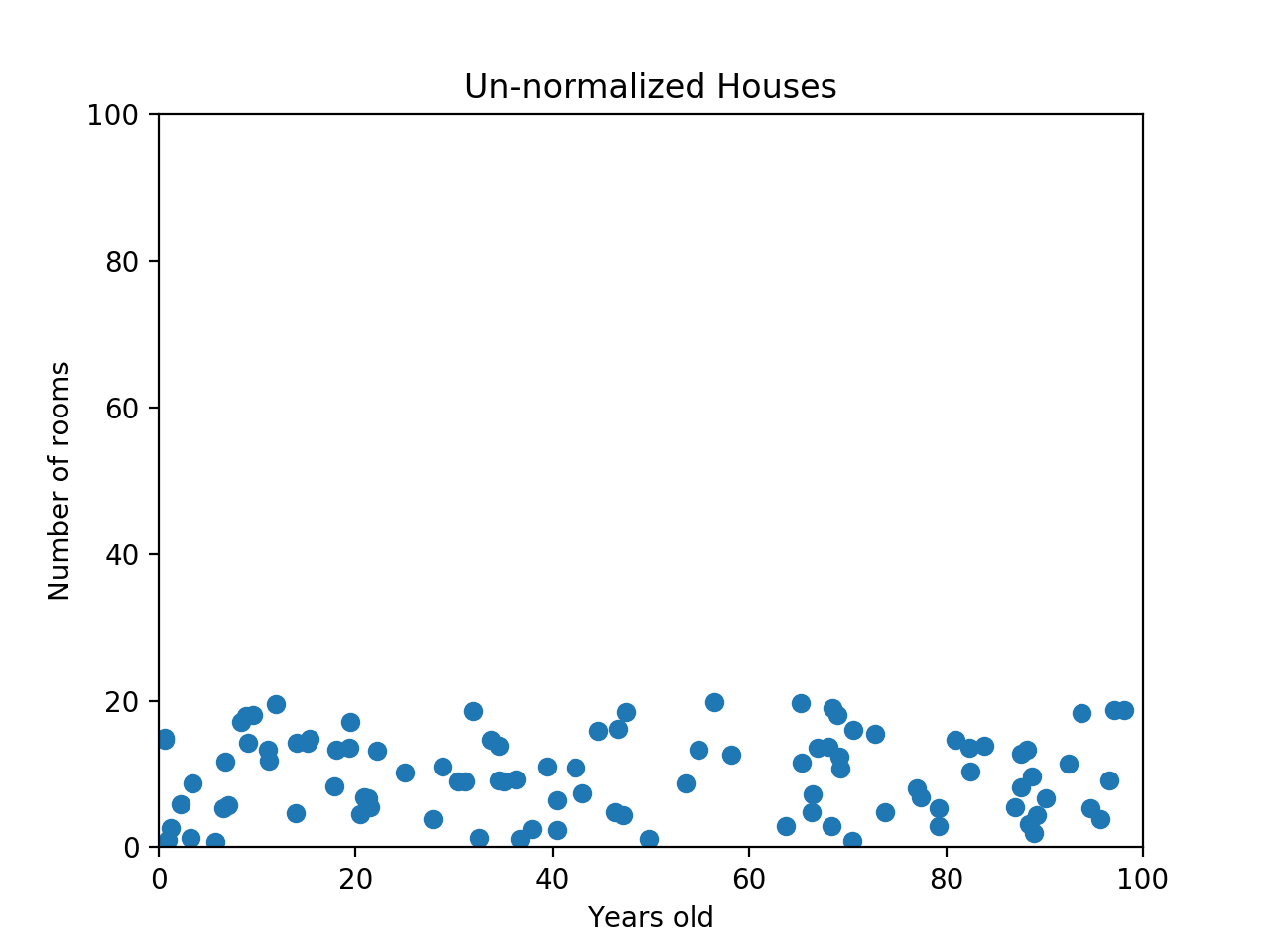
**R- Implementations**

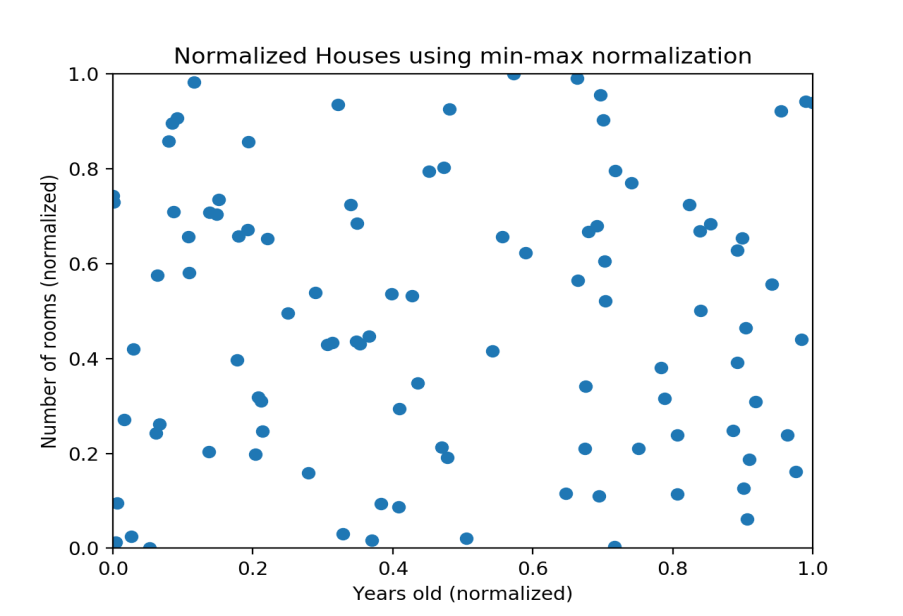
**Tutorial – Week 2: Pre-processing (normalisation, z-score, PCA)**

**Why Normalize?**

Many machine learning algorithms attempt to find trends in the data by comparing features of data points. However, there is an issue when the features are on drastically different scales. For example, consider a dataset of houses. Two potential features might be the number of rooms in the house, and the total age of the house in years. A machine learning algorithm could try to predict which house would be best for you. However, when the algorithm compares data points, the feature with the larger scale will completely dominate the other. Take a look at the image below:



When the data looks squished like that, we know we have a problem. The machine learning algorithm should realize that there is a huge difference between a house with 2 rooms and a house with 20 rooms. But right now, because two houses can be 100 years apart, the difference in the number of rooms contributes less to the overall difference. As a more extreme example, imagine what the graph would look like if the x-axis was the cost of the house. The data would look even more squeezed; the difference in the number of rooms would be even less relevant because the cost of two houses could have a difference of thousands of GBP. The goal of normalization is to make every datapoint have the same scale so each feature is equally important. The image below shows the same house data normalized using min-max normalization.



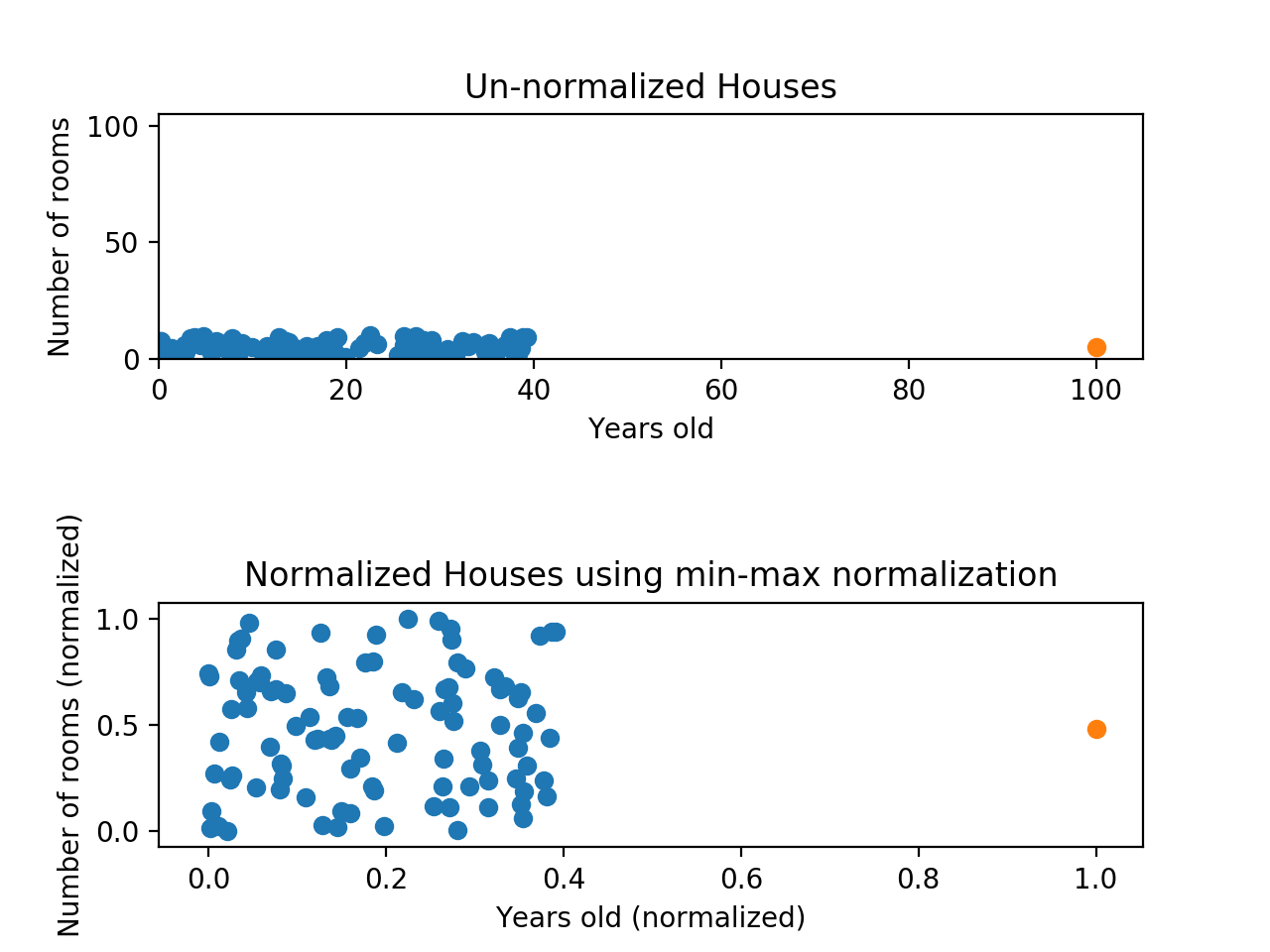
**Min-Max Normalization**

Min-max normalization is one of the most common ways to normalize data. For every feature, the minimum value of that feature gets transformed into a 0, the maximum value gets transformed into a 1, and every other value gets transformed into a decimal between 0 and 1. For example, if the minimum value of a feature was 20, and the maximum value was 40, then 30 would be transformed to about 0.5 since it is halfway between 20 and 40. The formula is as follows:

Norm\_value = (real\_value – min) / (max-min)

**Check yourselves how this formula has been derived (we can discuss it at Tutorial session as well)**

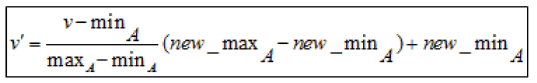
Min-max normalization has one fairly significant downside: it does not handle outliers very well. For example, if you have 99 values between 0 and 40, and one value is 100, then the 99 values will all be transformed to a value between 0 and 0.4. That data is just as squished as before! Take a look at the image below to see an example of this.



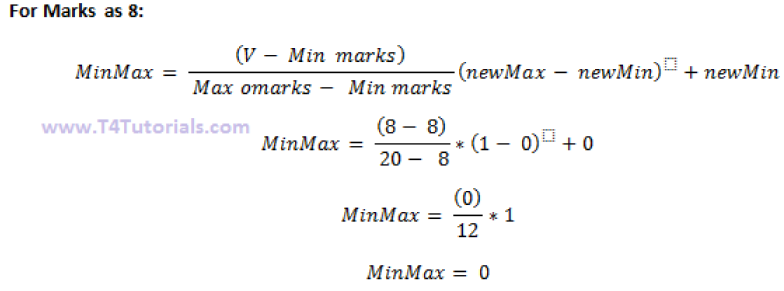
Normalizing fixed the squeezing problem on the y-axis, but the x-axis is still problematic. Now if we were to compare these points, the y-axis would dominate; the y-axis can differ by 1, but the x-axis can only differ by 0.4.

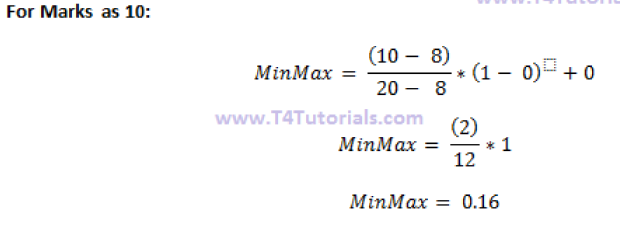
**The above normalisation formula, works only if we have 1 and 0 as the max and min normalised values. What is the case for general normalisation?**

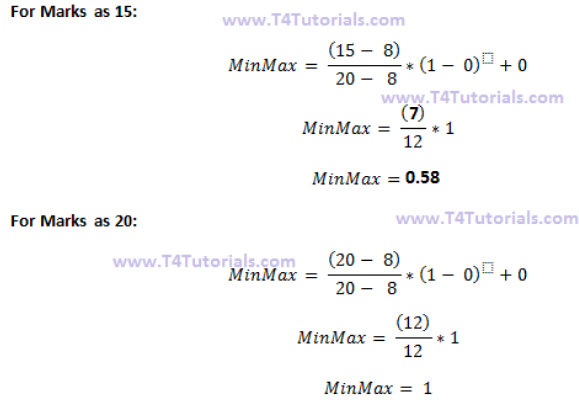
For more generic normalisation, here is the generic formula:



Suppose, we have 4 numbers: 8, 10, 15, 20, and we wish to normalise them







**What are z-score?**

A z-score measures exactly how many standard deviations above or below the mean a data point is.

Here's the formula for calculating a z-score:



Here's the same formula written with symbols:



Here are some important facts about z-scores:

* A positive z-score says the data point is above average.
* A negative z-score says the data point is below average.
* A z-score close to 0 says the data point is close to average.

Suppose, we have the same 4 numbers: 8, 10, 15, 20, and we wish to find their z- score





**Advantages of the z-score**

The z-score is a very useful statistic of the data due to the following facts;

* It allows a data administrator to understand the probability of a score occurring within the normal distribution of the data.
* The z-score enables a data administrator to compare two different scores that are from different normal distributions of the data.

**Why normalize or scale the data?**

There can be instances found in data frame where values for one feature could be in the range 1-100, and values for other feature in the range 1-10000000. In scenarios like these, owing to mere greater numeric range, the impact on response variables by the feature having greater numeric range could be more than the one having less numeric range, and this could, in turn, impact prediction accuracy. The objective is to improve predictive accuracy and not allow a particular feature impact the prediction due to large numeric value range. Thus, we may need to normalize or scale values under different features such that they fall under common range. Take a look at following example:

In these (and next) tutorials, **codes are provided in blue colour**. **Make sure that you have installed the appropriate packages before calling them via the library command.**

# Age vector

age <- c(25, 35, 50)

# Salary vector

salary <- c(200000, 1200000, 2000000)

# Data frame created using age and salary

df <- data.frame( "Age" = age, "Salary" = salary, stringsAsFactors = FALSE)

df

#Age  Salary

 # 25  200000

 # 35 1200000

 # 50 2000000

Pay attention on how values for age and salary vary in different ranges.

**Min-Max Normalization**

Data frame could be normalized using Min-Max normalization technique which is specified by the following formula to be applied on each value of features to be normalized.

(X - min(X))/(max(X) - min(X))

This is equivalent to the following R function

normalize <- function(x) {

    return ((x - min(x)) / (max(x) - min(x)))

  }

In order to apply the normalize function on each of the features of the above data frame (df), the following code could be used. Pay attention to usage of lapply function (check R documentation for further info).

dfNorm <- as.data.frame(lapply(df, normalize))

# One could also use sequence such as df[1:2]

dfNorm <- as.data.frame(lapply(df[1:2], normalize))

dfNorm

In case, one wish to specify a set of features such as salary, following formula could be used:

# Note df[2]

dfNorm <- as.data.frame(lapply(df[2], normalize))

# or df["Salary"]

dfNorm <- as.data.frame(lapply(df["Salary"], normalize))

If we wish to use the full normalization formula, then we need to create an additional function

new\_normalize <- function(x, new\_max=1,new\_min=0) # see how we define the max min values

{

a= (((x-min(x))\* (new\_max-new\_min))/(max(x)-min(x)))+new\_min

return(a)

}

fNorm1 <- as.data.frame(lapply(df[1:2], new\_normalize))

dfNorm1

**Z-Score Standardization**

The disadvantage with min-max normalization technique is that it tends to bring data towards the mean. If there is a need for outliers to get weighted more than the other values, z-score standardization technique suits better. In order to achieve z-score standardization, one could use **R’s built-in scale()** function. Take a look at following example where scale function is applied on “df” data frame mentioned above.

dfNormZ <- as.data.frame( scale(df[1:2] ))

dfNormZ

The output is given as:

         Age      Salary

1 -0.9271726 -1.03490978

2 -0.1324532  0.07392213

3  1.0596259  0.96098765

The scale function performs the z\_score. If we wish to create our own z-score function, we can do it easily.

z\_score = function(x) {

return((x - mean(x)) / sd(x))

}

dfNorm4 <- as.data.frame(lapply(df, z\_score))

dfNorm4

We get exactly the same as in the scale function.