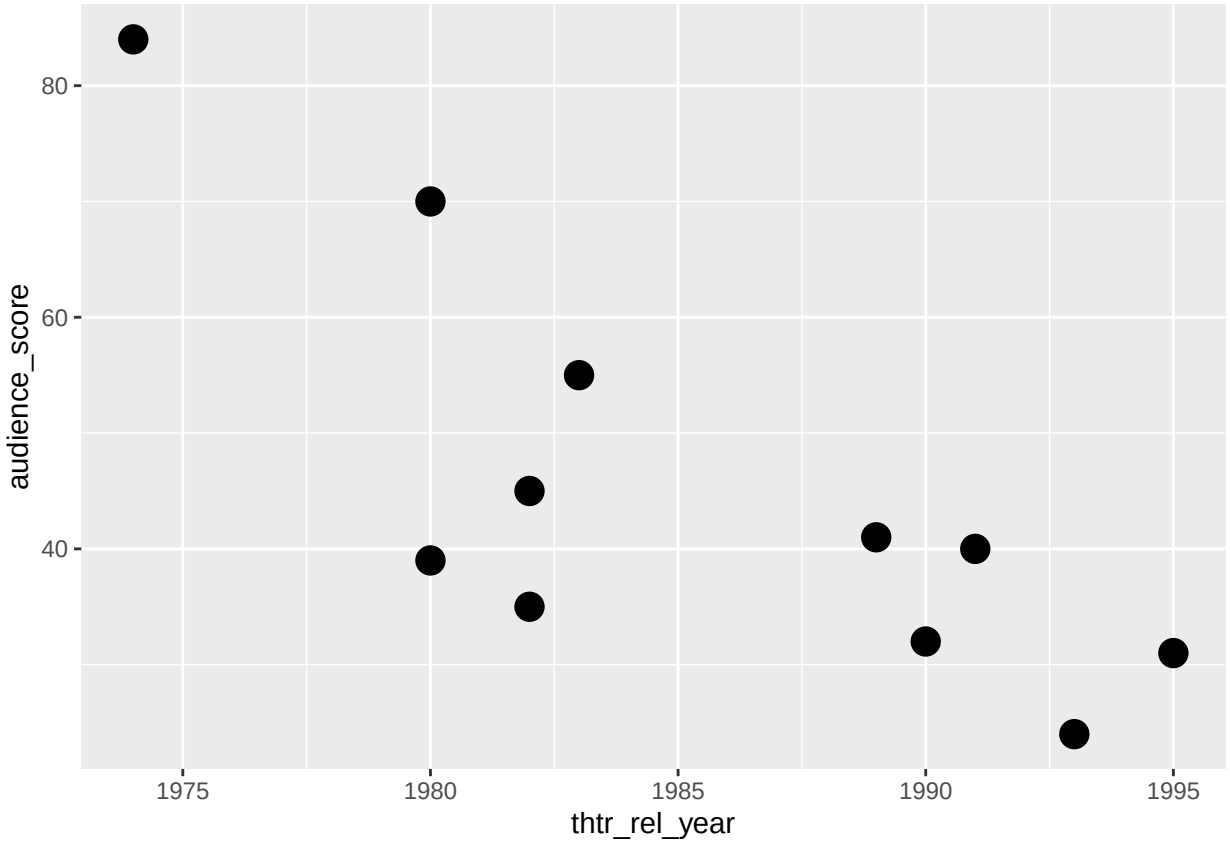


3.2 Preparing data for visualization

As we have seen above, ggplot is compatible with the %>% operator. This means that **it can be chained into the previously leared dplyr functions**, so data can be prepared directly within the same chain as the plotting function.

For example lets say we would like to get an idea of the **quality of horror** movies published **over time**. We can use a scatterplot to display the relationship between release year of the movie and the audience score. We can select horror movies published between 1972 and 2002, and work with only this dataset in our plot using the filter function as learned in previous exercises, and we can pass the outut of this functio directly into ggplot.

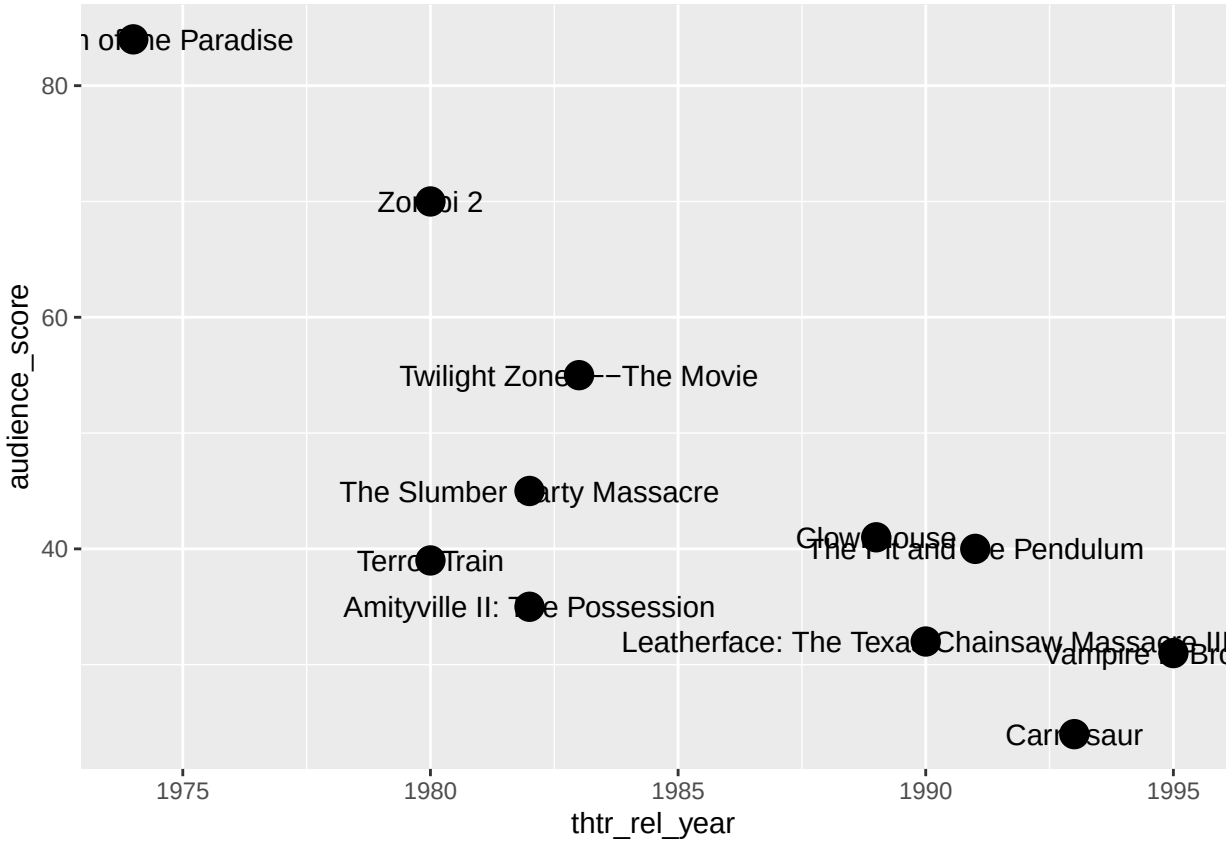
```
movies %>%
  filter(genre == "Horror", thtr_rel_year > 1972, thtr_rel_year < 2002) %>%
  ggplot() +
  aes(x = thtr_rel_year,
      y = audience_score) +
  geom_point(shape=16, fill = "white", size = 5)
```



3.3 Putting text on graphs

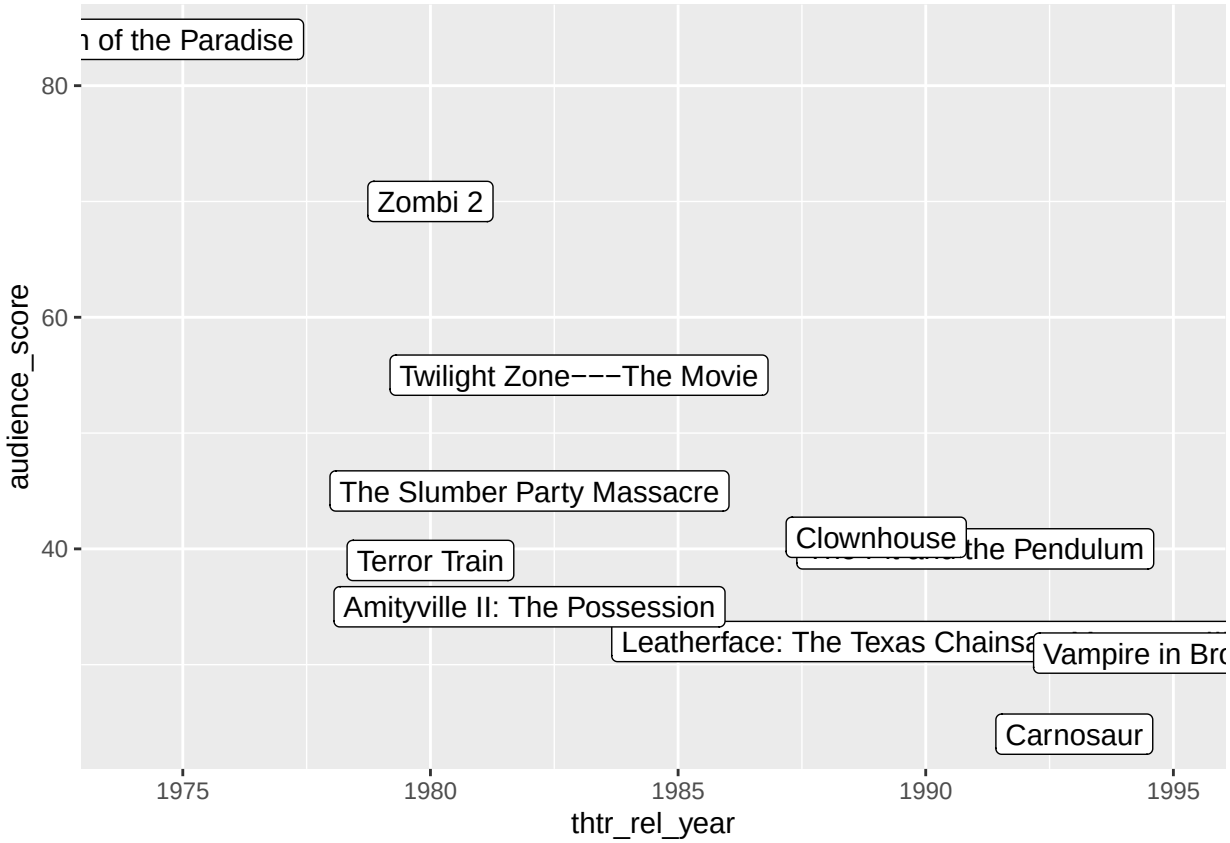
We can put text on graphs based on variables in the dataset by using the **geom_text()** geom

```
movies %>%
  filter(genre == "Horror", thtr_rel_year > 1972, thtr_rel_year < 2002) %>%
  ggplot() +
  aes(x = thtr_rel_year,
      y = audience_score,
      label = title) +
  geom_point(shape=16, fill = "white", size = 5) +
  geom_text()
```



or by using geom_label()

```
movies %>%
  filter(genre == "Horror", thtr_rel_year > 1972, thtr_rel_year < 2002) %>%
  ggplot() +
  aes(x = thtr_rel_year,
      y = audience_score,
      label = title) +
  geom_point(shape=16, fill = "white", size = 5) +
  geom_label()
```



Practice

Continue using the movies dataset.

- Display the relationship of audience_score and critics_score using a scatterplot just like we did before, but only include movies that were released in 1995 on th plot.
- Pot the title of the move on the plot on each point on the scatterplot as a label.