Transforming and Scaling Data

Matúš Seči (m.seci@ed.ac.uk)

Click here: Transforming and Scaling Data -

Tutorial aims:

1. Understand the purpose of transformations and scaling in statistical analysis.
2. Understand the underlying mathematics and use appropriate syntax and packages in R to apply both common and more advanced transformations and scaling procedures.
3. Learn how to reverse transformations and scaling to obtain estimates and predictions in the original units of measure.
4. Learn how to change the scales on the plot axes and label them appropriately.
5. Learn how to apply these concepts to real problems in ecology and environmental sciences involving data through worked examples.

1. Introduction

Data come in a wide variety of shapes and sizes. We use data distributions to study and understand the data and many models are built around assumptions that the data follow a certain distribution, most typically linear models always assume normal distribution of the data. However, real world data rarely perfectly align with the normal distribution and therefore break this assumption. Alternatively, there might be a situation where our data follow a non-linear relationship an our standard plots cannot capture it very well. For dealing with these issues we can use transformations and scaling. They are therefore powerful tools for allowing us to utilize a wide variety of data that would not be available for modelling otherwise and display non-linear relationships between data in more clear and interpretable plots.

This tutorial will teach you how to manipulate data using both common and more advanced transformations and scaling procedures in R. In addition, we will have a quick look at situations when adjusting scales on plot axes is a better decision than transforming or scaling data themselves. Throughout the tutorial we will work with datasets from ecological and environmental sciences in order to demonstrate that scaling data and using transformations are very useful tools when working with real world data.

In the following parts we will work with the data from the **Living Planet Index** which is an open-source database containing population data of a large number of species from all around the planet. In each part of the tutorial we will focus on a population of a different species. Let's load it into our script along with the packages we will use in this part of the tutorial.

ow we can look at the basic structure of the dataframe to get some idea of the different variables it contains.

```
str(LPI_species)
summary(LPI_species)
```

```
# Extract the white stork data from the main dataset and adjust the year variable
stork <- LPI_species %>%
  filter(Common.Name == 'White stork' & Sampling.method == 'Direct counts')%>%
  mutate(year = parse_number(as.character(year)))  # convert the year column to character
```

How did the population of the white stork change over time?

- Extract the white stork data and adjust the year variable to be a numeric variable using mutate() and **parse_number()**.

```
# Extract the white stork data from the main dataset and adjust the year variable
stork <- LPI_species %>%
  filter(Common.Name == 'White stork' & Sampling.method == 'Direct counts')%>%
  mutate(year = parse_number(as.character(year)))  # convert the year column to character
```

Define a custom plot theme:

We will use ggplot2 library for most of our visualizations. However, before we make the first plot and explore the data we will define a custom theme to give our plots a better look and save time not having to repeat code. This part is completely voluntary as it does not affect the main concepts presented, you can create your own theme if you want or even use some of the pre-built themes in ggplot2 such as theme_bw() or theme_classic().

```
# Define a custom plot theme

plot_theme <- function(...){
  theme_bw() +
  theme(
    # adjust axes
```

```
    axis.line = element_blank(),
    axis.text = element_text(size = 14,
                             color = "black"),
    axis.text.x = element_text(margin = margin(5, b = 10)),
    axis.title = element_text(size = 14,
                              color = 'black'),
    axis.ticks = element_blank(),

    # add a subtle grid
    panel.grid.minor = element_blank(),
    panel.grid.major = element_line(color = "#dbdbd9", size = 0.2),

    # adjust background colors
    plot.background = element_rect(fill = "white",
                                   color = NA),
    panel.background = element_rect(fill = "white",
                                    color = NA),
    legend.background = element_rect(fill = NA,
                                     color = NA),
    # adjust titles
    legend.title = element_text(size = 14),
    legend.text = element_text(size = 14, hjust = 0,
                               color = "black"),
    plot.title = element_text(size = 20,
                              color = 'black',
                              margin = margin(10, 10, 10, 10),
                              hjust = 0.5),

    plot.subtitle = element_text(size = 10, hjust = 0.5,
                                 color = "black",
                                 margin = margin(0, 0, 30, 0))
    )

}
```

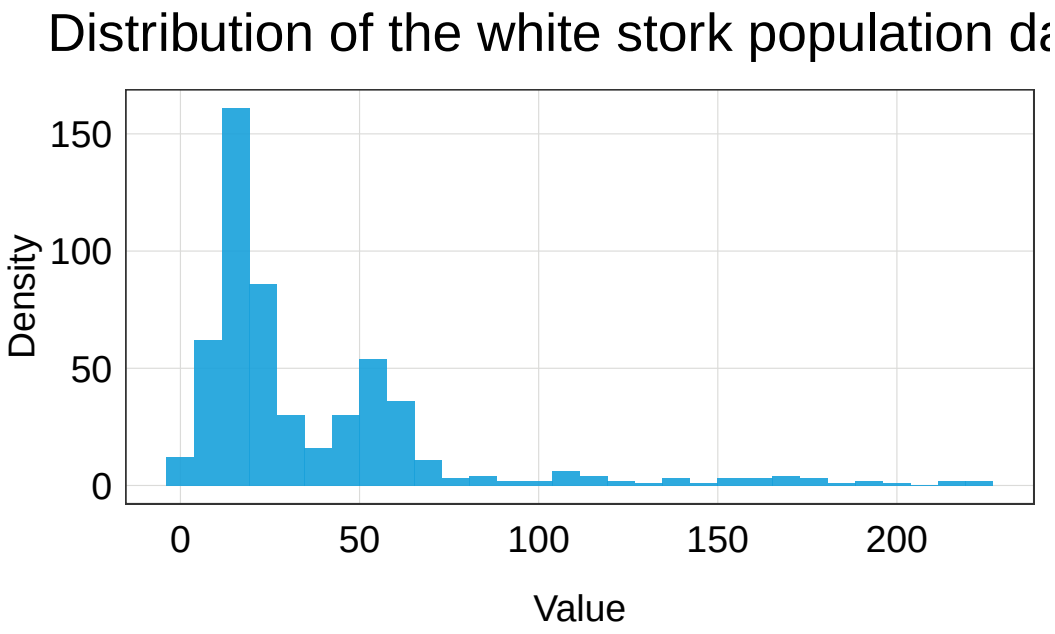
Let's look at the distribution of the data to get some idea of what the data look like and what model we could use to answer our research question.

```
# Remember, if you put the whole code in the brackets it will
# display in the plot viewer right away!

# Look at the distribution of the data
```

```
(stork_hist <- ggplot(data = stork) +
    geom_histogram(aes(x = pop),
                   alpha = 0.9,
                   fill = '#18a1db') +  # fill the histogram with a nice colour
    labs(x = 'Value',
         y = 'Density',
         title = 'Distribution of the white stork population data') +
    plot_theme())  # apply the custom theme
```

```
Warning: The `size` argument of `element_line()` is deprecated as of ggplot2 3.4.0.
i Please use the `linewidth` argument instead.
```

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



We can see that our data are very right-skewed (i.e. most of the values are relatively small). This data distribution is far from normal and therefore we cannot use the data directly for modelling with linear models which assume normal distribution. This is where transformations come in!

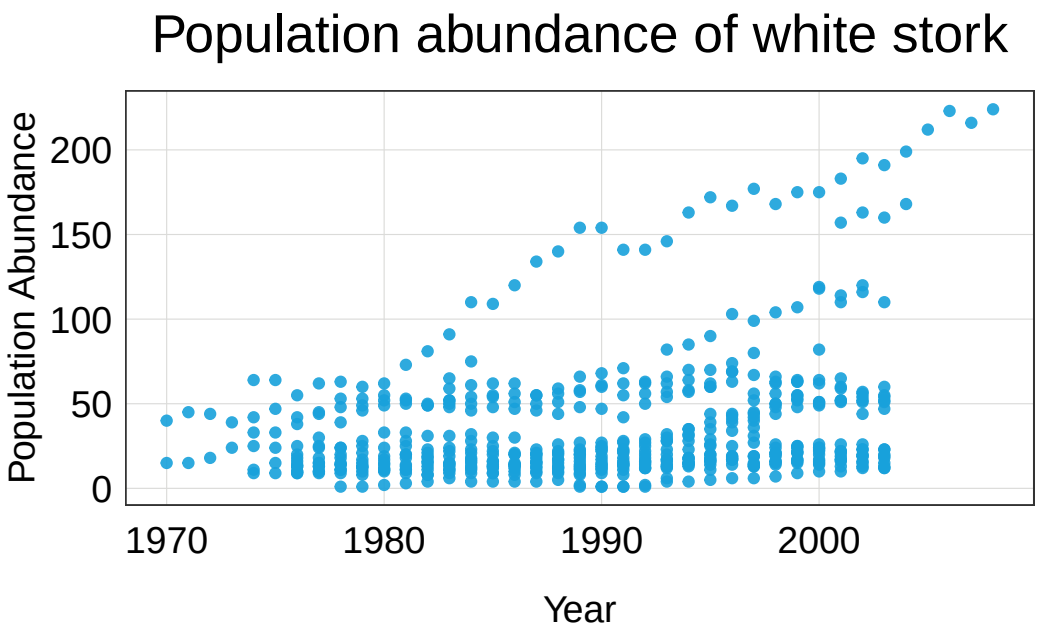
Observant learners will notice that we are dealing here with **count data** and therefore we could model this dataset using **generalized linear model** with **Poisson distribution**. This

would be a perfectly correct approach, however, for the sake of this tutorial we will stick with simple linear models to demonstrate how we can use transformations to model non-normally distributed data using simple linear models.

Logarithmic transformation

The histogram above showed that we are dealing with skewed data. We can also plot a simple scatter plot to see that these data would not be very well described by a straight line. An **exponential curve** would fit the data much better.

```
# Plot a scatter plot of the data
(stork_scatter <- ggplot(data = stork) +
    geom_point(aes(x = year, y = pop),  # change to geom_point() for scatter plot
                   alpha = 0.9,
                   color = '#18a1db') +
    labs(x = 'Year',
         y = 'Population Abundance',
         title = 'Population abundance of white stork') +
    plot_theme())  # apply the custom theme
```

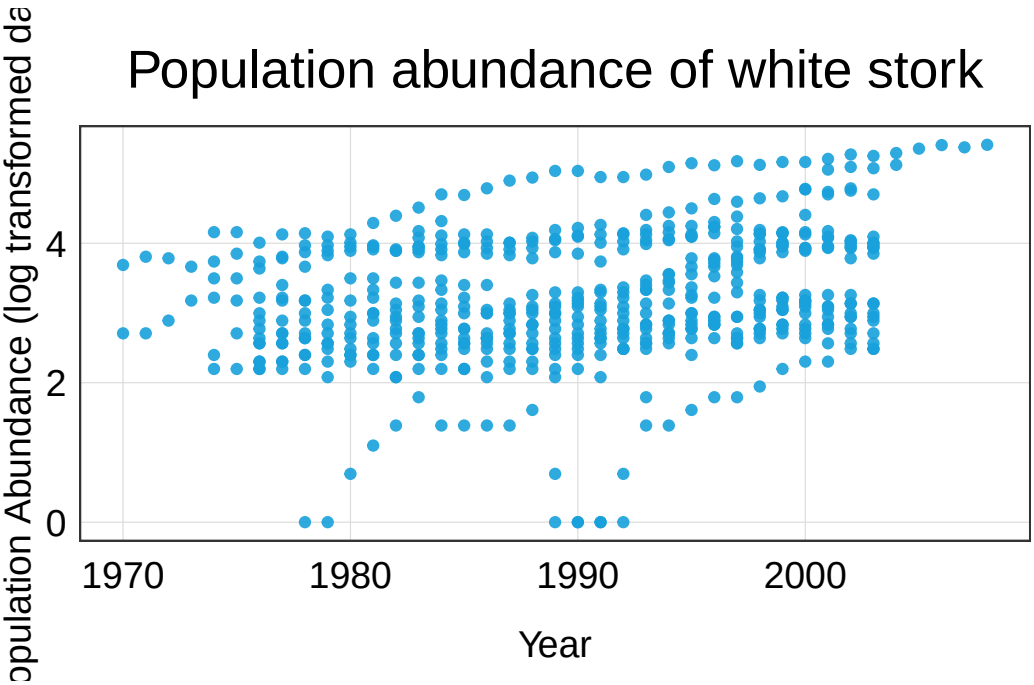


This means that we need to apply a **logarithmic transformation** which will **linearize** the data and we will be able to fit a linear model. Luckily, this procedure is very simple in R using

a base R function **log()** which by default uses **natural logarithm**, i.e. logarithm with *base e (Euler's number)*.

The choice of the base for a logarithm is somewhat arbitrary but it relates to the '*strength of transformation*' which we will cover a bit later in the tutorial. If you wanted to use a logarithm with a different base you could either define it in the function call like this log(x, base = 10) or for some common types use pre-built functions (e.g. log10(x) or log2(x)). Together with mutate() function we can create a new column with the transformed data so that we do not overwrite the original data in case we want to use them later.

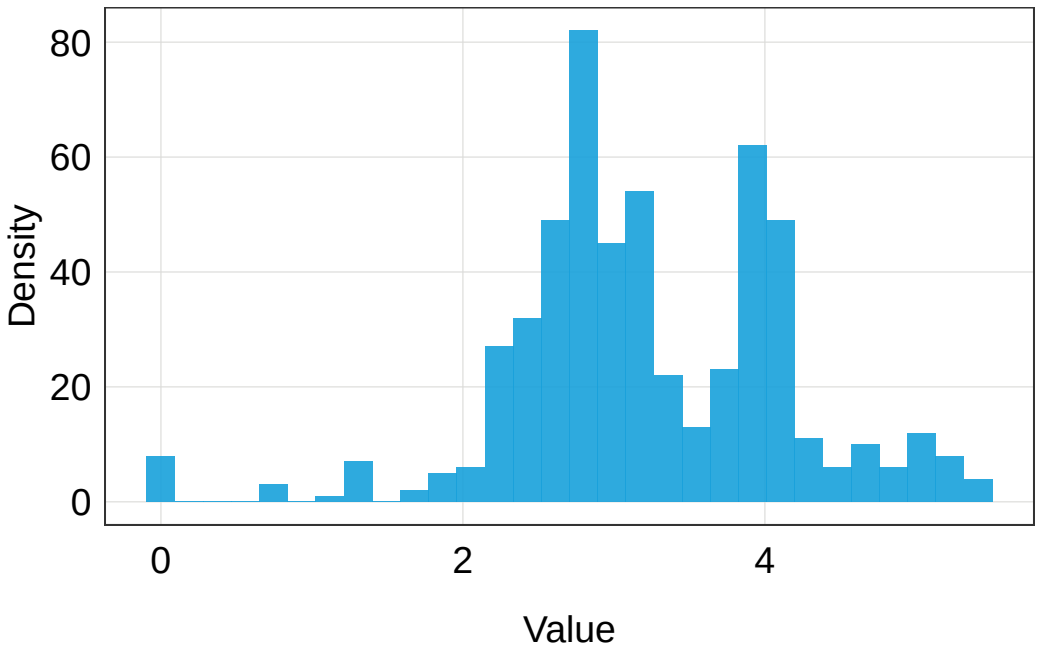
```
# Log transform the data
stork <- stork %>%
  mutate(logpop = log(pop))

# Plot a scatter plot of the log transformed data
(stork_scatter <- ggplot(data = stork) +
    geom_point(aes(x = year, y = logpop),  # change pop -> logpop
                   alpha = 0.9,
                   color = '#18a1db') +
    labs(x = 'Year',
         y = 'Population Abundance (log transformed data)',
         title = 'Population abundance of white stork') +
    plot_theme())  # apply the custom theme
```



We can see that the data have been constrained to a much narrower range (y-axis) and while there is not a crystal clear linear pattern we could argue that a linear line would fit the best for this scatter plot. Let's have a look at how the data distribution changed by looking at a histogram of the log transformed data.

```
# Plot the histogram of log transformed data
(stork_log_hist <- ggplot(data = stork) +
    geom_histogram(aes(x = logpop),
                   alpha = 0.9,
                   fill = '#18a1db') +
    labs(x = 'Value',
         y = 'Density') +
    plot_theme())
```

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



Even though the distribution is not perfectly normal it looks much closer to the normal distribution than the previous histogram!

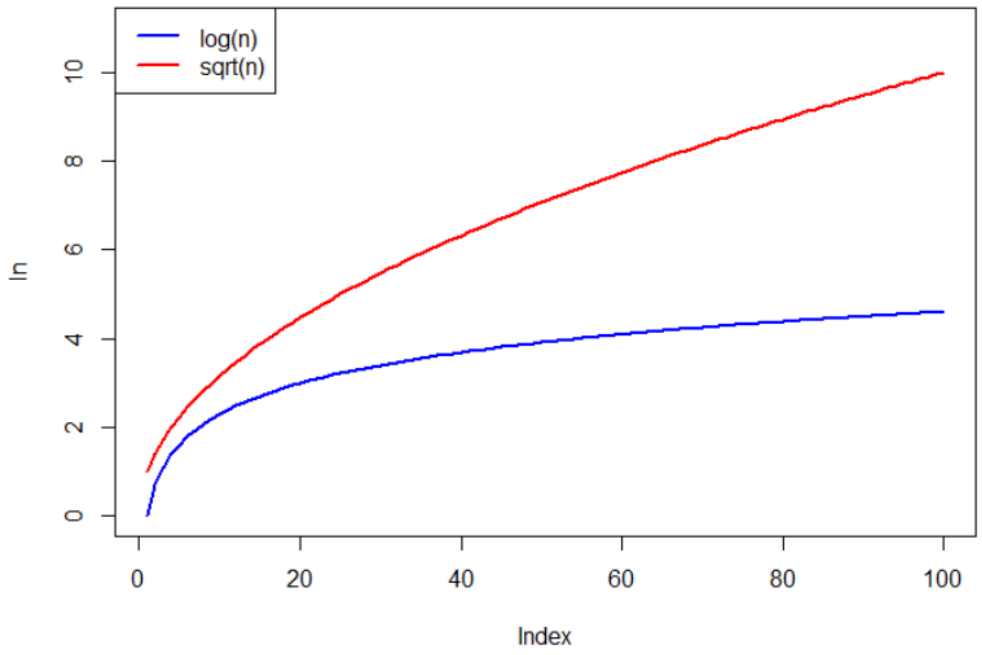
Log transformations are often used to transform right-skewed data, however, the transformation has a major shortcoming which is that *it only works for positive non-zero data.* This is due to the mathematical properties of the logarithmic function.

If you find out that your data have a 0 values but you would still like to use log transformation you can **add a constant** to the variable before performing the transformation, for example *log(x + 1)* where x is the variable. This way you can get rid of the negative or zero values. You can do this either manually or using `log1p()` function. [1]

Our data look quite normally distributed now but we might think that a weaker transformation could result in a data more centered than what we have now. We will therefore try to apply such a transformation - square-root transformation.

Square-root transformation

Square root transformation works in a very similar way as logarithmic transformation and is used in similar situations (right-skewed data), however, it is a weaker transformation. What do we mean by weaker? Well, to answer this question it is a good idea to look at the graphs describing log and sqrt functions.



Source: StackOverflow

As you can see the logarithmic function levels off much more quickly which means that it constrains large values much more strongly than square-root. As a result, with log transformation extreme values in the dataset will become less important. The plots also indicate that

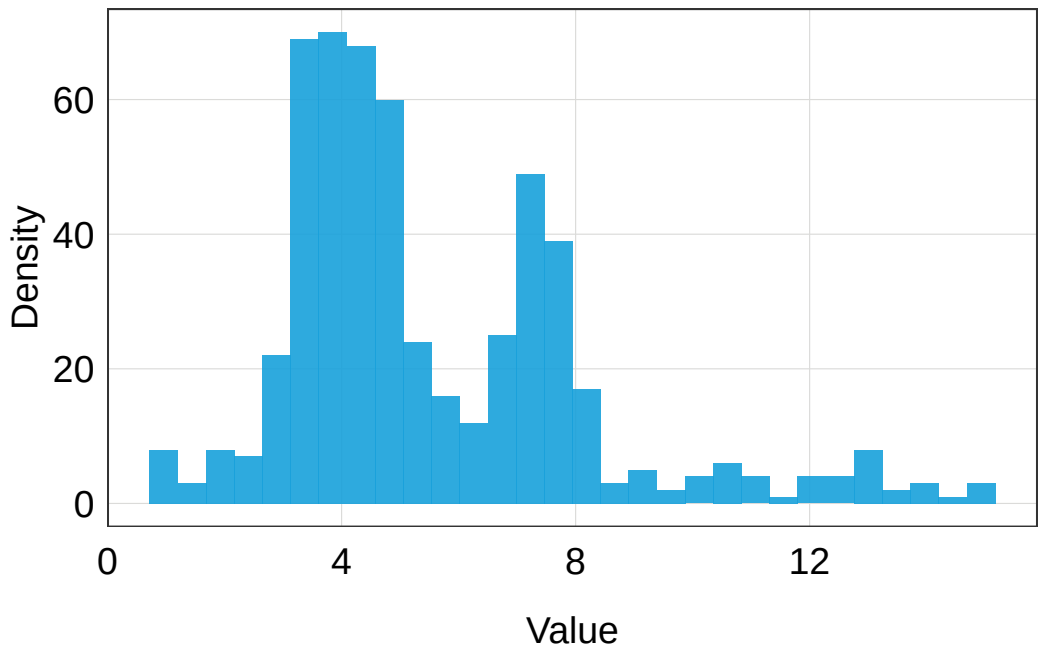
[1] However, you should use this method with caution as adding a constant changes the properties of the logartihm and it might not transform the data in a desirable way.

square-root transformation has the same disadvantage as log transformation - it can only be used on positive non-zero data.

Similar to the log transformation, we can use `sqrt()` function in base R to make this transformation.

```
# Create a square-root transformed column
stork <- stork %>%
  mutate(sqrtpop = sqrt(pop))

# Plot the histogram of square root transformed data
(stork_hist_sqrt <- ggplot(data = stork) +
    geom_histogram(aes(x = sqrtpop),  # change pop -> sqrtpop
                   alpha = 0.9,
                   fill = '#18a1db') +
    labs(x = 'Value',
         y = 'Density') +
    plot_theme())
```

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



This does not look bad but *The data are still quite skewed*. This probably means that out of