# Math381 Assignment 7: MDS

Jiahang Wu

November 2021

## 1 Introduction

When comparing different objects, people often need to include multiple factors to help them make decisions, such as length, weight, price, score, etc. However, as more and more factors are included, it is even harder to visualize the comparison in multiple dimensions, since human sights are limited in three-dimensional spaces. In order to solve this problem, we here introduce Multidimensional Scaling (MDS).

MDS helps people to create a low dimensional model of multidimensional objects. For example, when we want to compare different vehicles, we probably need to take size, engine, chassis and so on, but with MDS, we can incorporate all these factors and reflect the "distance", which represents how different these vehicles are overall, on a lower dimensional graph, such as a 2-D plot.

For better illustration on MDS, I will use a concrete example below to explicate the concept.

## 2 Dataset: Typewriters

In this project, we will compare different typewriter models and build MDS models with the help of R language, and we use the file 24 dataset from HARTIGAN Clustering Algorithm Datasets. and it can be accessed here.

We can take a brief look at the dataset:

```
# load the dataset
typewriters <- read.csv("https://sites.math.washington.edu/
                    ~conroy/m381-general/MDSdataSets/
                    hartigan-file24.csv",
                    header = FALSE)
# print the first few rows of the dataset
head(typewriters)
            V1    V2    V3   V4   V5    V6 V7 V8 ...
1      Olympia 10 15.75 15.00 7.75 21.0 19.38 44  1  ...
2      Olympia 13 16.00 19.00 8.00 25.5 13.25 44  1  ...
3 Smith Corona 12 14.50 17.75 6.00 19.5 11.88 44  1  ...
```

```
4        Sears 12 14.50 17.75 6.00 19.5 11.88 44  1  ...
5     Hermes 3000 14.25 13.25 6.00 17.0  9.75 44  1  ...
6       Wards 510 14.75 16.25 6.50 19.5  9.88 44  1  ...
```

There are 19 different typewriter models in the dataset, and there are 21 columns of factors describing these models. These factors can be categorized into three types:

1. First column: the name of the typewriter model
2. Second to sixth columns: numerical values describing the size and weight of the typewriter model
3. Seventh to last columns: binary values indicating whether the typewriter model has a specific function/key or not

Note that the seventh column is not binary originally binary, but since the values represent the number of keys of the typewriter model, and there are only two results of either 43 or 44, we can regard it as binary in this case.

Considering these different variable types, we will only use the second types of factors to plot multidimensional scaling, which are height, width and depth in inches, weight in pounds and the platen length in inches. It doesn't make sense to compare names of typewriter models, but we can include names to better visualize the MDS. However, for binary variables, although we can incorporate them into our MDS, they may result in a much more complicated situation, and we will discuss it later.

# 3 Preparations for Models

Before starting calculating distances, we want to make sure that all factors we take into account are comparable. Different dimensions use their own scales, and the difference between 100 and 200 is not a hundred times greater than the difference between 1 and 2 in MDS. Therefore, we need to "normalize" the dataset.

There are several ways to do so, and we can use multiple methods to find what effects they have on the calculated distance.

The first way is to normalize with the help of mean and standard deviation, and then we can calculate $\frac{\text{value}-\text{mean}}{\text{standard deviation}}$, so that no matter how large or small the values are, they are all scaled into a similar range of values. This is the same as calculating z-score in normal distribution, therefore the normalized values would be restricted to similar values, usually within $\pm 3$.

Another way of normalizing is to use the maximum and minimum of the columns. Similarly, we can scale the values to [0, 1] by applying $\frac{\text{value}-\text{min}}{\text{max}-\text{min}}$.

We can try both methods in R code:

```
# Mean and SD
normalized <- matrix(, nrow = numrow, ncol = 5)
```

```
for (i in 2:6) {
    column <- typewriters[,i]
    cmean <- mean(column)
    csd <- sd(column)
    normalized[,i-1] <- ((column - cmean) / csd)
}
# Max and Min
normalized2 <- matrix(, nrow = numrow, ncol = 5)
for (i in 1:5) {
  column <- typewriters[, i+1]
  cmax <- max(column)
  cmin <- min(column)
  normalized2[, i] <- (column - cmin) / (cmax - cmin)
}
```

With a normalized dataset, we are able to calculate the distances between these typewriter models. There are many ways to do so, and we will use Euclidean distance:

```
# For mean-sd method
distance <- as.matrix(dist(normalized, method = "minkowski", p=2))
# For max-min method
distance2 <- as.matrix(dist(normalized2, method = "minkowski", p =2))
```

In the code, we use the built in "dist" function to help us do the calculation, and we set the method to be "minkowski" here to use Euclidean distance, and set p = 2 so that it is classic Euclidean distance.

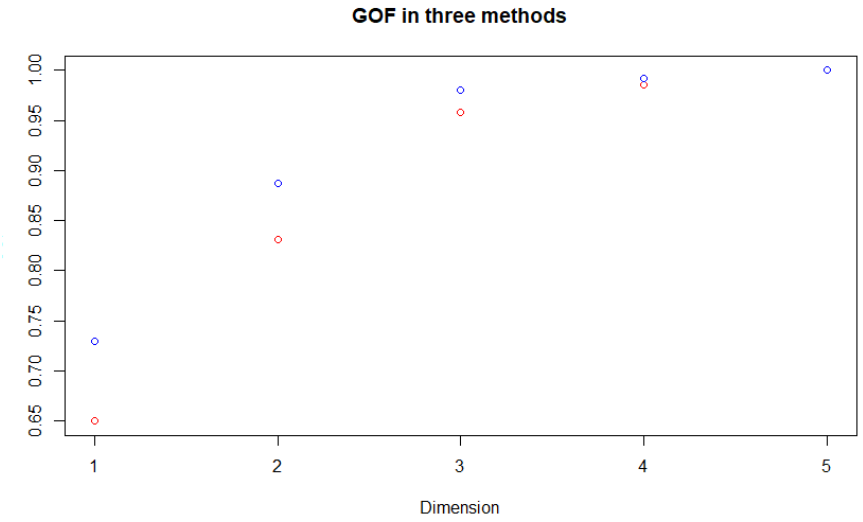
However, we don't know which of the normalization methods is the best to work with, and we are even not sure whether we should normalize it. Therefore, we need to compare these three datasets, which can be partly done by reviewing their goodness of fit (GOF). GOF is a value from 0 to 1, showing how good the model is. A completely random model has a GOF value around 0.16, so we should expect to see a much higher GOF to conclude that we have a good model.

GOF values can be obtained by calling "cmdscale" function in R, and this is also the function we are using to create models in different dimensions for MDS. Here we can use "cmdscale" function to draw a plot of goodness of fit (GOF) in these two normalization methods and the original dataset to find which one is the best to use:

```
GOFs = c() # GOF for mean-sd method model
for (i in 1:5) {
  GOFs[i] <- cmdscale(distance, k = i, eig = TRUE)$GOF[1]
}

GOFs2 = c() # GOF for max-min method model
```

```
for (i in 1:5) {
  GOFs2[i] <- cmdscale(distance2, k = i, eig = TRUE)$GOF[1]
}

GOFs_original = c() # GOF for the unaltered model
for (i in 1:5) {
  GOFs_original[i] <- cmdscale(dist(typewriters[,2:6]),
  k = i, eig = TRUE)$GOF[1]
}
print(GOFs)
print(GOFs2)
plot((1:5), GOFs2, main = "GOF in three methods",
xlab = "Dimension", ylab = "GOF")
points((1:5), GOFs2, col = "red")
points((1:5), GOFs_original, col = "blue")

Output:
0.6470321 0.8150131 0.9566370 0.9860096 1.0000000
0.6499218 0.8315716 0.9583112 0.9861701 1.0000000
```

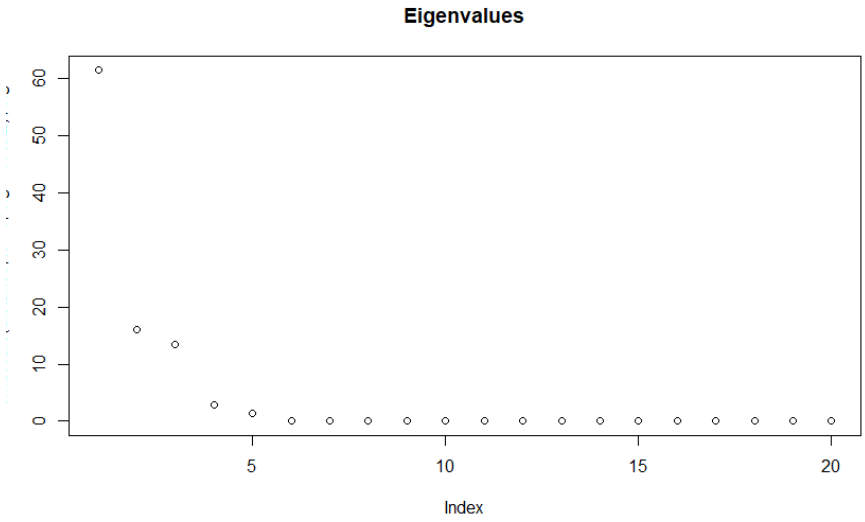
Note here that when we want to obtain GOF from cmdscale function, we usually receive a list of two values. With the help of "?cmdscale" command, we can look up how cmdscale function works, and we are able to know that these two values are calculated based on eigenvalues of the matrix in two ways, while one way we find the absolute value of the eigenvalue, and in the other way we take the largest one between the eigenvalue and 0. These two ways of calculation makes no difference when eigenvalues are positive, and fortunately but certainly, the eigenvalues of distance matrix calculated by Euclidean distance are always positive, so we can take either of the two GOF values for comparison.




The goodness of fit of the two normalized distances are approximately the same, while surprisingly we find that the unmodified dataset does the best job in fitting after MDS operation. This could indicate that we should use the original data and not normalize it. However, we should consider that different columns represent different measurements, and since we want to find distances by equally looking at all variables, we don't want to focus on any specific variable, and we want to get rid of the influence from different units. Therefore, we will still normalize the dataset so that each column contributes equal influence to the MDS model as others. Since the mean-sd method has similar GOF compared to the max-min method, we will stick with this normalization.

Another thing to do before building models is to decide the highest dimension we should probably build models, and we can find the necessary information in an eigenvalue graph:

```
plot(cmdscale(distance, k=1, eig=TRUE)$eig, main = "Eigenvalues")
```




We find that the first 5 entries are non-zero, which means that the our data fits up into 5-D models, so we don't need to consider models with dimensions higher than 5. Meanwhile, we can see that the first eigenvalue is much larger than the others, showing that the 1D model probably is not enough to describe the distance relationship well. The second and the third eigenvalues are relatively small, so we can 2D and 3D models will do a pretty good job. The last two eigenvalues are extremely small, so that we can expect to find a perfect model in 5D.

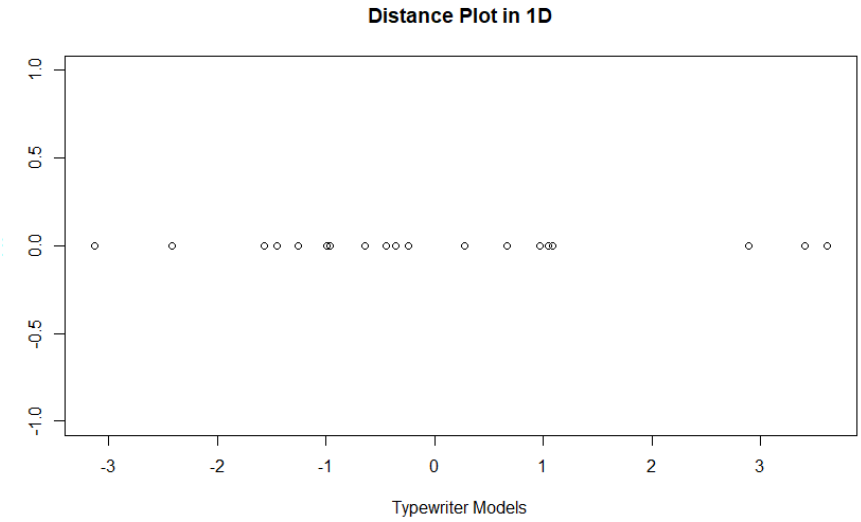
# 4 1D Model

After all the preparations, we are finally able to plot a MDS graph with the distance we have. As I have stated earlier, we will use "cmdscale", which stands for "classical multidimensional scaling", to generate necessary information for plots in different dimensions.

We will start with the simplest one, the 1D model, which displays distance relation on a linear line.

```
cmdscale(distance, k=1, eig=TRUE)
model1 <- cmdscale(distance, k=1, eig = TRUE)
for1Dplot <- data.frame(model1$points,0)
plot(for1Dplot, main = "Distance Plot in 1D",
xlab = "Typewriter Models")
```

We get the following plot:




The points in the graph represent each typewriter model, and we can generally check the distance between these points to find how different they are. However, if we calculate distances based on the 1D model and subtract it from the original distance matrix, we can calculate the mean absolute difference, which can help us understand better how different the model is from the original distance matrix:

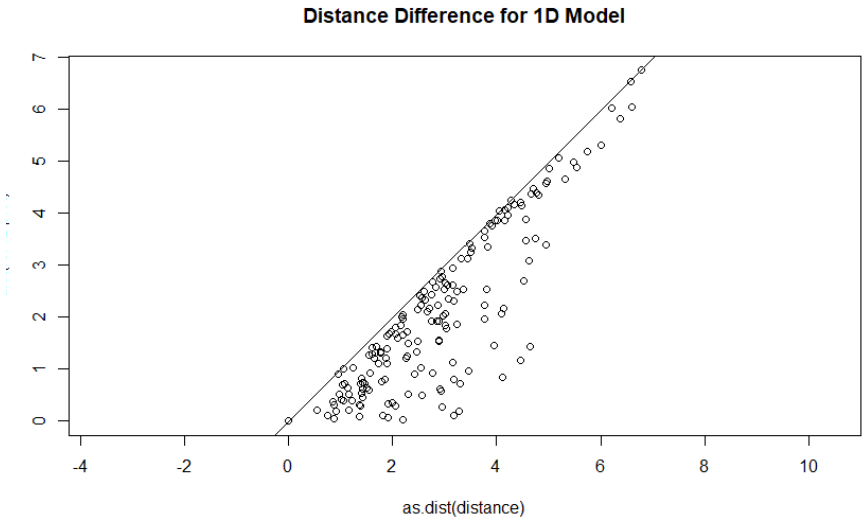
```
# the package for Mean Absolute Difference Calculation
library(DescTools)
MeanAD(distance - as.matrix(dist(for1Dplot)))
```

And we get:

```
[1] 0.5709129
```

This is a noticeable mean absolute difference, which means that we probably should not be satisfied with the 1D model. We can also use a visualization to better illustrate the difference:

```
plot(as.dist(distance),dist(for1Dplot),asp=1,
main = "Distance Difference for 1D Model")
abline(0,1)
```

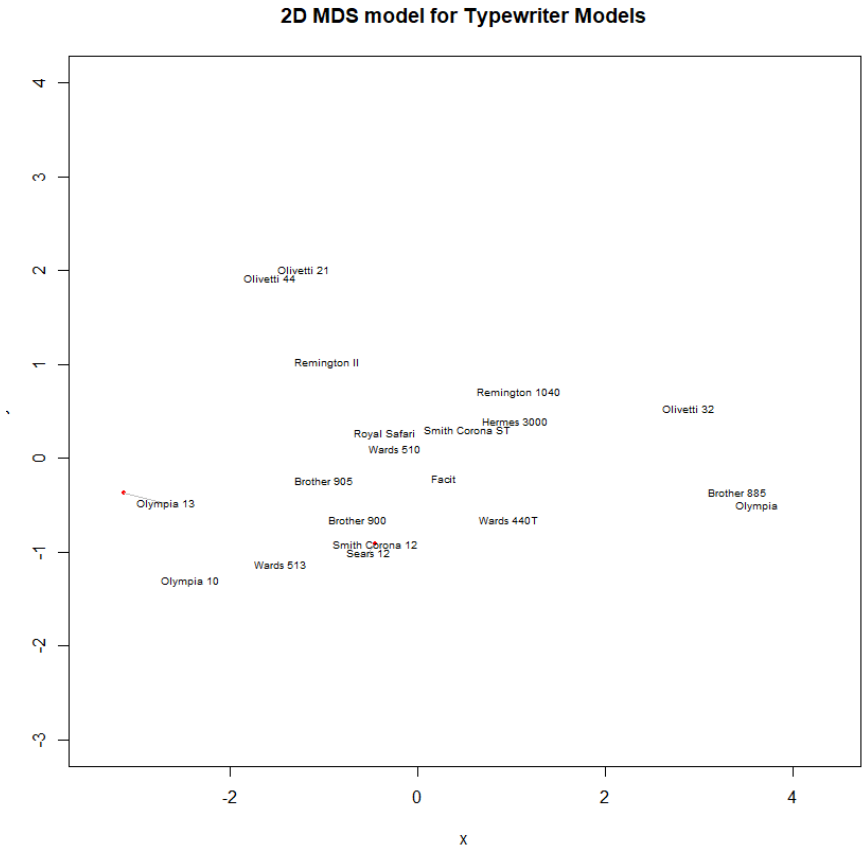



We can directly see from the graph that the difference between the distance in our 1D model and the input distance matrix is obvious, as many of the points don't fit onto the y=x line.

# 5 2D Model

Since 1D model is not good enough, we should proceed to 2D model with the similar code:

```
cmdscale(distance, k=2, eig=TRUE)
model2 <- cmdscale(distance, k=2, eig = TRUE)
for2Dplot <- data.frame(cmdscale(distance, k=2), 0)
textplot(model2$points[,1],model2$points[,2],
     typewriters[, 1],
     main = "2D MDS model for Typewriter Models",
     asp=1,xlim = c(-3, 4), ylim = c(-3, 4),
     cex=0.6)
```



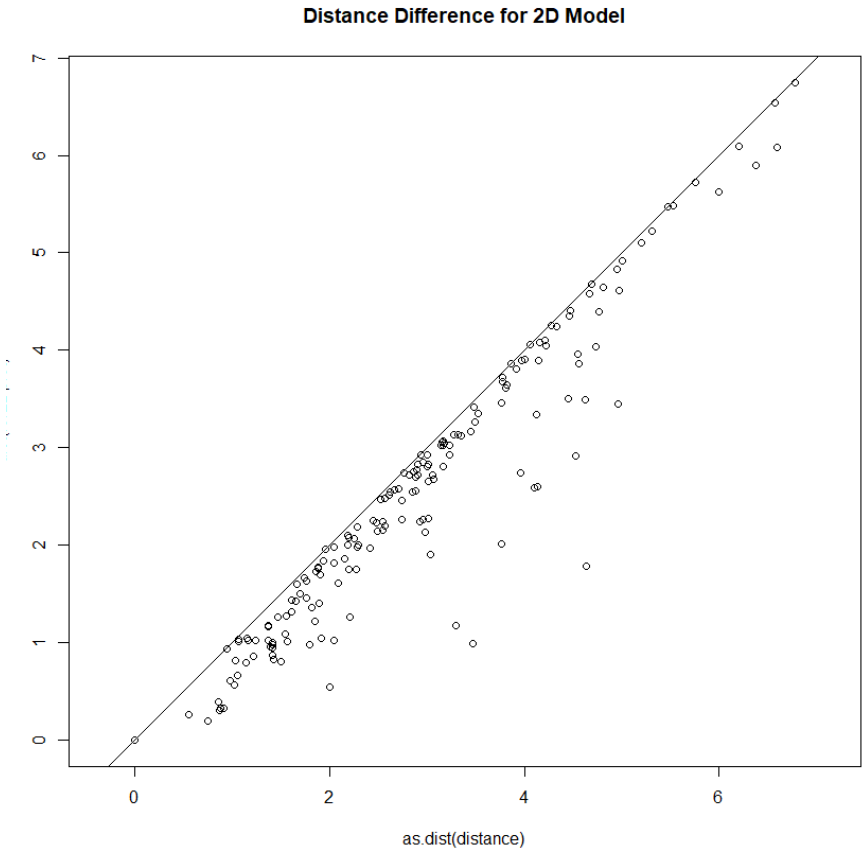

When constructing the 2D model, one thing different from the 1D model is that we need to keep track of the scale of x and y axes. Even though we have labels on both axes to help us perceive appropriate distance relationship between points in the graph, it would be confusing if 1 cm horizontally indicates a distance of 1, while 1 cm vertically indicates a distance of 100. Therefore, we want to manually set them to the same length, (-3, 4) in this case, so that the horizontal distance and vertical distance in the plot are perceived as the same when people look at the graph.

This time we can see clearer patterns in the graph, and we can again calculate the mean absolute difference:

```
MeanAD(distance - as.matrix(dist(for2Dplot)))
```

The output is 0.2839775, much smaller than the 1D model, and we can again visualize the difference to see how close we are:

```
plot(as.dist(distance),dist(for2Dplot),asp=1,
main = "Distance Difference for 2D Model")
abline(0,1)
```




The entries are not perfectly fit, but they are mostly on the y=x line, meaning that the model is quite accurate. We can also check the GOF value, which was already calculated in the previous part and equals 0.8150131, a high enough value to represent accuracy.

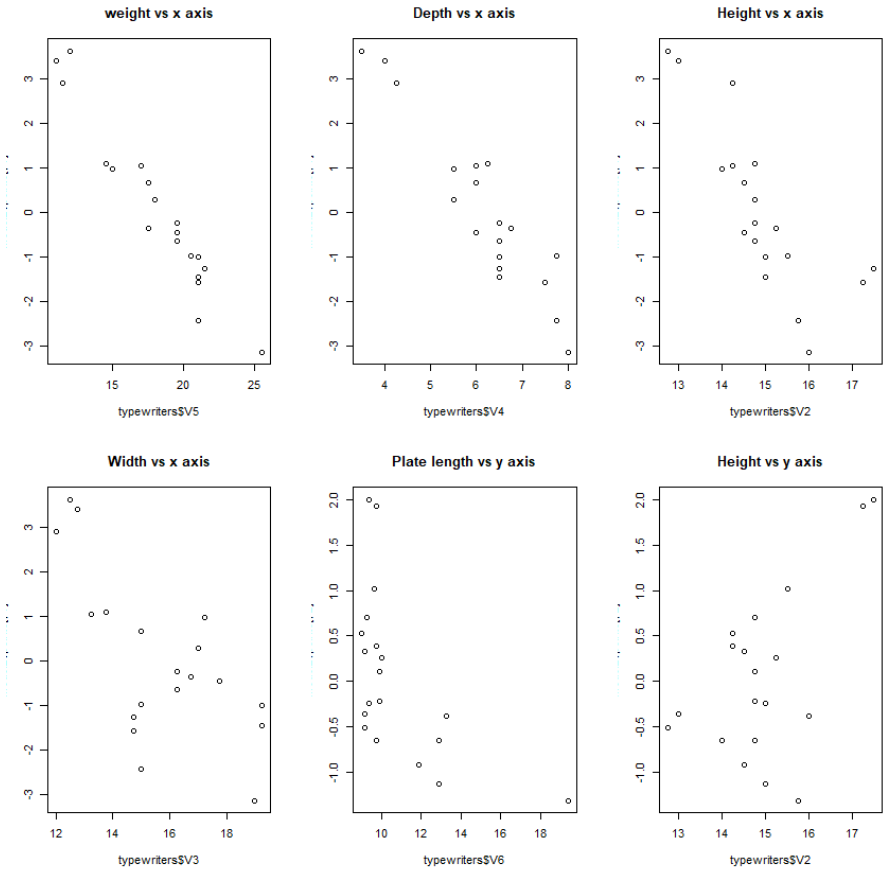
Since 2D model is the easiest one for people to obtain information from, we can analyze the situation and try to figure out what these axes mean in the graph.

We can look at the graph directly for patterns, such as finding that the right most models have lowest depths and widths, but we might not be able to find every factor influencing the position of these points of models. Therefore, we should try to find the relationship between 5 input dimensions and the x/y coordinates by calculating and graphing.

There are five variables we use to describe the typewriter models: length, width, depth, weight and platen length. We can use plot to visualize the relationship between them and the x or y coordinates in the 2D model:

```
attach(mtcars)
par(mfrow=c(2,3))
```

```
# Weight vs x axis, clear negative correlation
plot(typewriters$V5, model2$points[, 1],
     main = "weight vs x axis")
# depth vs x axis, strong negative correlation
plot(typewriters$V4, model2$points[, 1],
     main = "Depth vs x axis")
# height vs x axis, negative correlation
plot(typewriters$V2, model2$points[, 1],
     main = "Height vs x axis")
# width vs x axis, negative correlation
plot(typewriters$V3, model2$points[, 1],
     main = "Width vs x axis")
# Plate length vs y axis, slight and vague negative correlation
plot(typewriters$V6, model2$points[, 2],
     main = "Plate length vs y axis")
# height vs y axis, slight positive correlation
plot(typewriters$V2, model2$points[, 2],
     main = "Height vs y axis")
```




Excluding the completely random plots, we have six plots left, which we can find some correlation in. After reviewing these plots, we are able to find relative strong correlation between weight, height, depth, width and the x-coordinates. We can support this by finding that "Olympia" and "Brother 885", two typewriter models with the smallest height, almost the smallest width and the smallest depth, group up at the right end of the graph, while "Olympia 10", "Olympia 13", "Olivetti 21" and "Olivetti 44", as they all have large weight, widths and depths, are to the left side of the plot.

For y axis, the relationship is weaker, but we can still find correlation between y coordinates and height and platen length. "Olivetti 21" and "Olivetti 44" models have the largest heights, so they are on the top part of the graph, while "Olympia 10" has the longest platen length and is located at the bottom of the graph.

Therefore, we can generally find that models with large weight, height, depth and width tend to appear at the left side of the 2D plot. In other words, x axis in the 2D plot represents the size and the weights of different typewriter models.

Meanwhile, the models with larger heights are also located at the upper part

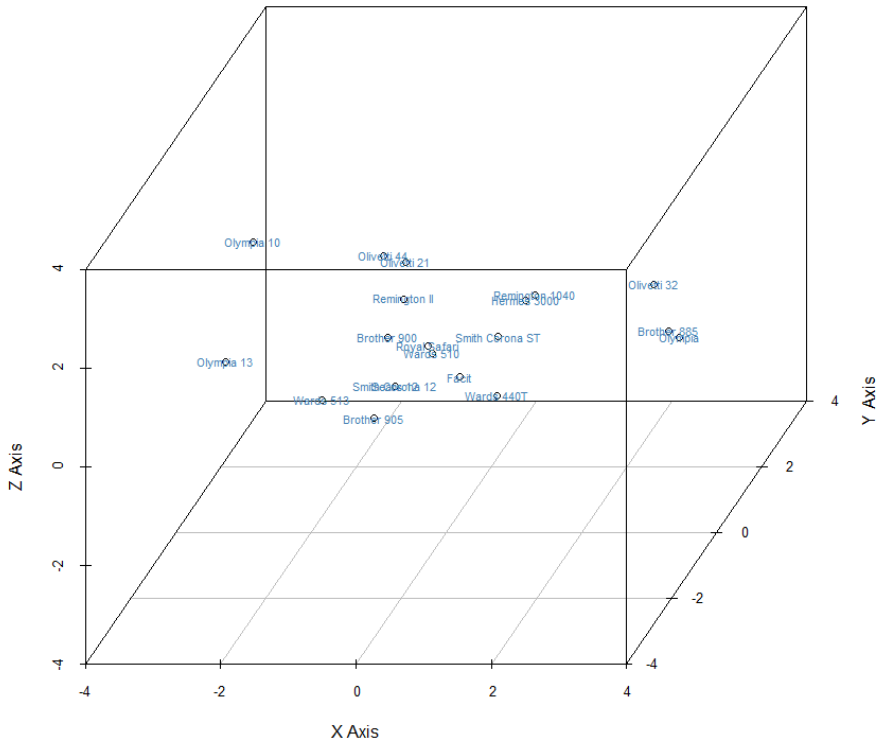
of the plot, while the ones with larger platen lengths are put to the bottom of the 2D plot. It is hard to find a concrete meaning for y axis, especially considering that the correlations we find are not strong, but it is understandable, since we cannot always expect to find a clear meaning for axes in MDS.

# 6 3D Model

We can continue to add dimension to our model to find a 3D model. However, since 3D plotting is different, we need to use "scatterplot3d" package to help us plot it:

```
library(scatterplot3d)
model3 <- cmdscale(distance, k=3, eig=TRUE)
s3d <- scatterplot3d(model3$points[, 1:3], angle = 60,
                 xlim = c(-4, 4), ylim = c(-4, 4), zlim = c(-4, 4),
                 xlab = "X Axis", ylab = "Y Axis", zlab = "Z Axis"))
text(s3d$xyz.convert(model3$points[, 1:3]), labels = typewriters$V1,
cex= 0.7, col = "steelblue")
```

The 3D graph looks like:

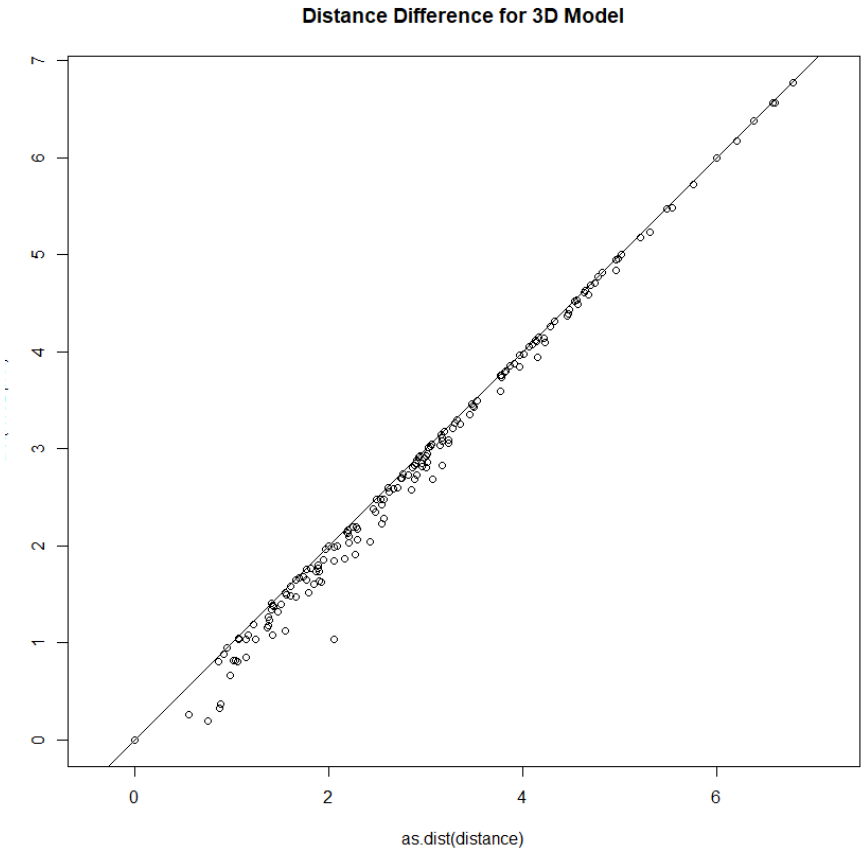



Compared to the 2D graph, this one is much more difficult for people to directly extract information from, since it requires people to imagine a 3D relationship. We again use mean absolute difference to find whether this is a good model:

```
MeanAD(distance - as.matrix(dist(for3Dplot)))
```

The result is 0.08601103, 4 times less than the 2D model, and we can plot the visualized difference compared to the input distance matrix:

```
plot(as.dist(distance), dist(for3Dplot), asp=1)
abline(0,1)
```




This time we can find that the vast majority of entries fall onto the y=x line, indicating that the model is very close to the input distance matrix in almost all entries.

Also, according to the GOF value we calculated before, we know that a 3D model's GOF is 0.957, almost reaching its maximum 1. Again, this indicates that a 3D model is a very accurate model to the original distance relationship.

# 7 4D and 5D Models

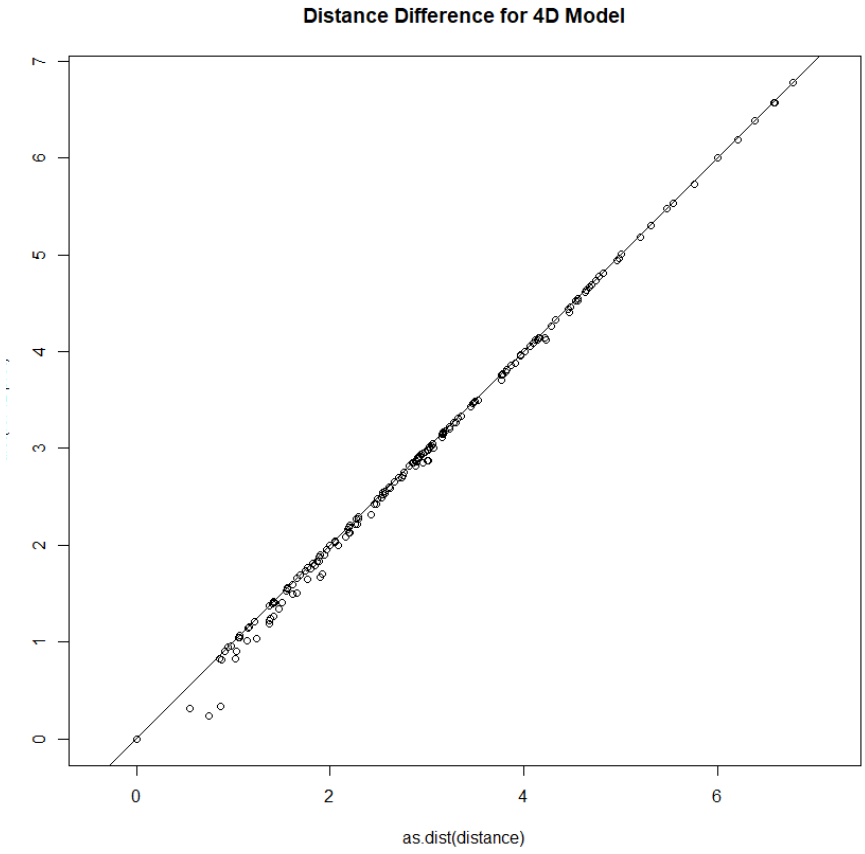
Even though we are able to either plot or imagine a 4D or 5D model, we can still calculate them with cmdscale:

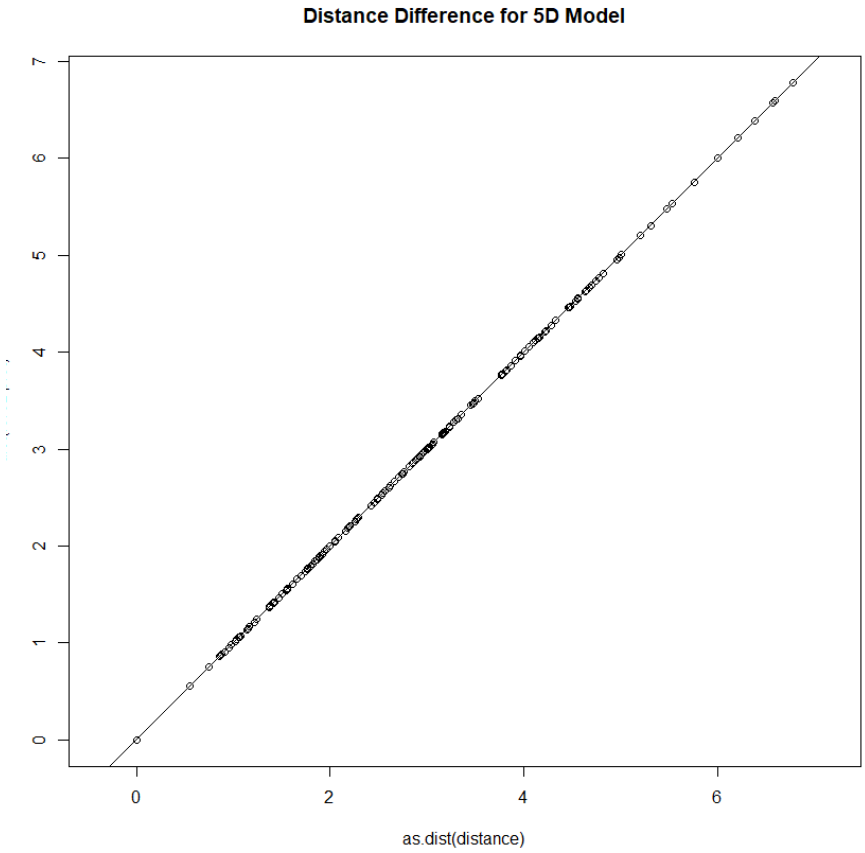
```
model4 <- cmdscale(distance, k=4, eig=TRUE)
model5 <- cmdscale(distance, k=5, eig=TRUE)
for4Dplot <- data.frame(model4$points, 0)
for5Dplot <- data.frame(model5$points, 0)
MeanAD(distance - as.matrix(dist(for4Dplot)))
```

```
MeanAD(distance - as.matrix(dist(for5Dplot)))
plot(as.dist(distance), dist(for4Dplot), asp=1,
     main = "Distance Difference for 4D Model")
abline(0,1)
plot(as.dist(distance), dist(for5Dplot), asp=1,
     main = "Distance Difference for 5D Model")
abline(0,1)
```

The mean absolute difference between 4D model and the input distance matrix is 0.04028656, and the mean absolute difference for 5D model is 1.666943e-15, nearly to be 0.

The corresponding plots of difference between models and the input distance matrix are shown below:

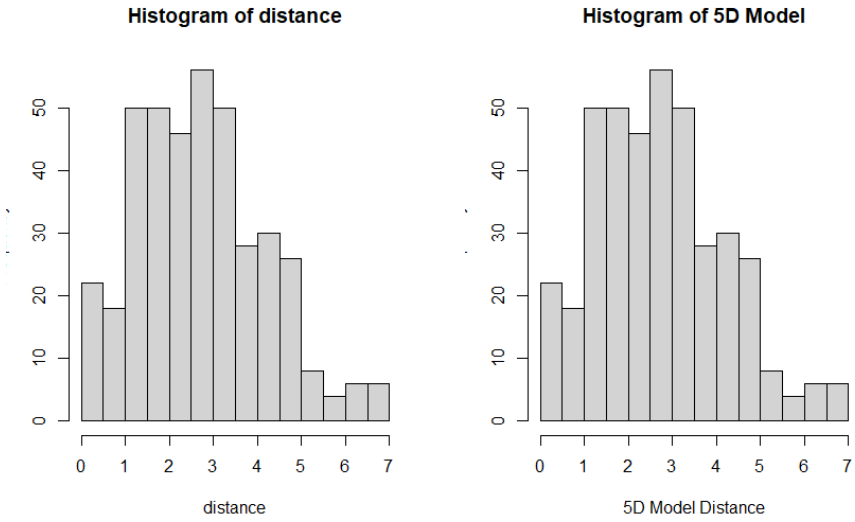





If we look back at the GOF values, we can find that these two models have GOF values as high as 0.986 and 1, which means that they, especially the 5D model, are almost perfect models for the input distance matrix. This is in accordance with the definition of MDS, as MDS is a lower dimensional model for high dimensional ones, and the 5D model shares the same dimension as the input.

We can compare histogram of the original distances and the histogram of the 5D model:

```
par(mfrow = c(1,2))
hist(distance)
hist(as.matrix(dist(model5$points)),
     main = "Histogram of 5D Model",
     xlab = "5D Model Distance")
```




The two histograms are identical, supporting that the 5D model is a perfect depiction of the input distance.

# 8 Conclusion

One important reflection in this project is about normalizing. In fact, since the dataset entries are in similar numerical ranges, and four of the five factors share the same unit of inches. However, normalizing is still important, as it can put all variables to the same level and prevent one variable from dominating the whole distance relationship.

Another idea comes to my mind about normalizing is that if we have a specific variable describing a property of the object, which contributes a lot to the object, we probably want to focus on the variable and want it to play a much more important role when calculating distance. In this case, we can even manually scale the variable to be larger and calculate the "weighted" distance, similar to weighted average.

I should also remember that the dist() function in R returns a semi-matrix instead of a full matrix, and it would render errors in calculation if we use it directly in comparisons. This is understandable, since the distance matrix is symmetric, and we need to put into matrix form by using as.matrix() to compare with the input distance matrix.

One question I have for my project is that the mean absolute difference between the 5D model entries and the input distance matrix entries should be 0 since the 5D model should perfectly describe the original input, but it is 1.666943e-15 instead of 0. I personally would regard this as internal mistake due to rounding in R calculation, but there might be other explanations to it.

# 9 Appendix

Code of all the contents above can be found in the R script through this Github link.