

Chapter 2

Transforming Data

Now we know some basics about summarizing data, however sometimes the data isn't always in the right format. Height, body mass, and points scored per game might be useful to combine together in order to summarize the appeal of a basketball player, but height could be given in centimeters or inches, body mass in kilograms or pounds, and these quantities *vary* in different ways too, so putting them together, e.g. by taking their arithmetic mean, might not give sensible results.

Are we still allowed to aggregate? Of course we are! We just have to make some wise decisions about how best to transform this data so that the output is useful. In this chapter, we will look at some alternative ways to transform data and then introduce the power means, a general family of means that includes the means we looked at in the previous chapter as special cases.

Assumed Background Concepts

- Standard deviation
What would you expect to be the standard deviation for heights? What is the sample standard deviation for 3, 5, 8, 2, 3?
- Log and exponential arithmetic
What does $\ln 3 + \ln 4$ simplify to? What is the value of $e^x(e^{\frac{1}{2}x} - e^{\frac{1}{3}x})$?
- Linear functions
Can you express the straight line that runs between the points (2, 3) and (-1, 8)? Can you draw the line representing the function $y = 5x - 2$ over the domain [4, 7]?
- Inverse functions
What is the inverse function for x^2 ?

Chapter Objectives

- Understand the different roles that transformation of variables can play in pre- and post-processing of data
- To introduce the *power means*, which generalize the geometric, harmonic and arithmetic means
- To build intuition about using transformations appropriately

2.1 The Problem in Data: Multicriteria Evaluation

Let’s start with an exercise (inspired by Lesh et al. [7]). Suppose we are charged with splitting the following 20 students¹ into two “fair” volleyball teams.

Student	Sprint 100 m (seconds)	Height (cm)	Serving (out of 100)	Endurance (out of 30)
Mizuho	15.78	148	94	17
Yukie	21.15	147	94	20
Megumi	14.30	134	91	17
Sakura	19.59	174	88	16
Izumi	10.96	145	93	16
Yukiko	19.17	158	83	12
Yumiko	18.35	157	99	20
Kayoko	14.09	177	82	23
Yuko	27.98	155	93	19
Hirono	16.51	165	85	7
Mitsuko	15.57	137	100	14
Haruka	14.16	162	93	16
Takako	22.40	176	95	15
Mayumi	21.34	153	97	9
Noriko	15.67	140	94	8
Yuka	19.12	155	81	3
Satomi	21.50	147	88	5
Fumiyo	40.29	161	95	19
Chisato	12.34	160	89	26
Kaori	13.38	134	81	16

We could set about this task in a few ways. One approach would be to base our decision on just one of the variables, however obviously this would have the drawback that even if we had two teams that were fairly matched in height, we might end up having all of the fast girls on one team and the slow ones on the other.

If we can get an ‘overall rating’ or ranking of each player, we could then use this overall statistic to split our teams, and one way to determine this rating is by aggregating the variables. However, looking closely at the data we come across a few problems.

Let’s just focus on the first 8 girls in the list. Taking the average of each variable would give the following.

¹The female student names here are borrowed respectfully from Kōshun Takami’s novel *Battle Royale* (although his novel is not related at all to volleyball).

Student	Sprint	Height	Serving	Endurance	AM
Mizuho	15.78	148	94	17	68.55
Yukie	21.15	147	94	20	70.43
Megumi	14.30	134	91	17	64.36
Sakura	19.59	174	88	16	74.50
Izumi	10.96	145	93	16	66.37
Yukiko	19.17	158	83	12	68.06
Yumiko	18.35	157	99	20	73.44
Kayoko	14.09	177	82	23	73.92

2.1.1 Which is Better: Higher or Lower?

One thing that you might notice immediately is that a slower sprinting time contributes to a higher value in the arithmetic mean. Mizuho is a lot slower than Izumi and has similar height, serving score and endurance, however her final rating is better because slower times add more points. Since we are talking about volleyball teams, we probably want to reward quicker sprinting times.

In some cases, it may not be the highest or lowest that becomes ideal but rather a mid-range value. For example, if we are looking at ideal holiday destinations and want to take the climate into account, the best temperature might be described as one that is “not too hot and not too cold”.

2.1.2 Consistent Scales

Megumi has the lowest score here—but that’s largely due to her height. If we look at the range of the heights, not only are the values higher, but Megumi and Kayoko differ by 43 cm. That means in the final arithmetic mean, Megumi will already be almost 11 points lower than Kayoko.

Recall that we usually are able to interpret the arithmetic mean in the same units as the inputs, however here obviously there is no scale over which the arithmetic mean makes sense other than just a score or ‘points’.

2.1.3 Differences in Distribution

Further to the problem of scale, sometimes the distribution should also be taken into account (Fig. 2.1).

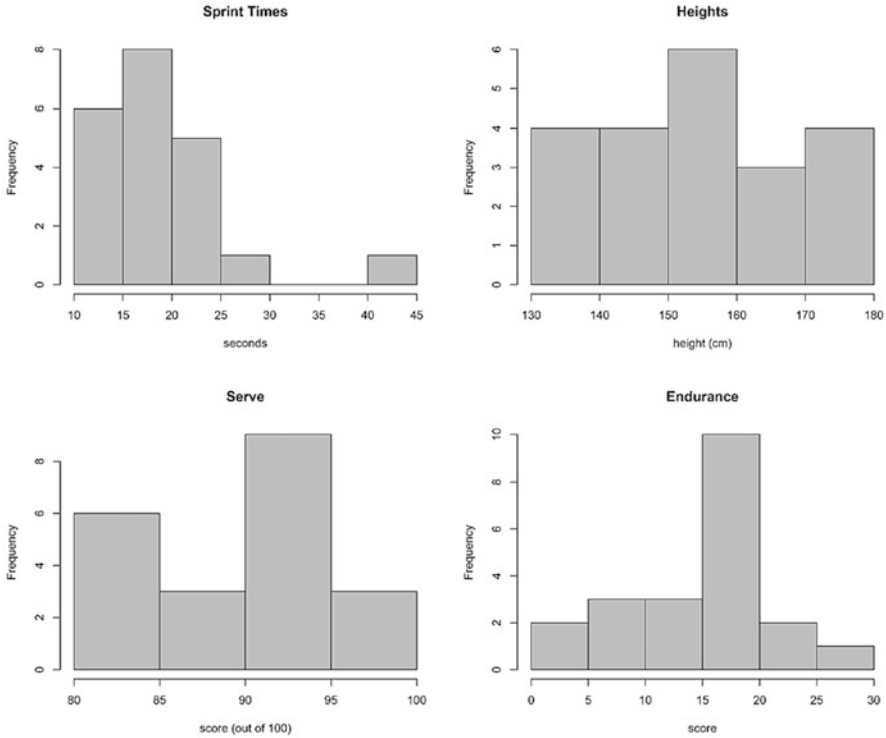


Fig. 2.1 Histograms generated in R showing the distribution of each variable in the Volleyball data

With sprinting times, the majority of students performed their sprint in between 10 and 20 s, however we would think the difference between a 10 and 15 s runner is more significant than the difference between a 20 and 25 s runner. On the other hand, the student who ran the sprint in 40 s may be much slower than everyone else, but we might not want this score in just one variable to have an undue influence on the final result.

Correcting for distributions can be very difficult. As we will see, there are some standard approaches we can use, however none of these is fool-proof and so we are often required to make a judgement call about what is reasonable.

In general, we should consider the following:

- If we are using *aggregation functions*, we need to bear in mind the interpretation of monotonicity. We need to ensure that it makes sense that an increase in the input should result in an increase to the output (e.g. for the students playing volleyball we need to transform the sprinting times);
- Scale and distribution of the data may differ. We may want/need to have all the data transformed to a particular interval, and we also might want to ensure that increases are treated similarly no matter where they are on that scale;

- Specific to averaging aggregation functions, we might want to consider the role that idempotency of the function plays. Does it make sense that a score of 0.8 for every input should result in an output that is also 0.8? For example, in grading students on entrance exams to university, a student that scores above average in every subject might be very rare, so we might actually want a score of 80 % on every subject to correspond with a score of about 95 % overall.
- The data might not be numeric at all. If we assign numeric values, are these reasonable and justified?

2.2 Background Concepts

In this chapter we will start to refer to more than just vectors of values to denote a set of inputs. We will make use of data organized in tables/arrays or matrices.

2.2.1 Arrays and Matrices (\mathbf{X})

In the students' volleyball data, each student (or observation/instance) could be considered to have a vector of attributes. As an example, for the data relating to Yukiko (the 6th student), we have $\mathbf{x} = \langle 19.17, 158, 83, 12 \rangle$. However in some cases we will also want to refer to the data relating to each variable, for example, the *Endurance* variable could be associated with the vector of length 20, $\mathbf{x} = \langle 17, 20, 17, 16, \dots, 26, 16 \rangle$.

As well as Yukiko's height, 158, being an entry of either Yukiko's row vector or the *height* column vector, it can also be thought of as an entry of the table. We will refer to data made up of rows or columns of multiple vectors as 'matrices' or 'arrays'. We will usually use capital bold letters, e.g. \mathbf{X} , to refer to them and specify their dimensions $m \times n$, where m is the number of rows and n is the number of columns. For example, excluding the variable recording the students' names, we can refer to the volleyball data as a 20×4 matrix and label it \mathbf{V} .

2.2.2 Matrix/Array Entries (x_{ij})

We can then identify each entry in terms of its row and column. We write x_{ij} (or $x_{i,j}$ if there are more than 10 columns/rows to avoid ambiguity) to refer to the entry in the i -th row and j -th column. The entry $v_{3,4}$ in the volleyball data then is the 3rd value in the 4th column or the 4th value in the 3rd row, i.e. it corresponds with Megumi's endurance score.

2.2.3 Matrix/Array Rows and Columns (\mathbf{x}_i , \mathbf{x}_j)

However we can also refer to entire columns or rows. We will use the notation \mathbf{x}_i to refer to row vectors and \mathbf{x}_j to refer to columns. Sometimes it wouldn't be clear whether an instance \mathbf{x}_3 refers to a row or a column, so in cases where it could be ambiguous, we will specify instances as explicitly relating to i or j , e.g. $\mathbf{v}_{i=4}$ is the data relating to Sakura while $\mathbf{v}_{j=3}$ relates to the vector of data pertaining to the *Serving* variable.

2.3 Negations and Utility Transformations

Negation functions transform the data so that high values become low and low values become high. If the values are given over the unit interval (between 0 and 1), then the **standard negation** is given by

$$N(t) = 1 - t.$$

The standard negation is an example of a strict negation (e.g. see [2, 3, 6]), the term 'strict' meaning the same thing it does when we talk about monotone increasing behaviour.

Definition (informal) 2.1 (Strict Negation). A strict negation is a strictly decreasing function of one variable that has a maximum and minimum output that are the same as the domain of the inputs. So if the data we are transforming ranges from 1 to 10, then after we apply the transformation, the data should still range from 1 to 10, however the low values will now be high and the high values low.

Definition 2.1 (Strict Negation). A strict negation N defined over a real interval $[a, b]$ is a function that:

- is monotone decreasing, i.e. if $x < y$ then $N(x) > N(y)$; and
- satisfies boundary conditions $N(a) = b$ and $N(b) = a$.

Side Note 2.1 *In research literature, there is also the concept of a strong negation, which is one that satisfies the property of "involution". This means that if we perform a negation of the negation then we get the original value.*

$$N(N(t)) = t.$$

Notation Note Function of a function

When we have a function of a function, written either as something like $f(g(x))$ or $f \circ g(x)$, it means that we replace the variable in f with the whole function $g(x)$. For example if $f(x) = x^3$ and $g(x) = x^2 + 1$ then $f(g(x)) = (g(x))^3 = (x^2 + 1)^3$.

In the case of the standard negation, $N(t) = 1 - t$, and so taking the standard negation of the standard negation will give $N(N(t)) = 1 - (1 - t) = 1 - 1 + t = t$.

However as is the case with our volleyball team data, we will often be dealing with data that is not expressed over the unit interval, and we therefore need to pay attention to the range of values. For the girls' volleyball team, we want to transform the *Sprint* variable. The data ranges from 10.96 to 40.29. It would be fine to use a negation like:

$$N(t) = 40.29 - t + 10.96 \quad \text{or} \quad N(t) = 51.24 - t$$

The result of this negation is depicted in Fig. 2.2.

We can note that this satisfies the boundary conditions since $N(10.96) = 40.29$ and $N(40.29) = 10.96$.

In general, if our interval is $[a, b]$, we can use

$$N(t) = b - t + a.$$

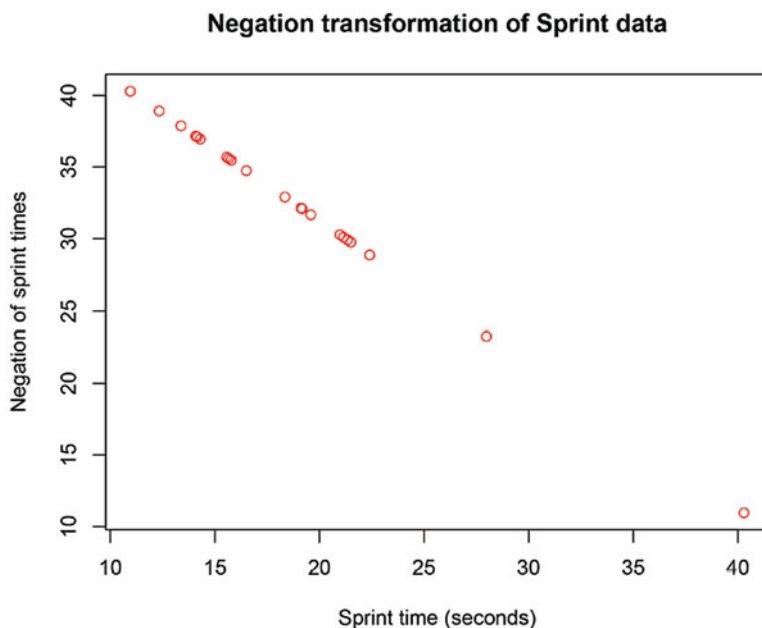


Fig. 2.2 Scatterplot showing sprint times and their negation which can then be interpreted so that 'higher means better'

Example 2.1. Show that the negation $N(t) = 51.24 - t$ over the interval $[10.96, 40.29]$ is a *strong negation*.

Solution. Since the function uses $-t$, it is clear that it will be monotone decreasing since the larger the input, the smaller the output. We just need to verify that it satisfies the property of involution.

Let's check:

$$\begin{aligned} N(N(t)) &= 51.24 - N(t) \\ &= 51.24 - (51.24 - t) \\ &= 51.24 - 51.24 + t \\ &= t \end{aligned}$$

So this is a strong negation over the given interval.

We can have other negations too, for example if our data is over the unit interval, $N(t) = 1 - t^2$ is a strict negation (but not a strong negation) and $N(t) = \sqrt{1 - t^2}$ is a negation that is both strong and strict.

Side Note 2.2 *With our sprint data, there is no particular reason why we need the transformed data to be given over the same range. We would now interpret the times as ‘time under 51.24’ or, more generally, that the values now represent a kind of utility or points/rewards system. In the following subsection we will look at getting all of our data to a consistent scale—so sometimes we may first transform the data to a unit interval and then apply a negation (or vice versa).*

As well as our data going in the wrong direction, we might have situations where ‘good’ might actually refer to some intermediate value, with values being worse as they get further away. In this case, we are looking for the “just right” value (i.e. as with the porridge and beds in the *Goldilocks and the Three Bears* children’s story²) (Fig. 2.3).

Whether we use the standard negation or something more complicated, our main concern is usually with the tendency of our output to increase or decrease with respect to our input variable. The standard negation will preserve most of the characteristics of our data distribution, e.g. if the majority of data lie within a given range then this density should stay the same if we only use a standard negation.

Example 2.2. What will be the transformed values for the *Sprint* variable for the first 8 students in the volleyball data using $N(t) = 51.24 - t$?

²On finding the three bears’ uneaten bowls of porridge, Goldilocks tastes and remarks that the father’s bowl of porridge is too hot, the mother bear’s is too cold, but the baby bear’s bowl is “just right” (so she eats the whole bowl).

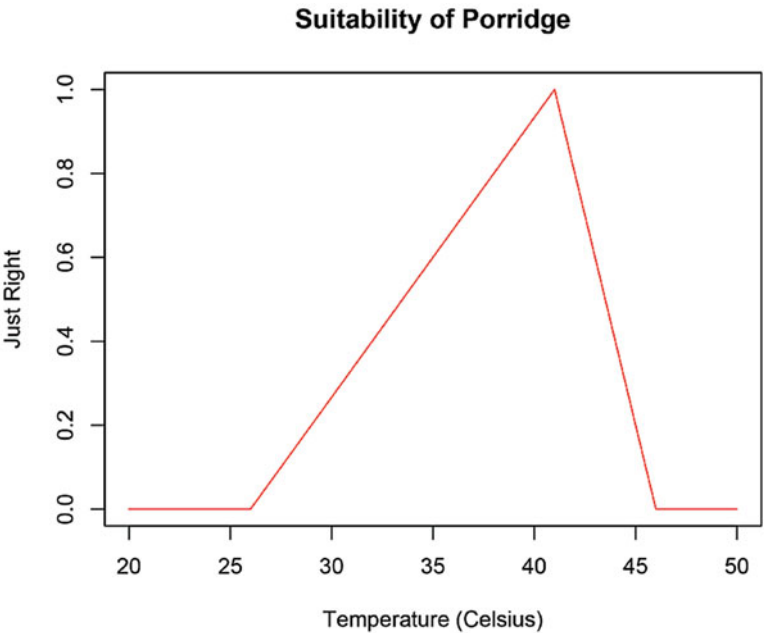


Fig. 2.3 Plot showing a transformation from raw temperatures to a degree of suitability, where porridge served too hot (above 46°) or too cold (below 26°) has a zero level of ‘suitability’. The transformed scores would then satisfy the ‘higher is better’ property needed for monotonicity to make sense

Solution. Applying the transformation leads to the transformed scores as follows,

Student	Sprint (time under 51.24 s)	Student	Sprint (time under 51.24 s)
Mizuho	35.46	Izumi	40.28
Yukie	30.09	Yukiko	32.07
Megumi	36.94	Yumiko	32.89
Sakura	31.65	Kayoko	37.15

Example 2.3. Suppose we are evaluating employees in terms of customer feedback, their average service time and the number of customers they have handled. We have 5 employees to score with the following data. The customer feedback has been recorded as either positive or negative.

	Customers	Positive	Negative	Avg. time
Pete	12	1	6	16.2
John	73	10	6	18.7
Richard	59	38	12	7.8
George	62	10	6	6.8
Paul	44	35	7	7.3

What are appropriate negations that could be used for this data?

Solution. We can assume that since the number of customers handled and the number of positive feedback responses should contribute positively to an overall evaluation, we don't need to apply negations for these.

For negative feedback responses, the data ranges from 6 to 12. We can use

$$f(t) = 18 - t$$

so that the values in the transformed variable increase with fewer negative responses. The new data will be $\langle 12, 12, 6, 12, 11 \rangle$.

For the average time, the data ranges from 6.8 to 18.7. Since 'quicker' is usually better, we can transform this data using

$$f(t) = 25.5 - t.$$

The transformed values will be $\langle 9.3, 6.8, 17.7, 18.7, 18.2 \rangle$.

Note that our transformed data maintains the same data range and we have not yet addressed the different scale of the variables.

2.4 Scaling, Standardization and Normalization

If all our data can take values over a consistent range and have a consistent interpretation, for example, when we are finding the average measurement for a group with respect to one variable (e.g. height, long jump, amount of time spent sleeping etc.) then there would usually be no need to change the scale. We can take an average (using the arithmetic mean, the geometric mean, the median, or a number of other averages) and the output can be interpreted in the same units.

On the other hand, if the source and type of inputs vary (as is the case with each vector associated with a student in the volleyball data), then this is no longer possible. If we do not have a consistent scale, more varied inputs may have an undue influence in the aggregation step. We should also bear in mind that, whether or not the scale is consistent, if the type of inputs differs we should be careful about how we interpret our aggregated value. Usually it comes to represent a 'score' or overall

evaluation, in which case our inputs should then be interpreted as contributing partial scores or evaluations with respect to different criteria.

The simplest technique for transforming each variable, which also preserves the distribution features, is to use linear transformations that scale the data to the unit interval. We will refer to this process as linear feature scaling.

Definition (informal) 2.2 (Linear Feature Scaling). When we have different variables or features, linear feature scaling for each variable is the process of transforming the data so that it ranges over the unit interval using only addition/subtraction and multiplication. A value a is subtracted from each entry and then we multiply by a factor b . For example, if we had heights ranging from 150 to 200 cm and weights ranging from 50 to 60 kg:

For the heights—we could subtract 150 (our a value) so that now the values range from 0 to 50, and then divide by 50 (our b value). Now all the values will be scaled to range from 0 to 1.

For the weights—we could subtract 50 (our a for this feature) and then divide by 10 (our b). This would mean both variables/features now have a consistent scale of 0 to 1.

Definition 2.2 (Linear Feature Scaling). For a set of values $\mathbf{x}_j = \langle x_{1,j}, x_{2,j}, \dots, x_{m,j} \rangle$ relating to a single feature, we let $a = \min(\mathbf{x}_j)$ and $b = \max(\mathbf{x}_j) - \min(\mathbf{x}_j)$. Each $x_{i,j}$ in \mathbf{x}_j can be scaled so that it takes a new value $x'_{i,j}$ over the unit interval using the transformation $x'_{i,j} = f(x_{i,j})$, where f is the single-variate function

$$f(t) = \frac{t - a}{b}.$$

Notation Note Transformations

With the index notation here, the j remains the same. This indicates that we are only considering the data pertaining to one feature (one of the columns in our matrix/array). The values for a and b would generally differ for each j , but they should be the same for each i when j is fixed (assuming that columns represent features and rows represent cases). We have also used the notation x' to denote a transformed or ‘updated’ value. We could also write something like,

$$x_{\text{new}} = \frac{x_{\text{old}} - x_{\min}}{x_{\max} - x_{\min}}.$$

We still use t as the variable in the function $f(t)$. Just remember that the entry of the matrix $x_{i,j}$ is what we substitute for t when we write $f(x_{i,j})$.

Example 2.4. What will be the transformed values for the first 8 students if we scale the *Height* variable to the unit interval using linear feature scaling?

Solution. The tallest height (for the 20 students) is 177 and the shortest is 134. We therefore can use $f(t) = \frac{t-134}{43}$ to give the following heights.

Student	Height (score between 0 and 1)	Student	Height (score between 0 and 1)
Mizuho	0.33	Izumi	0.26
Yukie	0.30	Yukiko	0.56
Megumi	0	Yumiko	0.53
Sakura	0.93	Kayoko	1

As mentioned already, linear feature scaling will preserve the essential features of the distribution, however sometimes this can cause problems. For example, if we have a single very large outlier, then now most of our values will be scaled so that they are close to zero. If you have studied statistics, you might have come across the idea of standardization or Z-scores.

Definition (informal) 2.3 (Standardization). Standardization usually refers to the process of subtracting the mean and dividing by the standard deviation for each value, which centers the data at zero and scales the standard deviation to 1. Applying this technique to different variables makes them commensurable, and we then essentially interpret them in terms of how abnormal they are. *Having a good feel for interpreting standardized values and averages of standardized values requires familiarity with the normal distribution, which is beyond the scope of the topics we cover but can be found in any introductory statistics book (e.g. see [4] for more information as well as a number of standard statistical techniques for transforming data).

Definition 2.3 (Standardization). For an input vector $\mathbf{x}_j = \langle x_{1,j}, x_{2,j}, \dots, x_{m,j} \rangle$ where $SD(\mathbf{x}_j) = \sqrt{\sum_{i=1}^n \frac{(x_{i,j} - \mu)^2}{n-1}}$ is the sample standard deviation of \mathbf{x}_j (or the true mean (μ) and standard deviation (σ) may be known *a priori* for the wider population of data observations), standardization involves transforming each $x_{i,j}$ using $x'_{i,j} = f(x_{i,j})$, where

$$f(t) = \frac{t - AM(\mathbf{x}_j)}{SD(\mathbf{x}_j)}.$$

Notation Note Transformations

Once again, we could express this transformation as an update process from old to new values,

$$x_{\text{new}} = \frac{x_{\text{old}} - AM(\mathbf{x}_j)}{SD(\mathbf{x}_j)}.$$

This is an appropriate transformation if our data all follow a normal distribution and differ only in terms of their mean and standard deviation. Following the

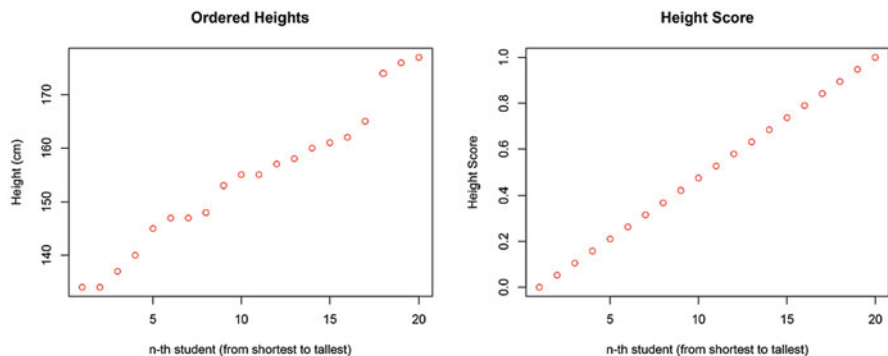


Fig. 2.4 The student data for the height variable before (*left*) and after (*right*) rank-scaling

standardization step, our data would then usually fall between -2 and 2 , with approximately 5 % lying above or below. If we wanted our data to lie between 0 and 1 we could then use linear feature scaling on the standardized data. Assuming there are no extreme outliers, one transformation that in most cases will allow normally distributed data to be scaled to the unit interval is:

$$f(t) = 0.15 \left(\frac{t - \text{AM}(\mathbf{x})}{\text{SD}(\mathbf{x})} \right) + 0.5$$

The 0.15 multiplying the usual standardization formula will first scale the data so that 99.7 % of the data should fall between -0.45 and 0.45 (three standard deviations) and then adding 0.5 will shift the interval to $[0.05, 0.95]$.

You will note the similarity between the form of these two types of transformation, which can both be considered as kinds of ‘normalization’. They only involve simple operations on the data and essentially preserve the features of the distribution in a relative way. Another simple transformation that can be applied is rank-scaling (Fig. 2.4). In this case, we are only interested in the relative order of the data—not how far apart they are. Rank-scaled data also enables the data to be easily interpreted in terms of percentiles, i.e. a score of 0.9 means the value is higher or ‘better’ than 90 % of the other evaluations.

Definition (informal) 2.4 (Rank-Scaling). Rather than the actual scores, the relative ordering or ranking might be what is more important. In rank-scaling we first order the data and then allocate scores based on the percentage of data below each value. E.g. for $\mathbf{x} = \langle 102, 37, 10, 39, 28 \rangle$ we would have $\langle 1, 0.5, 0, 0.75, 0.25 \rangle$.

Definition 2.4 (Rank-Scaling). For an input vector $\mathbf{x}_j = \langle x_{1,j}, x_{2,j}, \dots, x_{m,j} \rangle$, let $O_j(x_{i,j})$ denote the rank of $x_{i,j}$ with respect to the other entries in \mathbf{x}_j , so that

$O_j(x_{ij}) = 1$ means that x_{ij} is the ‘best’ or highest score, $O_j(x_{ij}) = 2$ means x_{ij} is the second highest, and so on. We can transform each x_{ij} into a score out of 1 using $x'_{ij} = f(x_{ij})$, where

$$f(t) = \frac{m - O_j(t)}{m - 1}.$$

We can also use the ranks themselves if we prefer. Ranking transformations can be useful for data that is ordinal or naturally refers to a scale that is not necessarily numerical. If there are ties, usually each alternative in the tied position is allocated the average of their ranks. For example, if 3 alternatives 4th, 5th and 6th were tied, they would each receive a rank of 5. If two alternatives in positions 8 and 9 were tied, they would each receive a rank of 8.5.

Side Note 2.3 *We should bear in mind, however that we are imposing a numerical interpretation that may not be valid. For example, in iMDb movie ratings, it is not necessarily the case that the difference between 9/10 and 10/10 is the same as the difference between 6/10 and 7/10. What we need to be aware of when we perform rank-transformations (or in fact any kind of utility transformation) is that we are essentially awarding a ‘score’ or a ‘partial score’ based on that variable or feature. Whenever we aggregate data with averaging functions, low values in one variable can be compensated for by high values in others. We should think carefully about the implications of our transformations with respect to this property.*

2.5 Log and Polynomial Transformations

Other common transformations used as part of some statistical techniques includes the use of increasing functions like

$$f(t) = \ln t,$$

or

$$f(t) = t^2.$$

Such functions can help if data are exponentially distributed or skewed.³

The log function in particular is useful for data that can have a few very large inputs (Fig. 2.5). Incomes, populations in ecology and academic journal citations are all examples of data that could exhibit such properties.

³Characterizing distributions is beyond what we cover, so we will just quickly note that exponentially distributed data has most of the data gathered towards very low values with fewer and fewer high values. The high values, however can be very high.

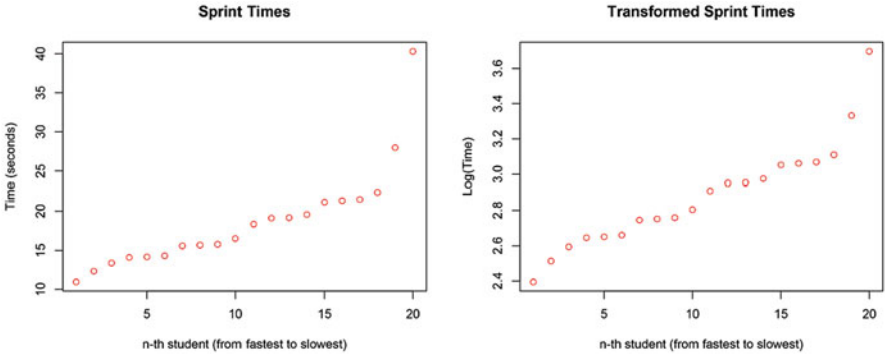


Fig. 2.5 A log transformation of the sprint variable. The relative difference between lower values is increased and higher values is decreased. The natural log (with the base e) is used

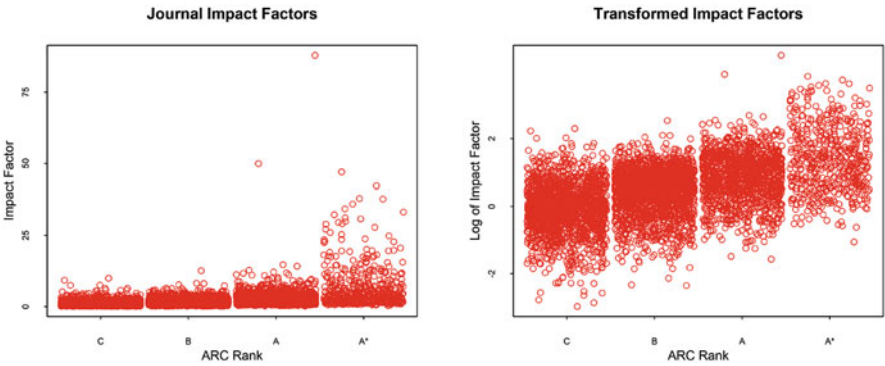


Fig. 2.6 Journal impact factor data for 5311 journals organized by the 2011 ARC rankings (and randomly within ranks). The log transformation (*on the right*) makes it easier to see the spread of the different classes

Although the general difference between the values is somewhat maintained, values toward the lower end of the scale become more spread out while higher values are pushed closer together. Another example is shown in Fig. 2.6. In this case, the data relates to the Australian Research Council journal rankings released in 2011 and the impact factors (the ratio of citations to a journal’s papers within the last two years to articles published) are shown with journals organized into their evaluated rankings.⁴ The transformation allows for the outliers to become less pronounced.

Polynomial function transformations $f(t) = t^2, f(t) = t^{\frac{1}{2}}$ can have a similar (but less drastic) effect, allowing skewed distributions to become more symmetric.

⁴See [1] for a study on predicting these rankings from citation indices. The data shown is available at <http://aggregationfunctions.wordpress.com/data-sets>.

For t^p , $p > 1$ can be used when there are fewer very high values (positive skew), while $0 < p < 1$ can be used if the majority of data is gathered in the high range with fewer very low values (negative skew).

Example 2.5. For which of the variables in the volleyball data would it make sense to use a log transformation?

Solution. Knowing when to use a log transformation is not straightforward. In some statistical approaches when looking for relationships between variables, the aim is usually to use transformation functions like log and then check if the relationship appears linear.

For our sprint times (before being transformed by a negation) we can see that many of the values are clustered together toward the lower end of the scale and there are is one isolated value in the higher end. A log transformation will spread out the lower values and bring the outlier closer to the main group, however it might be preferable to just remove the outlier—without it, the data is more or less normal.

In general, log transformations make sense when the data looks similar to the sprint times when represented as a histogram, however with the majority of values in the first interval close to zero (or the minimum).

2.6 Piecewise-Linear Transformations

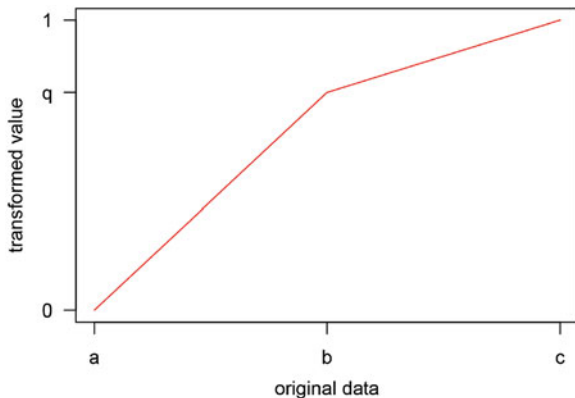
If we have good knowledge about the data and distribution, we might want to apply custom transformations that spread out the data in a way tailored to our application. The easiest of these (although it can still be complicated) is to construct a transformation from either linear or even quadratic ‘pieces’. We already saw a similar tactic used for transformation of the porridge temperatures to a degree of how “Just Right” they are.

When doing this, we usually split our domain into subdomains and our aim is that on the border of each domain the transformation functions are equal.

Definition (informal) 2.5 (Piecewise-Linear Transformations). The piece wise-linear transformations we use will usually be monotone functions where the domain is split into intervals and a different linear function (i.e. $f(t) = mt + c$) is used to transform the data over each interval. The functions should connect on the border of the domains for the transformations to be continuous. For example, if we split the unit interval in half, and use $f(t) = 0.4t$ over the sub-interval $[0, 0.5]$, and $f(t) = 1.6t - 0.6$ over the sub-interval $[0.5, 1]$, then the effect will be that the lower half of the data is pushed closer together and the upper half will be stretched.

The following allows us to scale the data to the unit interval with two linear functions (see Fig. 2.7).

Fig. 2.7 Diagram of a piecewise-linear function with 2 pieces. Everything between a and b is linearly scaled to range between 0 to q and everything from b to c scales to a value between q and 1



Definition 2.5 (Piecewise-Linear Transformations (2 Pieces)). For a set of evaluations $\mathbf{x}_j = \langle x_{1,j}, x_{2,j}, \dots, x_{m,j} \rangle$ given over $[a, c]$, we split the domain into two sub-intervals, $[a, b)$ and $[b, c]$. Let $q \in [0, 1]$ be the transformed value we want our variable to take when $x_{i,j} = b$. Letting $x'_{i,j} = f(x_{i,j})$ with the following piecewise function scales the data to the unit interval.

$$f(t) = \begin{cases} q \frac{t-a}{b-a} & , \quad a \leq t < b, \\ q + (1-q) \frac{t-b}{c-b} & , \quad b \leq t \leq c. \end{cases}$$

Notation Note Cases

The large ‘{’ indicates cases, while the right hand inequations $a \leq t < b$ and $b \leq t \leq c$ specify intervals that tell us when each case should be used. When our input is less than b , we use the first function and the second when it is b or above. The round bracket in the interval notation is the same as using a strictly less than symbol ‘<’, i.e. indicating that the value is not included in the interval but everything below it is. We need to have one of the borders open (with a round bracket) to make it clear which function we should use when $x_{i,j} = b$, however we could also use $[a, b]$ and $(b, c]$. Since both functions should be equal at b , it doesn’t matter which.

Similarly we can create a piecewise-linear transformation if want to split our function into three sections.

Definition 2.6 (Piecewise-Linear Transformations (3 Pieces)). For a set of evaluations $\mathbf{x}_j = \langle x_{1,j}, x_{2,j}, \dots, x_{m,j} \rangle$ given over $[a, d]$, we split the domain into three sub-intervals, $[a, b)$ and $[b, c]$ and $[c, d]$. Let $q \in [0, 1]$ be the transformed value we want our variable to take when $x_{i,j} = b$ and $r \in [q, 1]$ be the value we want our variable to take when $x_{i,j} = c$ with $a < b < c < d$. Then we can apply the following transformation function.

$$f(t) = \begin{cases} q \frac{t-a}{b-a}, & a \leq t < b, \\ q + (r-q) \frac{t-b}{c-b}, & b \leq t < c, \\ r + (1-r) \frac{t-c}{d-c}, & c \leq t \leq d. \end{cases}$$

We can continue this pattern for any number of sub-intervals. We just need to have a set of points where we know what the transformed output should be.

Whenever the multiplier is less than 1 over a given sub-interval, the data will be pushed together, while whenever the multiplier is greater than 1, the values will be spread further apart from one another. You will notice that the fractions involved in the expressions are of a similar form to the linear feature scaling and standardization formulas.

The following piecewise function was used with journals data in [1] to perform a custom scaling of the impact factors, which were later used along with other variables to estimate the overall ranking of each journal. In this ranking exercise, the A* ranked journals were considered to be in the top 5 % of journals, the A ranked journals were in the top 20 %, B in the top 50 % and C otherwise. The domain was split into sub-intervals corresponding with the median impact factor score for each journal and the values in the unit interval these values were chosen so that the percentiles values would fall between them, i.e. $y_C = 0.3, y_B = 0.7, y_A = 0.9$.

$$x'_{i,j} = \begin{cases} y_C \frac{x_j - \min(x_j)}{C_{Med} - \min(x_j)}, & x_j < C_{Med}; \\ y_C + (y_B - y_C) \frac{x_{i,j} - C_{Med}}{B_{Med} - C_{Med}}, & x_j < B_{Med}; \\ y_B + (y_A - y_B) \frac{x_{i,j} - B_{Med}}{A_{Med} - B_{Med}}, & x_j < A_{Med}; \\ y_A + (1 - y_A) \frac{x_{i,j} - A_{Med}}{A^*_{Med} - A_{Med}}, & x_j < A^*_{Med}; \\ 1 & \text{otherwise.} \end{cases} \quad (2.1)$$

The plot of one of the piecewise functions created in this way is shown in Fig. 2.8. Figure 2.9 shows the impact factors and transformed scores separately so that differences in the distribution can be more clearly seen.

Example 2.6. Suppose we have data for one variable that ranges from 82 to 109, how can we transform it with a piecewise-linear function such that anything above 100 has a value of between 0.8 and 1, while anything below 100 ranges from 0 to 0.8?

Solution. We can use two functions, one to operate over the domain [82, 100] and one for values the values in (100, 109]. Using the formula for a 2-piece function where $[a, b] = [82, 109]$ and $q = 100$ we will have,

$$f(t) = \begin{cases} 0.8 \frac{t-82}{18}, & t \leq 100, \\ 0.8 + 0.2 \frac{t-100}{9}, & \text{otherwise.} \end{cases}$$

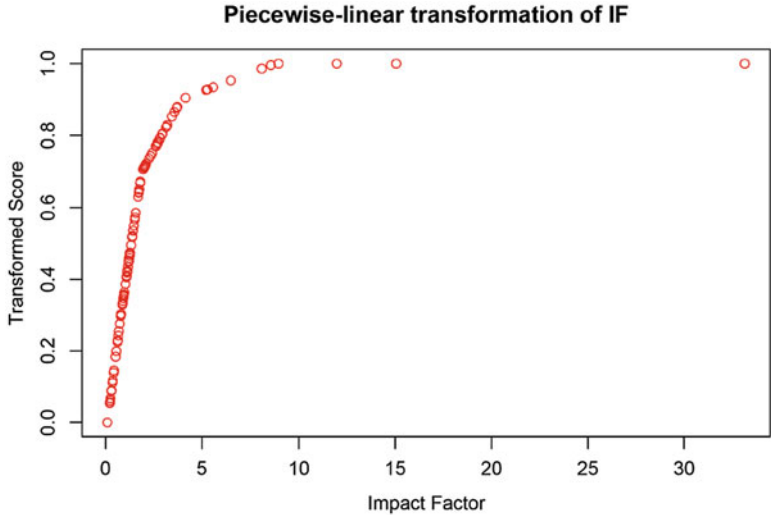


Fig. 2.8 Plot showing an example of a piecewise-linear transformation of the form in Eq. (2.1). For a sample of 100 journals, we assume the medians for each rank are denoted by 0.797, 1.877, 3.938 and 8.779 respectively, then assign these our desired outputs of 0.3, 0.7, 0.9 and 1

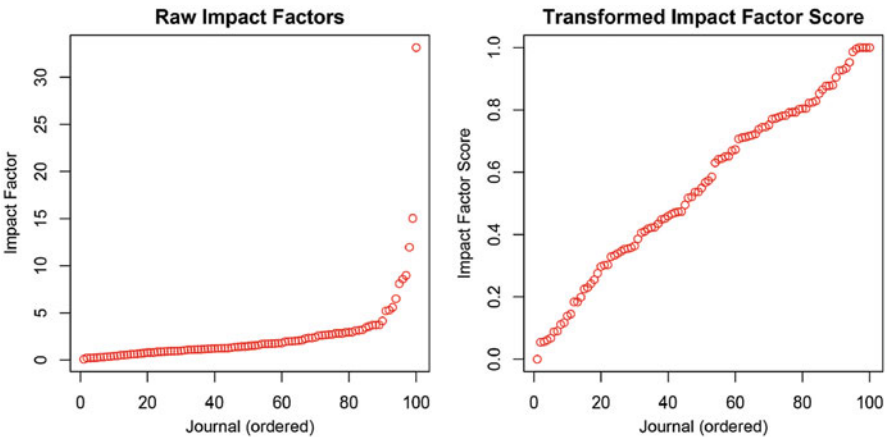


Fig. 2.9 Plots showing the ordered data for the raw impact factors (*left*) and transformed scores (*right*). The effect is that the relative differences for the lower 85 journals are increased while those of the upper 15 are pushed closer together

We can see that at $t = 100$, both values give the output of 0.8, while anything above will be between 0.8 and 1.

2.7 Functions Built from Transformations

Transformation functions can also be used in the construction of new functions, whose properties can be interpreted in light of the transformations.

One such function is called the **dual** aggregation function, which is built from a negation.

Definition (informal) 2.6 (Dual Aggregation Function). The dual aggregation function uses a negation (usually the standard negation). First, all of the inputs are transformed using that negation, the data is aggregated, and then finally the negation is used on the result. For example, using the arithmetic mean and the standard negation on the inputs $\langle 0.3, 0.9 \rangle$, the negation gives $\langle 0.7, 0.1 \rangle$, the average of these is 0.4, and using the negation again gives 0.6. Note that this is the same as the average of the two original values (this always happens for the arithmetic mean but not in general for duals of other aggregation functions).

Definition 2.7 (Dual Aggregation Function). For an aggregation function A with inputs given over the unit interval $[0, 1]$, its dual A^d (using the standard negation) is given by:

$$A^d(\mathbf{x}) = 1 - A(1 - x_1, 1 - x_2, \dots, 1 - x_n).$$

For averaging aggregation functions, the dual will also be an averaging aggregation function, so we still have $A(a, a, \dots, a) = a$ and $A(b, b, \dots, b) = b$ and the final output bound between the minimum and maximum inputs.

In fact the dual of the arithmetic mean is the arithmetic mean itself, and so in this case, taking the dual does nothing at all. We can demonstrate this for two arguments.

$$\begin{aligned} AM^d(x_1, x_2) &= 1 - AM(1 - x_1, 1 - x_2) \\ &= 1 - \frac{1}{2}(1 - x_1 + 1 - x_2) \\ &= 1 - \frac{2}{2} + \frac{x_1 + x_2}{2} = \frac{1}{2}(x_1, x_2). \end{aligned}$$

The arithmetic mean is not affected because it treats inputs the same whether they are at the higher or lower end of the input range. However if we have a function that tends to be affected more by high inputs, its dual will be more affected by low inputs instead. As an example, the dual of the geometric mean has an absorbent element of 1 (i.e. if *any* of the inputs are 1, then the output will immediately be 1) (Fig. 2.10).

The dual function essentially exhibits reciprocal properties to the original function, but it remains monotone increasing.

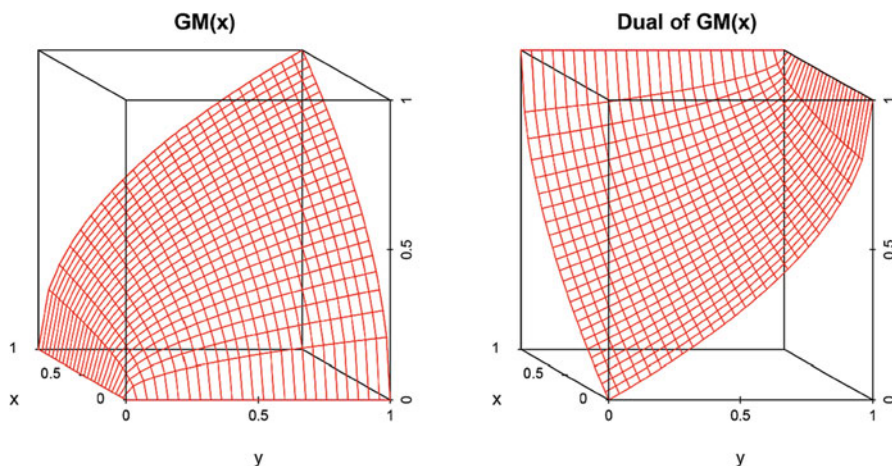


Fig. 2.10 The geometric mean (*left*) and its dual (*right*). The dual has an absorbent element of 1

Example 2.7. Define the dual of the geometric mean explicitly for two arguments.

Solution. Using the formula for construction of the dual with the geometric mean $GM(x_1, x_2) = \sqrt{x_1 x_2}$ gives,

$$GM^d(x_1, x_2) = 1 - \sqrt{(1 - x_1)(1 - x_2)}.$$

2.8 Power Means

Previously we saw that transforming data could be used to help address distributions that were skewed or included extreme values. There are special families of means built from similar transformations.

The power means constitute one such family, based on a parameterized transformation $f(t) = t^p$ where the parameter p can be any real number from negative infinity to infinity. Lower values of p result in outputs that are more sensitive to lower inputs, while high values of p result in power means that tend more toward the higher inputs. Note here that as opposed to our previous application of transformations, here all the inputs are transformed in the same way.

The difference between using transformations as we have done previously and using the arithmetic mean on these transformed values is that the power mean makes use of an inverse function, which returns outputs to their original scale. The upshot of this is that the resulting function will be idempotent and averaging, meaning

that we can still interpret the output in the same way we interpret the inputs, and potentially use the same units if the inputs are all of the same type.

Definition (informal) 2.7 (The Power Mean). When calculating the power mean, we first transform all the inputs using $f_1(t) = t^p$, after which we take the arithmetic mean, and finally, use the transformation $f_2(t) = t^{1/p}$ on the result. This latter function f_2 is the inverse of f_1 . As an example, if we have $\mathbf{x} = \langle 2, 3 \rangle$, then using the power $p = 2$ we would find the average of 2^2 and 3^2 , which is $(4 + 9)/2 = (6.5)$. Raising 6.5 to the power of a half (or taking the square root) gives approximately 2.55, which is slightly higher than the arithmetic mean of 2 and 3 (which would be 2.5). Here we assume that all of the inputs are greater than or equal to zero.

Definition 2.8 (The Power Mean). For an input vector $\mathbf{x} = \langle x_1, x_2, \dots, x_n \rangle$, with $x_i \geq 0$ for all i , the power mean is

$$PM_p(\mathbf{x}) = \left(\frac{1}{n} \sum_{i=1}^n x_i^p \right)^{\frac{1}{p}} = \left(\frac{x_1^p + x_2^p + \dots + x_n^p}{n} \right)^{\frac{1}{p}}.$$

Notation Note Powers

Remember that raising a value to a fractional power is the same as taking the n -th root. Note here as well that we are not thinking about the power mean transformations x^p in terms of scaling or transforming variables, since these inputs may correspond with different criteria. In general, we assume that any scaling or normalization has already taken place to make the data commensurable.

The power mean is not just one function but a family of functions, with each member being determined by the value of p .

For $p = 1$, we obtain the arithmetic mean; for $p = -1$, we have the harmonic mean; and for $p = 0$ (a limiting case⁵) we actually obtain the geometric mean. Furthermore, as $p \rightarrow -\infty$ we approach the minimum function and $p \rightarrow \infty$ we approach the maximum. We summarize these in the table below.

⁵By ‘limiting case’ we are making reference to the mathematic concept of the limit. In the case of $p = 0$, as the parameter p gets closer and closer to 0, the outputs of the power mean become more and more similar to the outputs of the geometric mean, however we could never have p being equal to exactly 0, because then we would have $\frac{1}{0}$ as our fractional power and dividing by zero is not allowed. A value of $p = 0.00000001$ will give results that are numerically very similar to using the standard formula for the geometric mean. We often write “as $p \rightarrow 0$ ”, which means “as p approaches zero”.

Special case	p	
Maximum	∞	$\max(x_1, x_2, \dots, x_n)$
Quadratic mean	2	$\sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2}$
Arithmetic mean	1	$\frac{1}{n} \sum_{i=1}^n x_i$
Geometric mean	0	$\left(\prod_{i=1}^n x_i \right)^{1/n}$
Harmonic mean	-1	$\left(n \sum_{i=1}^n x_i^{-1} \right)^{-1}$
Minimum	$-\infty$	$\min(x_1, x_2, \dots, x_n)$

Side Note 2.4 *The power mean is sometimes referred to as a generalized mean, since it includes many means as special cases.*

Power means are averaging aggregation functions for any value of p . This means they are always monotone and satisfy the boundary conditions. They are also homogeneous for any p , however it is only in the case of the arithmetic mean ($p = 1$) that they will be translation invariant.

Guided by our special cases, we can consider how the function will treat inputs depending on the value of p .

- Larger values of p (and greater than 1) will mean that the output of the function is more influenced by larger values in the input set than by smaller ones. So the output will be ‘dragged’ towards higher inputs. When p becomes infinitely large, the output will simply be the highest input value.
- Values of p below 1 will drag the function towards lower inputs. Furthermore, if $p \leq 0$, we obtain functions that will have an output of zero if any of the inputs are equal to zero (e.g. we already saw that the geometric mean ($p = 0$) and harmonic mean ($p = -1$) had an *absorbent element* of 0). Values of p between 0 and 1 will still tend toward lower values but won’t have this absorbing element property.

Example 2.8. Compare the values of the power mean for $p = -5, 0, 1$ and 3 for the input vector $\mathbf{x} = \langle 0.3, 0.9, 1 \rangle$.

Solution. Calculating these values respectively gives outputs of approximately 0.373, 0.646, 0.733 and 0.837. It is clear that while $p = -5$ pushes the output closer to the 0.3, as it is increased the output draws closer to the 1.

2.9 Quasi-Arithmetic Means

An even more general family of averaging functions is the quasi-arithmetic means. Rather than a transformation of the variable using $f(t) = t^p$, the transformation f can be of any form. It is then referred to as a generating function, which we will denote by $g(t)$. The generating function g can be almost anything that is defined over the domain of the inputs, provided it is monotone and has an inverse (e.g. with $f(t) = t^p$, the inverse was $t^{1/p}$, i.e. it is a function that ‘undoes’ the other). We can even use piecewise-linear functions. We will not focus in great detail on quasi-arithmetic means since the power means are broad enough for most of our needs, however the following definition is provided for those interested and further information can be found in any of the key references [2, 3, 5, 6, 9].

Definition 2.9 (The Quasi-Arithmetic Mean). For an input vector \mathbf{x} and suitable generating function g , the quasi-arithmetic mean is given by

$$QAM_g(\mathbf{x}) = g^{-1} \left(\frac{1}{n} \sum_{i=1}^n g(x_i) \right).$$

Of course if g is t^p then we obtain the power means as a special case. As well as being a special case of the power mean, the geometric mean can also be expressed as a quasi-arithmetic mean with respect to the generator $g(t) = \ln t$. Rather than relying on the idea of the limit, in this case we can show that using $\ln t$ and its inverse recovers our original expression for the geometric mean as the product of all arguments raised to the power $\frac{1}{n}$. The key to this is the log law that adding logarithms together is the same as taking the log of their product, i.e. $\log a + \log b = \log(ab)$. If $g(t) = \ln t$, then $g^{-1}(t) = e^t$ and we will have:

$$\begin{aligned} \text{GM}(\mathbf{x}) &= e^{\left(\frac{1}{n} \sum_{i=1}^n \ln x_i \right)} \\ &= e^{\frac{1}{n} \ln \left(\prod_{i=1}^n x_i \right)} = e^{\ln \left(\prod_{i=1}^n x_i \right)^{\frac{1}{n}}} \\ &= \left(\prod_{i=1}^n x_i \right)^{\frac{1}{n}}. \end{aligned}$$

Notation Note Logarithms

The notation $\ln t$ means $\log_e t$ or log to the base e . We could actually use a logarithm with any base, since its inverse would still cancel out the way it does with $\ln t$ and e^t .

2.10 Summary of Formulas

Strict Negation

Over a real interval $[a, b]$, a strict negation N is a function that satisfies

$$\text{If } x < y \text{ then } N(x) > N(y) \text{ (monotonicity)} \quad (2.2)$$

$$N(a) = b, N(b) = a, \text{ (boundary conditions)}$$

Linear Feature Scaling

$$f(t) = \frac{t - a}{b}, \quad (2.3)$$

$$a \leq \min(\mathbf{x}), \quad b \geq \max(\mathbf{x}) - \min(\mathbf{x})$$

Standardization

$$f(t) = \frac{t - \text{AM}(\mathbf{x})}{\text{SD}(\mathbf{x})}, \quad (2.4)$$

Rank-Scaling

$$f(t) = \frac{m - O(t)}{m - 1}, \quad (2.5)$$

(m is the number of data, $O(t)$ is the ranking among the data relating to the variable we are transforming)

Piecewise-Linear Transformations (2 Pieces)

$$f(t) = \begin{cases} q \frac{t-a}{b-a} & , \quad a \leq t < b, \\ q + (1-q) \frac{t-b}{c-b}, & b \leq t \leq c. \end{cases}, \quad (2.6)$$

Piecewise-Linear Transformations (3 Pieces)

$$f(t) = \begin{cases} q \frac{t-a}{b-a} & , \quad a \leq t < b, \\ q + (r-q) \frac{t-b}{c-b}, & b \leq t < c, \\ r + (1-r) \frac{t-c}{d-c}, & c \leq t \leq d. \end{cases} \quad (2.7)$$

Dual Aggregation Function

$$A^d(\mathbf{x}) = 1 - A(1 - x_1, 1 - x_2, \dots, 1 - x_n) \quad (2.8)$$

The Power Mean

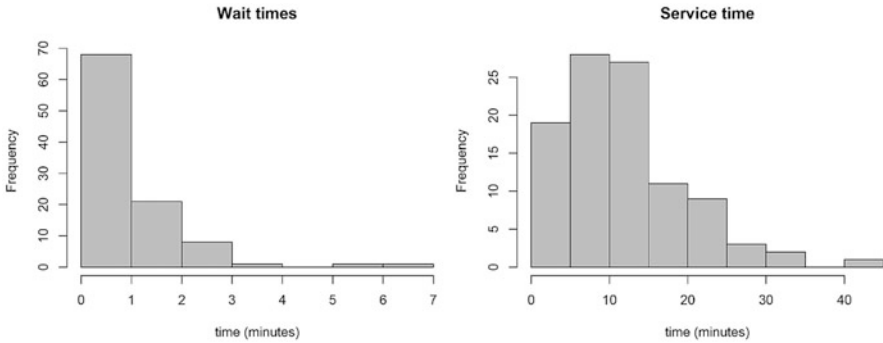
$$\text{PM}_p(\mathbf{x}) = \left(\frac{1}{n} \sum_{i=1}^n x_i^p \right)^{\frac{1}{p}} = \left(\frac{x_1^p + x_2^p + \dots + x_n^p}{n} \right)^{\frac{1}{p}} \quad (2.9)$$

The Quasi-Arithmetic Mean

$$\text{QAM}_g(\mathbf{x}) = g^{-1} \left(\frac{1}{n} \sum_{i=1}^n g(x_i) \right) \quad (2.10)$$

2.11 Practice Questions

1. Consider the following wait and service times for 100 customers.

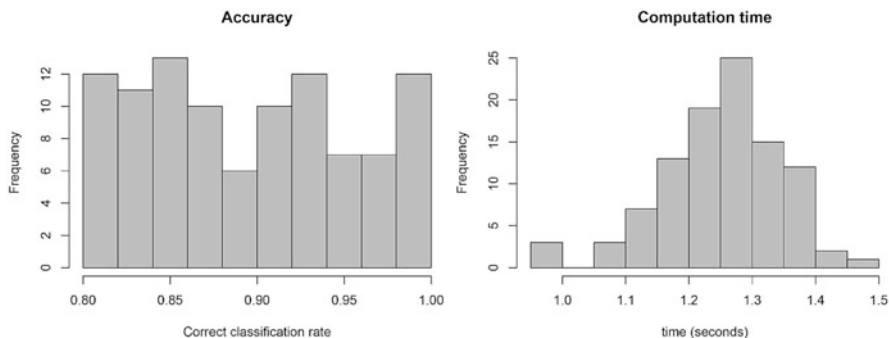


What would be some potential transformations that could be used to scale these variables to the unit interval?

2. Below are histograms of the accuracy and computation time taken for varying classification parameters. We want to be able to aggregate the two scores in order to be able to determine the best overall classifier. Suggest some transformations that would be able to transform the variables so that it makes sense to take their average.
3. For the following piecewise function,

$$f(t) = \begin{cases} \frac{t}{5}, & t < 3, \\ \frac{3}{5} + 2\frac{t-3}{10}, & 3 \leq t \leq 5, \end{cases}$$

what will be the value when $t = 2$? How about when $t = 4$?



4. Is the power mean symmetric, homogeneous and translation invariant? Explain.
5. Does the power mean have absorbent elements? Explain.
6. Write out the power mean explicitly for 3 arguments when $p = 2$.
7. Write out the power mean explicitly for 2 arguments when $p = -4$.
8. If $\text{PM}_p(9, 10, 17, 16) = 13$ for some value of p , can we work out the value of $\text{PM}_p(12, 13, 20, 19)$ and $\text{PM}_p(18, 20, 34, 32)$ without knowing p ?
9. If $p = -4$, can we determine the value of $\text{PM}_p(3, 0, 2, 8)$ without calculating?
10. Write out the explicit formula (i.e. without the \sum symbol) for $\text{PM}_p(x_1 + 3, x_2 + 3)$.
11. If $\text{PM}_p(3, 7, 29, 45) = 7.162$, is the value of p likely to be greater than 1 or less than 1?
12. If $\text{PM}_p(3, 7, 29, 45) = 36.258$, is the value of p likely to be greater than 1 or less than 1?
13. If $\text{PM}_p(2, 3, 8, 3) = 7$, is the value of p likely to be high or low?

2.12 R Tutorial

Here we will start working with whole matrices of data and in R [8] learn how to implement transformations and power means.

2.12.1 Replacing Values

We learnt how to assign vectors and values in the previous chapter. If we want to change the value of just one entry in a vector, we can use the assign command and indicate the index in square brackets, e.g. to change the 5th entry in a vector `a` we would enter

```
a[5] <- 10
```

We can also replace multiple entries at once

```
a[c(5,7)] <- c(10,3)
a[3:5] <- c(1,0,1)
```

R Exercise 6 *Create the vector and change the entries.*

```
a <- rep(0,20)
a[5] <- 1
a[c(3,7,11)] <- c(2,6,1)
a[17:20] <- c(1,2,1,4)
```

Your expected output when you type a and press enter should now be

```
0 0 2 0 1 0 6 0 0 0 1 0 0 0 0 0 1 2 1 4
```

2.12.2 Arrays and Matrices

Sometimes we will need to consider whole datasets at a time. For this, rather than just vectors, we will need rows of vectors, or matrices and arrays.

We can build them step by step with the `cbind()` (column bind) and `rbind()` (row bind) functions.

```
a <- cbind(c(2,3,5,1,0), c(6,1,8,2,9))
b <- rbind(c(4,1), c(2,-2), c(5,6))
```

R Exercise 7 *Assign the vectors and perform the `cbind()` and `rbind()` operations.*

```
a <- c(1,2,3,7,9)
a <- cbind(a, c(21,2,1,5,6))
a <- cbind(a, c(2,-1,5,0,-1))
a <- cbind(a, c(1,9,7,2,1), array(6,5))

b <- c(3,6,1,92)
b <- rbind(b, c(3,2,1,8,9))
b <- rbind(b, c(4,1,12,1,2))
```

Note that for vectors like `c(3,2,1,8,9)`, R doesn't treat it as either a row or a column when it's by itself and so it is flexible when it comes to combining vectors either as rows or as columns. However once we have a matrix, the `rbind` and `cbind` operations will come up with a warning if the row length is not the same—however it will still merge them (it just fills the remaining space by repeating the sequence of the row).

We can also create large $m \times n$ arrays and then input values later. We will use the `array` function for this.

Side Note 2.5 *Matrices can also be constructed using `matrix()`, however the array function will be the most straightforward to use at this time.*

As with the array function previously, the first entry is the default value that will populate the array, however this time we will use a vector `c(rows, columns)` to indicate how many rows and how many columns there needs to be. A matrix/array with 3 rows and 4 columns (prepopulated with 0s) would be

```
A <- array(0, c(3, 4))
```

Side Note 2.6 *We also can have higher dimension arrays, however for the moment we will stick to two.*

Once we've created our array with default values, we can then proceed to fill it. We refer to the cells using `A[row, column]` so that the entry in the first row and the second column would be `A[1, 2]`. We can also consider entire columns, e.g. the second using `A[, 2]` and entire rows using `A[1,]`. Note that this corresponds with our x_i , x_j and x_{ij} notation. Using `A[, 2]`, will treat the column as a normal vector, however we can also use `A[, 2, drop=FALSE]`, which maintains its column structure.

R Exercise 8 *Create a 3×4 array and then assign values to different entries using the following.*

```
A[3, 1] <- 4
A[1, ] <- c(1, 2, 3, 4)
A[, 2] <- c(6, 5, 4)
A[3:4, 2:3] <- array(-1, c(2, 2))
```

Your final matrix should appear as

	[, 1]	[, 2]	[, 3]	[, 4]
[1,]	1	6	3	4
[2,]	0	5	-1	-1
[3,]	4	4	-1	-1

2.12.3 Reading a Table

Often we will have data available from some other source, e.g. as an Excel spreadsheet. We can import this data using the `read.table()` function. The easiest way is to have the data in a txt or csv file. We need to know whether the entries are separated by commas or spaces and whether they have labels. In the simplest case,

we can save the table to an array. If the file is in the same folder as our R working directory,⁶ we can use the command

```
A <- read.table("thedata.csv")
```

If the data has headers (i.e. the first row is not entries but data labels), or is separated by commas, then we can add extra options.

```
A <- read.table("thedata.csv", header=TRUE, sep=",")
```

It is easiest if our data is already numerical data. In some cases, the data can be coded as text even if it is numbers. If columns in the data are numeric and R has trouble interpreting them as numbers (e.g. it says NA when you try to add 2, where NA indicates not available, i.e. a missing value), they can be extracted from the table using the `as.numeric()` command. The following exercise uses the `write.table()` command as well.

R Exercise 9 Use the following to create and write a table to the working directory

```
write.table(cbind(c("a",1,2),c(3,2,5)), "wordy.txt")
```

Now read the table from the file and assign it to `my.table` using the `read.table()` command.

```
my.table <- read.table("wordy.txt")
```

See what happens when you input the following.

Input	Expected output
<code>my.table</code>	<pre>V1 V2 1 a 3 2 1 2 3 2 5</pre>
<code>my.table+2</code>	<pre>V1 V2 1 NA 5 2 NA 4 3 NA 7</pre>
<code>my.table[2,]+2</code>	<pre>V1 V2 2 NA 4</pre>
<code>as.numeric(my.table[2,])+2</code>	<pre>3 4</pre>

⁶In RStudio, the working directory can be set using the Session→Set Working Directory option from the menu. In the standard R application on a Mac, the option to change the working directory is under Misc, while in Windows it is under File. You can also use the `setwd()` function directly from the console.

2.12.4 Transforming Variables

Simple transformations of variables can be achieved using operations that we have already learned. We can either choose to simply replace our values with the transformed ones or we can create a new matrix.

Let's first load our volleyball data⁷ and then make a copy which we will store as "original".

```
V <- read.table("volley.txt")
original <- V
```

The second of these lines simply copies the table "V" and assigns the copy to `original`. The following command replaces the *Sprint* variable with transformed values according to our negation function.

```
V[,1] <- 51.24 - V[,1]
```

We've used `V[,1]` to access the whole first column and when we replace it with `51.24 - V[,1]`, each entry in the column is subtracted from 51.24. We can then transform it to the unit interval with the following

```
V[,1] <- (V[,1] - min(V[,1])) / (max(V[,1]) - min(V[,1]))
```

Be careful with brackets and make sure your operation is working correctly. Sometimes it's good to manually check the first few values to be sure. We can use `head(V)` to view the first few entries of the table.

Now let's transform the height variable using standardization. For this, we will use the `sd()` function to calculate the standard deviation.

```
V[,2] <- (V[,2] - mean(V[,2])) / sd(V[,2])
```

R Exercise 10 Use the linear feature scaling technique to get this column and the remaining columns, 3 and 4, to range between 0 and 1.

2.12.5 Rank-Based Scores

There are three functions in R that can help us with situations where we are interested in the order of arguments in a vector. These are `sort()`, `order()` and `rank()`.

⁷All data files can be found at http://www.researchgate.net/publication/306099814_AggWAFit_R_library or alternatively, <http://aggregationfunctions.wordpress.com/book>. These can be saved to your R working directory.

The `sort()` function re-orders a vector into increasing (or non-decreasing) order. So with the vector `(1, 6, 2, 3)`,

```
sort(c(1, 6, 2, 3))
```

would have an output of `1 2 3 6`.

Order, on the other hand, tells us the ordering permutation (the indices from highest to lowest).

```
order(c(1, 6, 2, 3))
```

would have the output `1 3 4 2` because the ordering is $x_1 < x_3 < x_4 < x_2$. If there are ties then the `order()` function will sort them according to the index, i.e. `(3, 2, 3)` would be sorted `2 1 3` and not `2 3 1`. With both of these functions, we can change to descending order by adding the additional argument `decreasing = TRUE`, i.e.

```
order(c(1, 6, 2, 3), decreasing = TRUE)
```

will produce the output `2 4 3 1`.

Rank tells us the relative ranking of the variables. So

```
rank(c(1, 6, 2, 3))
```

will have an output of `1 4 2 3` because the 6 is ranked fourth, the 2 is ranked second etc. The decreasing option is not available for `rank()`, however by using a negative in front of the input vector the opposite ranking will be obtained, i.e.

```
rank(-c(1, 6, 2, 3))
```

would be `4 1 3 2`.

The most useful function for us in order to convert our *Sprint* times to rank-scores would hence be the `rank()` function. The following ranks the times and then scales them to the unit interval (the first line retrieves the original times).

```
V[,1] <- original[,1]
V[,1] <- (rank(-V[,1]) - 1) / (length(V[,1]) - 1)
```

In this case we used `rank(-V[,1])` because the lowest value would usually be given the rank 1 but we want it to have the highest score. The subtractions of 1 will mean that the worst time will be given a score of zero.

2.12.6 Using `if()` for Cases

Using the `if()` expression requires a careful consideration of the sequence and logic of our function. Let's first consider an example of a piecewise-linear function for values between 0 and 1 that has its join at $(0.5, 0.7)$, i.e. if the input is 0.5, then the output is 0.7. If the input is below 0.5, then it gets increased at the same ratio,

and if it is above 0.5, then the ratio of increase drops off so that it still has the output of 1 if the input is 1.

Using Eq. (2.6), we express the function as an equation in the following way

$$f(t) = \begin{cases} 0.7 \frac{t}{0.5}, & 0 \leq t < 0.5 \\ 0.7 + 0.3 \frac{t-0.5}{0.5}, & 0.5 \leq t \leq 1. \end{cases}$$

As a function in R, we need to create a clause that transforms it using the first equation if the value is less than 0.5, and using the second equation if it is above 0.5. There are a few different ways to do this. The easiest is to use `if (...) { ... } else { ... }`. Inside the `if ()` brackets, we have something like `t < 0.5` or `t >= 0.5` (the latter means greater than or equal to 0.5). It is a logical condition that must evaluate to a single TRUE or FALSE (not NA, and not multiple values). In the first case brackets `{...}`, we tell the function what to do if the `if ()` statement is true, and the second case brackets tells the function what to do otherwise. The following programs our piecewise function above.

```
pw.function <- function(t) {
  if(t < 0.5) {0.7*t/0.5} else {0.7+0.3*(t-0.5)/0.5}
}
```

R Exercise 11 Enter in the function and calculate the outputs for a few values, e.g. `pw.function(0.1)`, `pw.function(0.6)`, to see that it makes sense and is working correctly.

We can repeat this process in nested form to define more cases. In the following, we interpolate the points (0.5, 0.7) and (0.8, 0.9). In this case our intervals in terms of the input cases are [0, 0.5], [0.5, 0.8] and [0.8, 1] respectively.

```
pw.function.2 <- function(t) {
  if(t < 0.5) {0.7*t/0.5}
  else {if(t < 0.8) {0.7+0.2*(t-0.5)/0.3}
        else {0.9+0.1*(t-0.8)/0.2} }
}
```

Another way to have a look at our variables and make sure our functions are working correctly is to plot them.

2.12.7 Plotting in Two Variables

If we just want to view a variable to get an idea of the distribution, we can input the vector containing that variable's data. So to plot our original Sprint data.

```
plot(original[,1])
```

This just plots the value in sequence, so along the x-axis is the index of the datum and the y-axis contains its value. It can be easier to see the distribution by first sorting the data. So we can enter

```
plot(sort(original[,1]))
```

Of course, usually histograms are used to plot distributions. So we can also use the following to get a snapshot of our data.

```
hist(original[,1])
```

If we want to plot a single variate function, plotting is reasonably straightforward, however we usually need to create our 'x' values and 'y' values beforehand. To create equispaced x values, the easiest way is to use the `seq()` command.

So if we want to create a vector of 100 points over the unit interval (starting with 0.01 and going up by 0.01 each entry), we can write `seq(0.01, 1, length.out=100)`. If we want to start at 0 (which would give us 101 points), then we can write `seq(0, 1, length.out=101)`.

To plot the function $f(t) = t^2$ for these values, we can either use

```
x <- seq(0,1,length.out=101)
plot(x^2)
```

or if we want the x labels to correspond with the data in that variable, we would include these as the first argument and the transformed values as the second argument.

```
x <- seq(0,1,length.out=101)
plot(x,x^2)
```

However, if we want to plot a function like our piecewise function, we need to create the y values separately first. This is because in our cases we used an if-statement, `if(t < 0.5)`, so when we try to input `pw.function(x)` it will ask whether the vector is less than 0.5, which is not a valid operation. To create the vector of y values, we will also now need to use a repeating operation `for()`.

The `for()` statement can perform an operation for every entry of a set. This is very useful when we can index our numbers.

The following sequence of operations first creates a vector of y-values (pre-filled with zeros). It uses `length(x)` so that it will be the same length as our x vector, but we could also just write 100. It then says for all the numbers in 1 to 100 (using the `1:100` vector), each entry in the y vector will be replaced by the piecewise function of the *corresponding* entry in the x vector. The `#` can be used in R for notes. Anything on the line that comes after the `#` is "commented out".

```
y <- array(0,length(x))      # 1. create a vector of zeros
for(i in 1:length(x)) {     # 2. perform this operation
  y[i] <- pw.function(xs[i]) #   changing 'i' for the numbers
}                             #   between 1 to 100.
```

We now should be able to plot this piecewise function.

```
plot(x,y)
```

2.12.8 Defining Power Means

We now have all the tools necessary to define our power means. This time we will have two inputs to the function, ‘x’, which will be a vector of inputs and ‘p’, which will be the power used.

```
PM <- function(x,p) {      # 1. pre-defining the function inputs
  (mean(x^p))^(1/p)        # 2. our calculation which will also
}                           #    be the output
```

However this function will not work if $p = 0$. So we need to create a special case. For this, we will use the `if()` command.

```
PM <- function(x,p) {      # 1. pre-defining the function inputs
  if(p == 0) {             # 2. condition for 'if' statement
    prod(x)^(1/length(x)) # 3. what to do when (p==0) is TRUE
  }
  else {
    (mean(x^p))^(1/p)      # 4. what to do when (p==0) is FALSE
  }
}
```

Note here that for ‘=’ conditions, we use a double equals sign ‘==’. So the possible conditions we can use inside the `if()` brackets are ‘==’, ‘<’, ‘>’, ‘<=’ and ‘>=’.

R Exercise 12 Define the power mean as a function in R and check the following.

Input	Expected output
PM(c(3,2,7),2)	4.546061
PM(c(1,0,7),0)	0
PM(c(0.28,0.4,0.47),-557)	0.2805528
PM(c(0.28,0.4,0.47),-558)	Inf

Note that in the last case, ‘Inf’ should not actually be the result. It’s just that at this point, the system cannot tell the difference between one of the transformed inputs and infinity, i.e. if you compare $0.28^{(-557)}$ and $0.28^{(-558)}$, the latter will be ‘Inf’ and then all of the operations become absorbed by this value (see [7]). It can be fixed by including a statement in the function definition that if $p < -500$ then the output should be the minimum.

2.13 Practice Questions Using R

1. Suppose you have $\mathbf{x} = \langle 0.3, 0.8, 0.1, 0 \rangle$, Calculate the power mean for the following cases
 - (i) $p = 4$
 - (ii) $p = 2.5$
 - (iii) $p = 0$
 - (iv) $p = -3.1$
 and comment on (i.e. compare) the results.
2. Create a 2-variate function for the power mean when $p = 3$ using


```
PM3 <- function(x,y) {...}
```

 i.e. so that it takes the inputs \mathbf{x} and \mathbf{y} , which will be numbers rather than vectors.
3. Load the data file “wait.service.txt” which has the wait and service times from Sect. 2.11 Question 1, and perform the following.
 - (i) Use appropriate scaling techniques so that the two variables both have data given over the same range.
 - (ii) Calculate the output of the power mean for $p = -1, p = 0, p = 1$ and $p = 2$.
 - (iii) What is the value of service and wait time that has the best aggregated value using each of the values of p ?
 - (iv) Plot the outputs for each of the functions and compare the results.
4. Load the two data files “comp.acc.txt” and “comp.time.txt” and, either merging them into a single table or keeping them as separate vectors, perform the following.
 - (i) Use appropriate scaling techniques so that the two variables both have data given over the same range.
 - (ii) Calculate the output of the power mean for $p = -1, p = 0, p = 1$ and $p = 2$.
 - (iii) What is the value of accuracy and time that has the best aggregated value using each of the values of p ?
 - (iv) Plot the outputs for each of the functions and compare the results.

References

1. Beliakov, G., James, S.: Citation based journal ranks: the use of fuzzy measures. *Fuzzy Sets Syst.* **167**(1), 101–119 (2011)
2. Beliakov, G., Pradera, A., Calvo, T.: *Aggregation Functions: A Guide for Practitioners*. Springer, Heidelberg (2007)
3. Beliakov, G., Bustince, H., Calvo, T.: *A Practical Guide to Averaging Functions*. Springer, Berlin/New York (2015)

4. De Veaux, R.D., Velleman, P.F., Bock, D.E.: Stats: Data and Models. Pearson, Essex (2016)
5. Gagolewski, M.: Data Fusion. Theory, Methods and Applications. Institute of Computer Science, Polish Academy of Sciences, Warsaw (2015)
6. Grabisch, M., Marichal, J.-L., Mesiar, R., Pap, E.: Aggregation Functions. Cambridge University press, Cambridge (2009)
7. Lesh, R., Cramer, K., Doerr, H.M., Post, T., Zawojewski, J.S.: Model development sequences. In: Lesh, R.A., Doerr, H.M. (eds.) Beyond Constructivism: Models and Modeling Perspectives on Mathematics Problem Solving, Learning, and Teaching, pp. 35–58. Routledge, New York (2003)
8. R Core Team: R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna (2014) <http://www.R-project.org/>
9. Torra, V., Narukawa, Y.: Modeling Decisions. Information Fusion and Aggregation Operators. Springer, Berlin/Heidelberg (2007)

An Introduction to Data Analysis using Aggregation
Functions in R

James, S.

2016, X, 199 p. 29 illus., 20 illus. in color., Hardcover

ISBN: 978-3-319-46761-0