**CSC3060 AIDA – Assignment 1**

Fionn Mc Conville

40156103

16/11/18

# Introduction

# In this report I will attempt to demonstrate and describe the differences in handwritten symbols, these symbols are in three different sub-groups; digits, letters and mathematical symbols. In my report below I will detail how I created the dataset of images (section 1), the feature engineering I performed on the dataset, (section 2) and finally the statistical analysis I performed on the feature data which enabled me to single out features which are statistically significant in discriminating between the subset groups and within the subset groups (section 3).

# Section 1: Creating the dataset

I created the handwritten symbol images using the touchscreen computers in the CSB using the application software GIMP. These images were saved as PGM files, type ASCII, with the appropriate file labelling as requested in the assignment instructions. In the code for section 1 I then converted these files to binary matrices in a csv file format using python. To do this I started by creating a function called convertToBinary() which took a PGM file as an argument and converted it to binary – (with 0s being white pixels/whitespace and 1s being black pixels) then outside that function I created a for loop that cycled through the directory containing each csv file and performed the convertToBinary() function on it. The code within the loop then converted these files to a numpy array, split the array into a 20x20 matrix then converted it to a dataframe where it was then converted to a csv file and placed in a local directory as specified by the assignment instructions. I was going to convert the files straight from numpy array to csv however this proved unreliable and so by converting the numpy arrays to dataframes first I found the results more agreeable.

# Section 2: Feature Engineering

## Again, for this section, I used python to engineer the features for this assignment. I began by loading in the csv files I had created earlier in section 1 as numpy arrays so they would be easily processed by the methods I created to calculate the features of each image. The first two methods of label and index are simply done by reading the filenames and splitting the strings where appropriate.

## Section 2.1: Features counting black pixels

This section of the report is aptly named “counting black pixels because that is largely what the next several features do, just in a different variety of ways. The methods are largely all similar though with just small variations within them according to their desired functionality.

The first method **nr\_pix** – which simply counted black pixels –totalled the black pixels using a counter. I achieved this by using a nested for loop – (for row in array: for item in row) that cycled through each element/pixel in the numpy array and incrementing the counter each time a 1 occurred.

A likewise method was used for the **height** feature – (which counted the number of rows containing at least one black pixel) however it just tested if 1 was in row. This worked well because in the nested for loop, when going through “for row **in** array”, takes the row as a separate array and python has an easy condition “in” that can return true if it identifies the element you are seeking in the for condition.

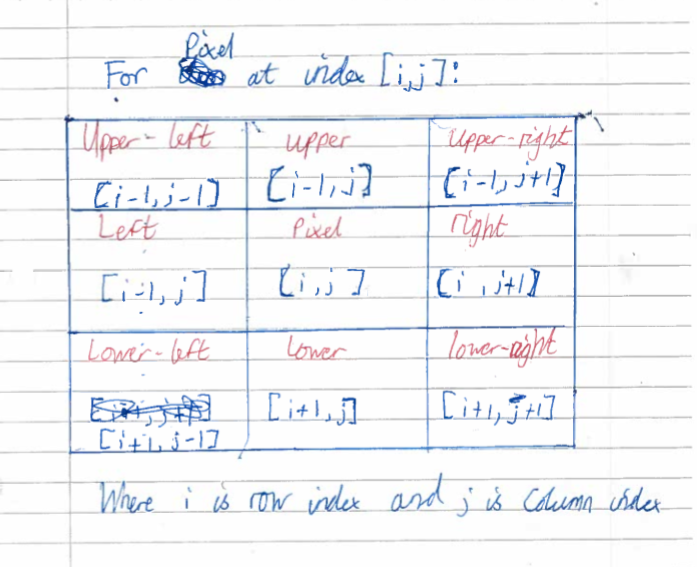
The **width** feature – (counting the number of columns containing at least 1 black pixel) was just a repeat of the height function, but it transposes the array so that columns are rows and rows are columns, in other words an element at say (5, 4) in a matrix is now at (4, 5). The width feature is just an inverse of the height feature and so by inversing/transposing the numpy array I can use the methods which are successful in calculating row-based features and use them on similar column-based features. This will come in very useful later in the assignment. The code to calculate **tallness** – (height/width) is self-explanatory.

The **rows\_with\_1** feature is calculated largely with a rehash of the code in the nr\_pix method but it has two counters, one for black\_pixels and another for black\_pixels\_in\_row – this counter is reinitialised to 0 every iteration of the first for loop – reset to 0 for each row. Within each iteration of the first for loop it checks with an if statement whether black\_pixels\_in\_row is equal to 1, if it is equal to 1 it increments the black\_pixels counter which is the eventual return value. The **cols\_with\_1** feature has code exactly like this, but its difference is that the matrix is transposed – which works as described earlier.

The **rows\_with\_5** feature and **cols\_with\_5** feature are calculated pretty similarly to their counterparts described above, however the if statement checks whether the black\_pixels\_in\_row or black\_pixels\_in\_col counters are greater than or equal to 5 before they increment the return counter.

## Section 2.2: Features counting neighbouring pixels

This section focuses on how I created methods to obtain values for features dealing with neighbouring pixels. I started by trying first to determine what exactly constitutes a neighbouring pixel and how do I calculate it within the confines of a for loop. I drew the following diagram to make sense of it:

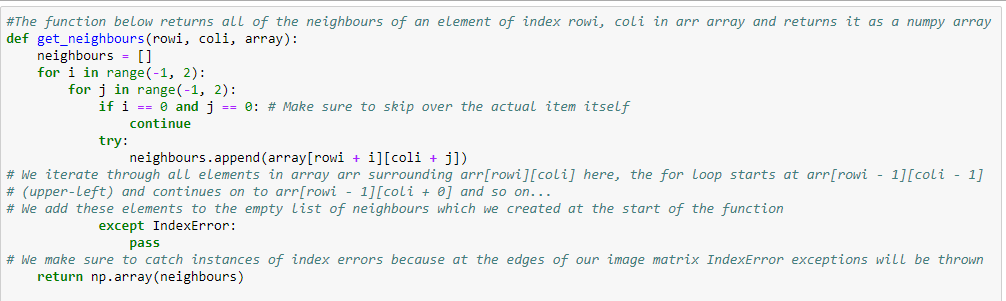


Therefore, from the outset I knew what constituted a neighbouring pixel in my dataset, I began by using an enumerated for loop, (for rowi, row in enumerate(array)) and nested within that a likewise enumerated column for loop that would allow me to log not only the row/column – but also their indexes (rowi, coli) respective to the overall array. Within the **one\_neighbour** function, I began by logging each neighbouring pixels using the code below within each iteration:



However this proved problematic as it threw up an IndexOutOfBounds exception every time the function was attempted, as the for loop obviously starts iterating at [0,0] in the numpy array we are passing as an argument and so indexes like [-1,-1] do not exist and throw up an exception. I attempted to resolve this by using a try catch block over the above code however this again proved problematic as it would skip over the entire pixel if that exception was thrown for any one of the neighbouring pixels and so any black pixel which had an index near the edges of the image would not be counted in the feature.

My overall solution to this obstacle was to create a function that would return a numpy array containing the values of each neighbouring index in the array. As an argument, it took a row index and a column index as well as the overall array., below is the function alongside descriptive comments:

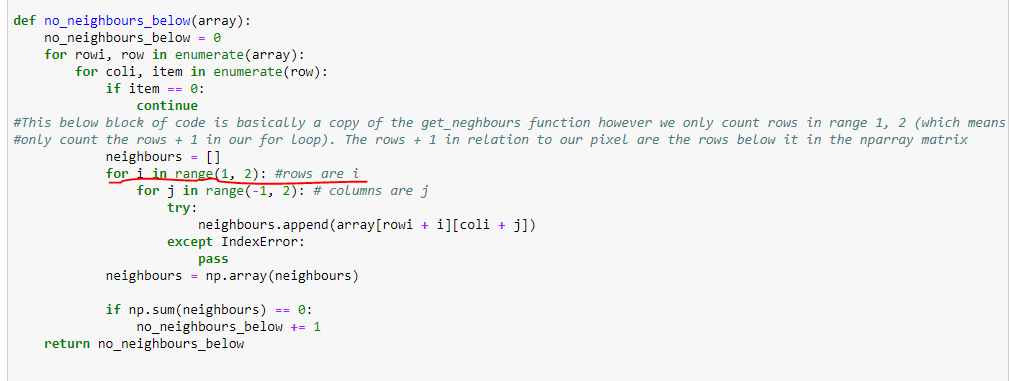


As we can see, the function iterates over the rows directly before and after as well as the columns directly before and after the index passed as an argument. The nested for loop also makes sure that it passes over the index itself with a continue statement for when the index is equal to array[rowi[[coli]. Finally, the function resolves the earlier issue of an IndexOutOfBounds exception by executing a pass on that index should an exception arise.

The lengthy description of this function is necessary because the code within it is vital to the workings of the next few functions calculating features. For instance, it is used to simplify the method of **one\_neighbour** which is calculated like so:within each nested for loop if the sum of the neighbours numpy array returned by the get\_neighbours function is equal to 1 then the counter - which is the eventual return value - is incremented by 1. The **three\_or\_more\_neighbours** function is almost identical however it checks if the sum of the neighbours numpy array is greater than or equal to 3.

In both of these functions and the following functions within this section, at the start of each for loop the index item must equal to 1 ( a black pixel) otherwise none of the functionality described above is performed on it as a continue statement makes us go to the next iteration of the for loop if the condition of the index item being equal to zero is met

The **no\_neighbours\_below** function takes the code from the get\_neighbours function and alters it slightly within it’s nested for loop by making it only iterate through the rows below it, on the following page is a snippet of the code to demonstrate:

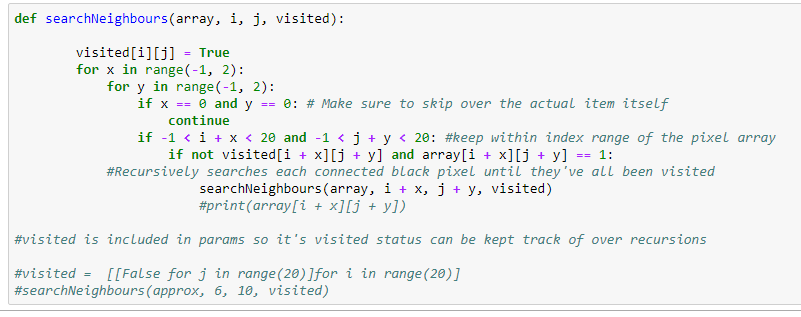


Underlined here is the area of interest, we only iterate through the row +1 of our neighbouring pixels, the columnar iterations stay the same. Again, a neighbours array is calculated for each iteration of the for loop and if it’s sum is equal to zero then the counter - which is the eventual return value – is incremented.

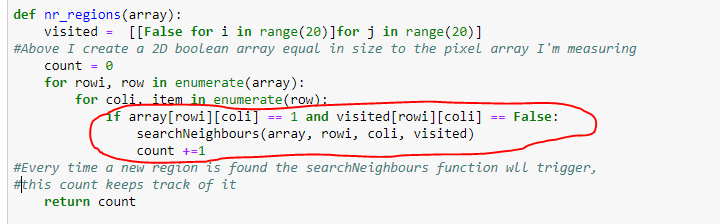
The next few functions are almost identical in functionality, but they all vary slightly on what rows/columns they iterate through depending on their desired purpose; **no\_neighbours\_above** only checks through rows -1 (i + 1) the index, **no\_neighbours\_after** only checks through columns + 1 (j + 1) the index, and **no\_neighbours\_before** only checks through columns -1 (j – 1) the index.

## Section 2.2: Features searching neighbouring pixels

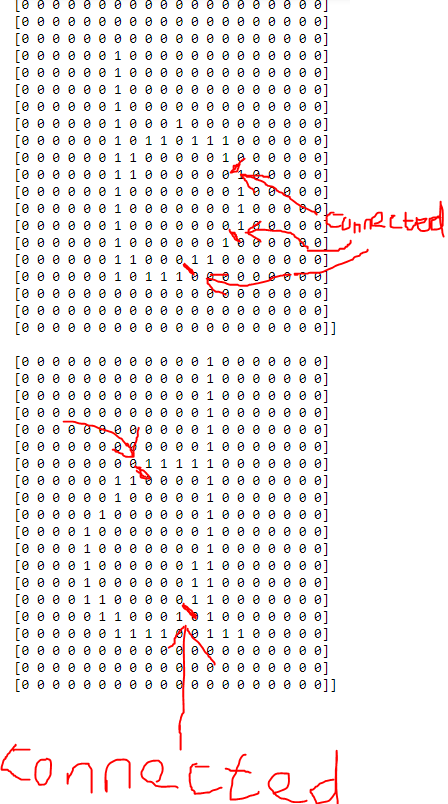
For the calculation of nr\_regions in this section I employed a recursive search to cycle through the neighbours of each connected black pixel and with each iteration of the search mark the pixel visited so that it was not mistakenly searched again. I used the Boolean visited in my code to mark if the index in question was visited, below is my code for searching the neighbours of each black pixel:



This function was used in the **nr\_regions** code, as the recursive nature of the function meant that it only broke out of it’s recursion when all connected black pixels were searched – which is exactly the functionality I required for the nr\_regions feature. When this function broke a counter – the eventual return value – was incremented. Below is the function of nr\_regions with the important area in question highlighted:



The purpose of the **nr\_eyes** feature was quite similar to the nr\_regions code in that it too was calculating a region – just that the region was not a maximally connected set of black pixels but rather it was a connected region of whitespace. I assumed that the function would transfer over easily, and that I could just substitute the 1’s that nr\_regions was searching through with 0’s.

However, this proved not to be the case, what I found was that the nr\_eyes function, if implemented in this way, would never identify a region of whitespace surrounded by black pixels as it would exit the region by sometimes going to a **diagonally connected** white pixel, alongside is an illustration to demonstrate.

With this problem the solution was then easy once the issue was identified, I prevented the function from searching diagonally connected pixels, by slightly altering the searchNeighbours function with one line of code; after the nested for loop I inserted an if statement to check if absolute x was equal to absolute y – this meant that all instances of diagonally connected indexes were excluded from the search as the diagonally connected indexes are at (-1, -1 – upper left, -1, +1 – upper right, +1, -1 – lower left and +1, +1 – lower right) . This was made into a function; searchNeighboursNoDiagonal and it worked seamlessly from there

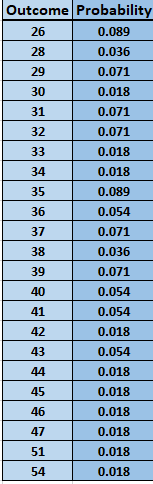
## Section 2.3: Other features and exporting the results to a csv file

Late on in this assignment I decided to try to attempt to create the bd feature function as well as the two improvisational feature functions. I had success with the b\_or\_d function, with an uncomplicated method of identifying a column which had 7 or more pixels in it, then later on in the function determining if that long vertical line was on the left or right side of the image by seeing which side of 10 (the midway index in a 20x20 matrix) that long vertical line was on. This simple method proved effective when only having to distinguish between a b or a d but I’m not sure how useful it would be other than that.

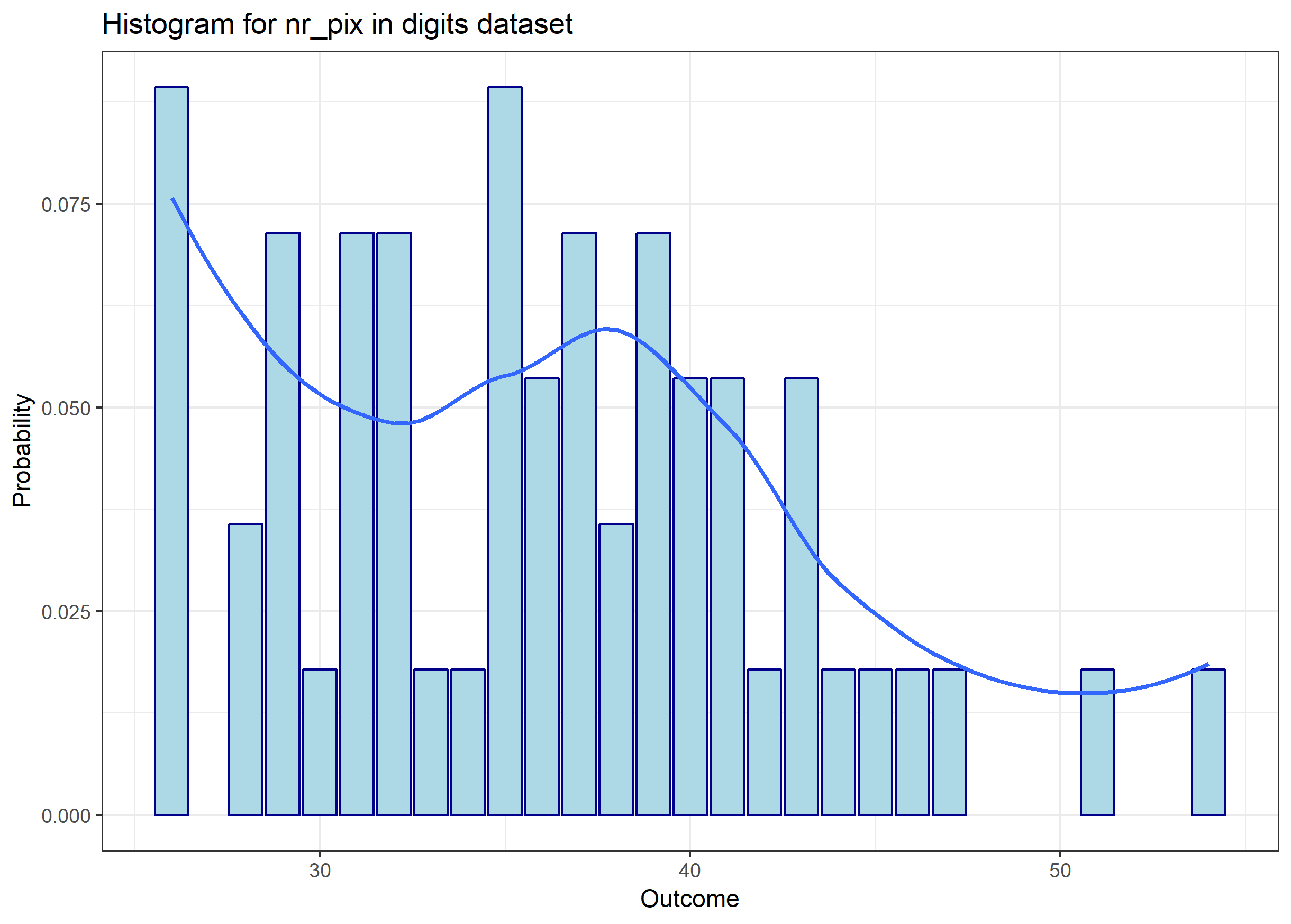
The other two features I devised late on in the assignment were the identify\_isolated\_horizontal function and its vertical counterpart. I came up with these functions late, so I was not able to test if they were good for distinguishing symbols statistically, or if they were error prone on a large scale, still I think the idea is a good one and it could have been built upon further given time.

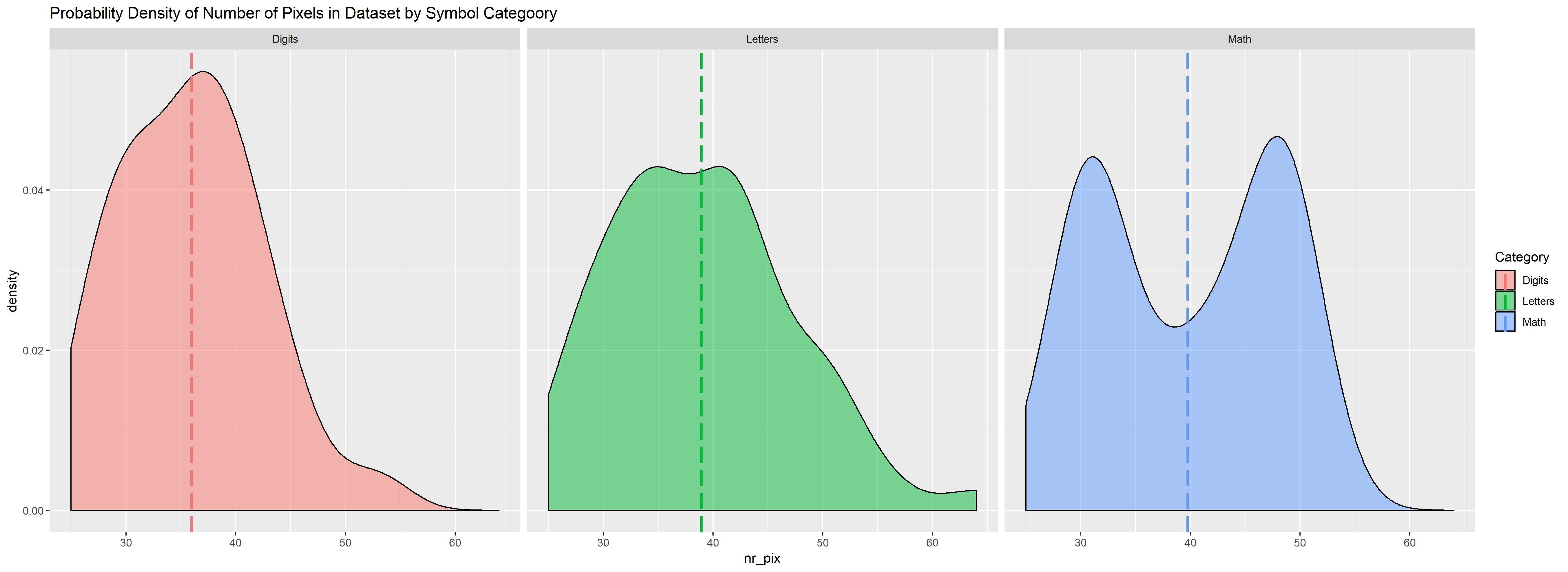
Finally, to put the features in a csv file so that it could be processed later by my R code I created the function calculate\_features() which did an Os.walk to iterate through each csv file in my earlier created ..section1\_images directory and created a stacked numpy list for each of the 168 images in that directory which it then returned. This returned list was converted to csv and exported to the local section2\_features directory as per instructions in the assignment.

# Section 3: Statistical analysis of the feature data

For this section, as requested, I used R to statistically analyse the feature data of each handwritten image. A significance value of 0.05 was used, therefore for most of my findings I had a 90% confidence level – this is the case unless specified otherwise. This was used on most occasions, except when doing multiple comparisons, in which a correction was made.

## Section 3.1: Probability distribution for the nr\_pix feature

Below I will show the probability distribution for each of the three symbol groups and briefly discuss the shape of the distribution for each group, as well having the accompanying probability distribution table below it. Originally, I set out to display the probabilities like so:

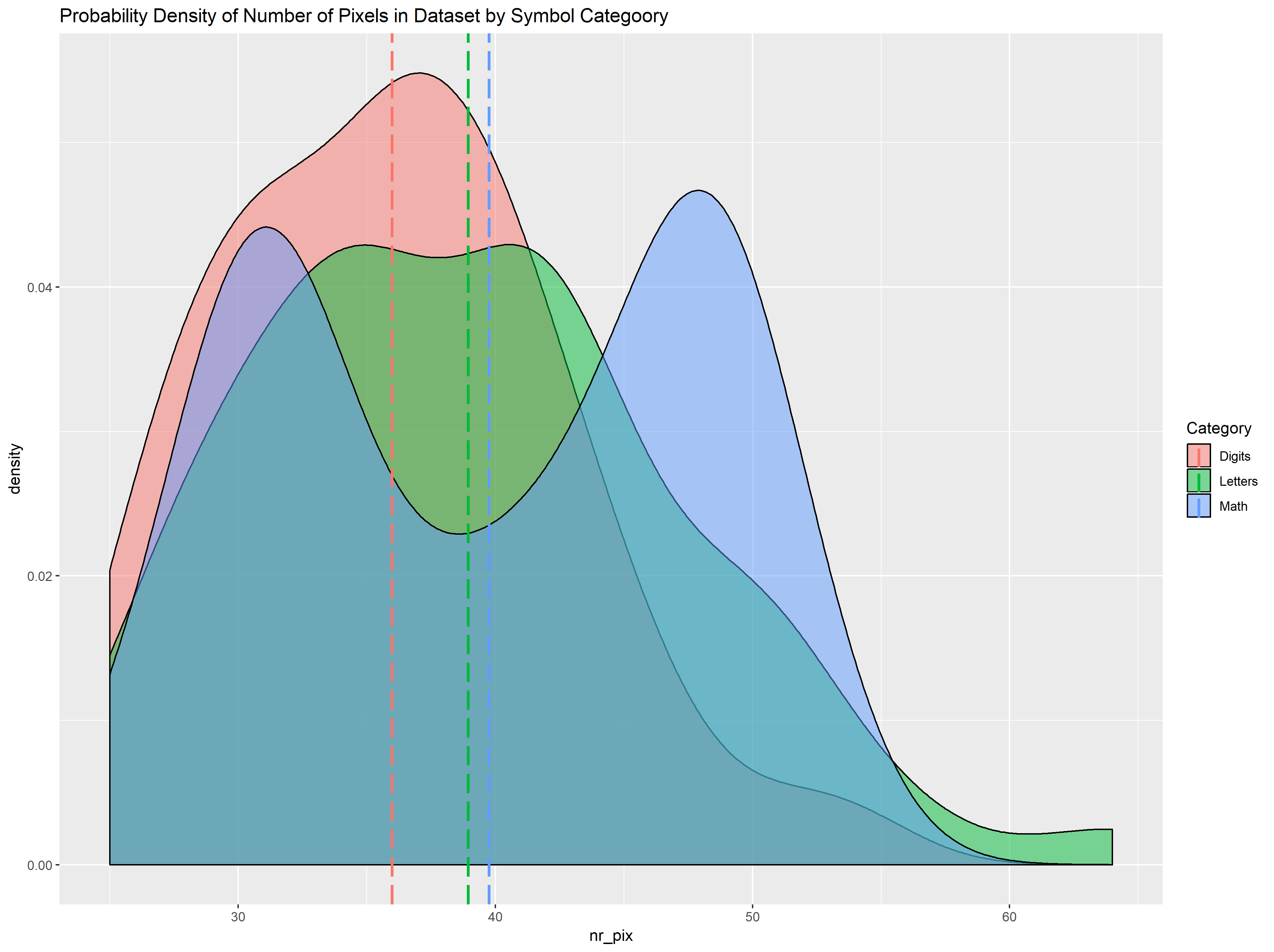
However, this looked unkempt and I only was required to give a rough estimation of the probability distribution. With time, my skill in R improved throughout the assignment, so I created the graph below to more cleanly showcase the probability distribution of each symbol group:

For the digits dataset (in red), the distribution seems right skewed with a mostly normal distribution. It seems pretty unimodal with its peak at the start of the graph. It has a few outliers beyond the 50-pixel mark, this might come from the same symbol, and it could prove useful later when I need to distinguish a digit from the rest of the group. Despite these outliers the distribution seems relatively normal

The letters dataset is unmistakably unimodal, with again, a right skew. Like the digits dataset it has a few outliers, but this time they seem to be beyond the 60 mark, which could prove useful in distinguishing from within that sub-group later.

The math symbols dataset has a bimodal distribution and is symmetric in its distribution, as a result of this it does not seem to have many outliers.

The probability density graph above demonstrate that within my dataset, the feature values for nr\_pix are safe to assume normality on and within each of the datasets for the nr\_pix feature the data collected is not extremely skewed, which will be useful for later when hypothesis testing is used. Below is the probability distribution for nr\_pix on each group within the same axis.

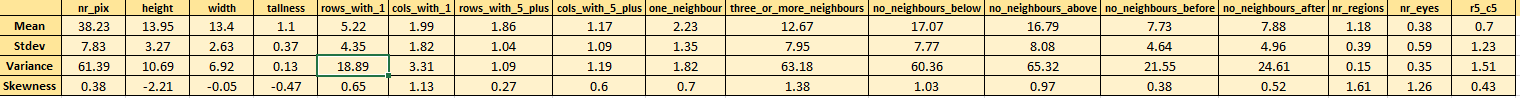
The dashed lines in this graph are the respective means of each dataset and as we can see, the mean number of pixels for the digits dataset is the least, followed by the letters dataset, then closely followed by the math symbols dataset. This graph helps us to compare the difference in feature values for each group and I may use it more throughout this report to display these differences for feature values.

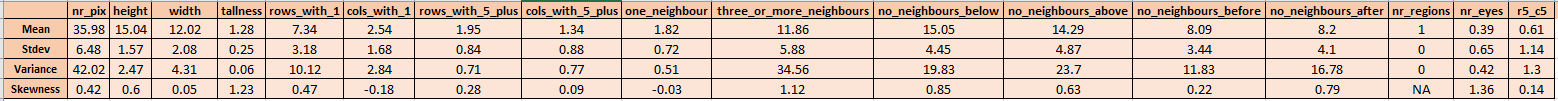
## 

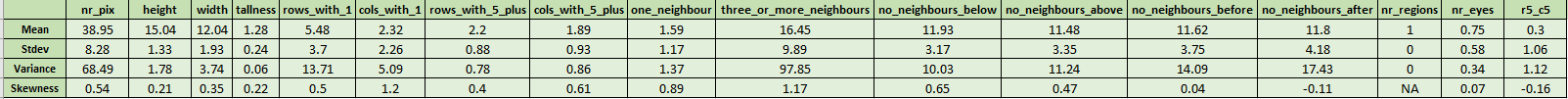
## Section 3.2: Probability that a random digit image has greater than 20 pixels

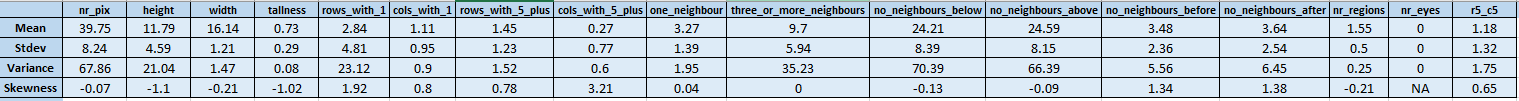
The probability that an image with greater than 20 pixels is 1, or certain. Each image within my digits group of symbols has more than 20 pixels. Alongside is the probability distribution of number of pixels within the digits dataset, (shown before) to demonstrate:

## Section 3.3: Summary statistics for all the features

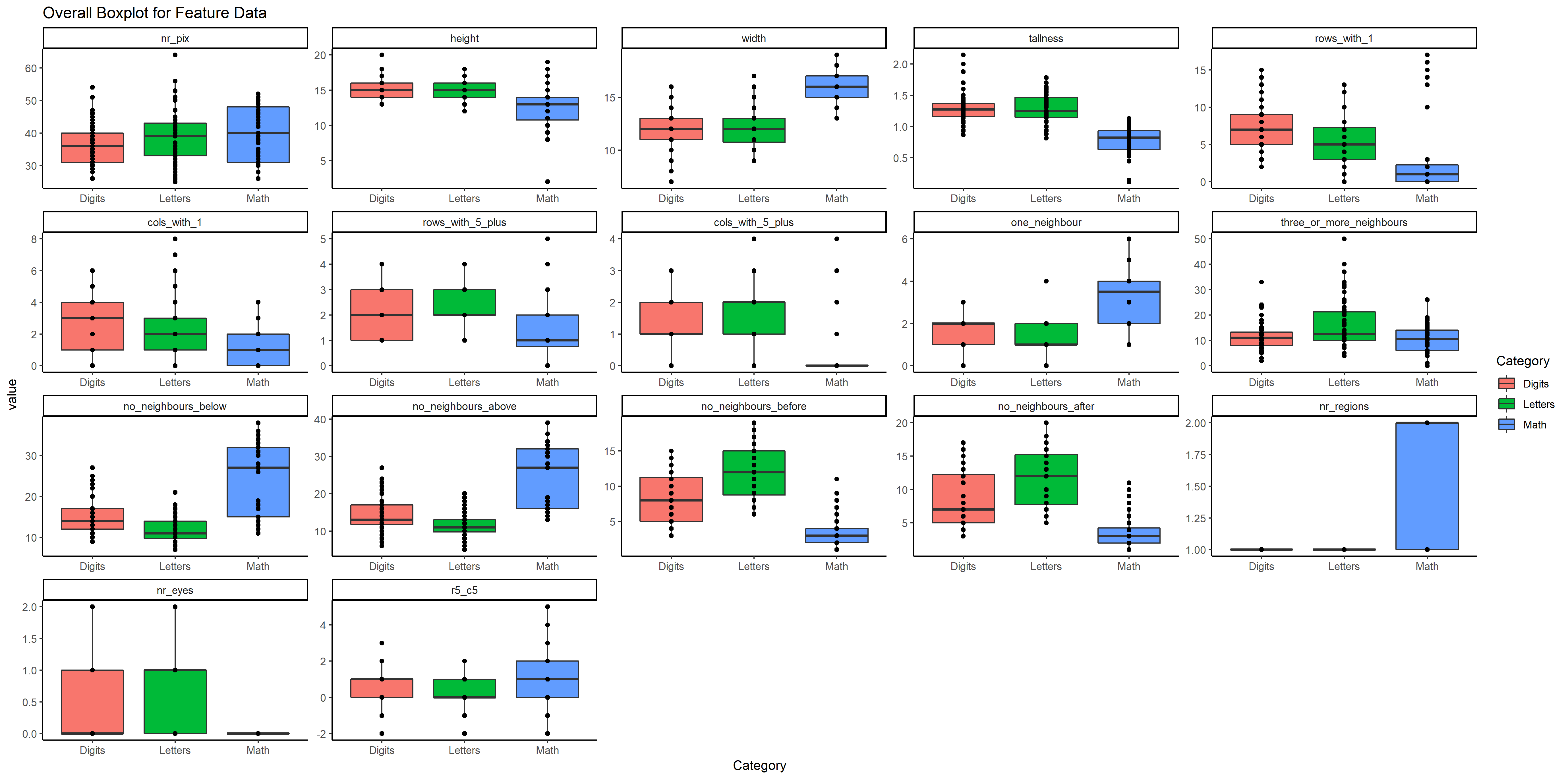
**(a): Summary table for all of the features in overall dataset:**

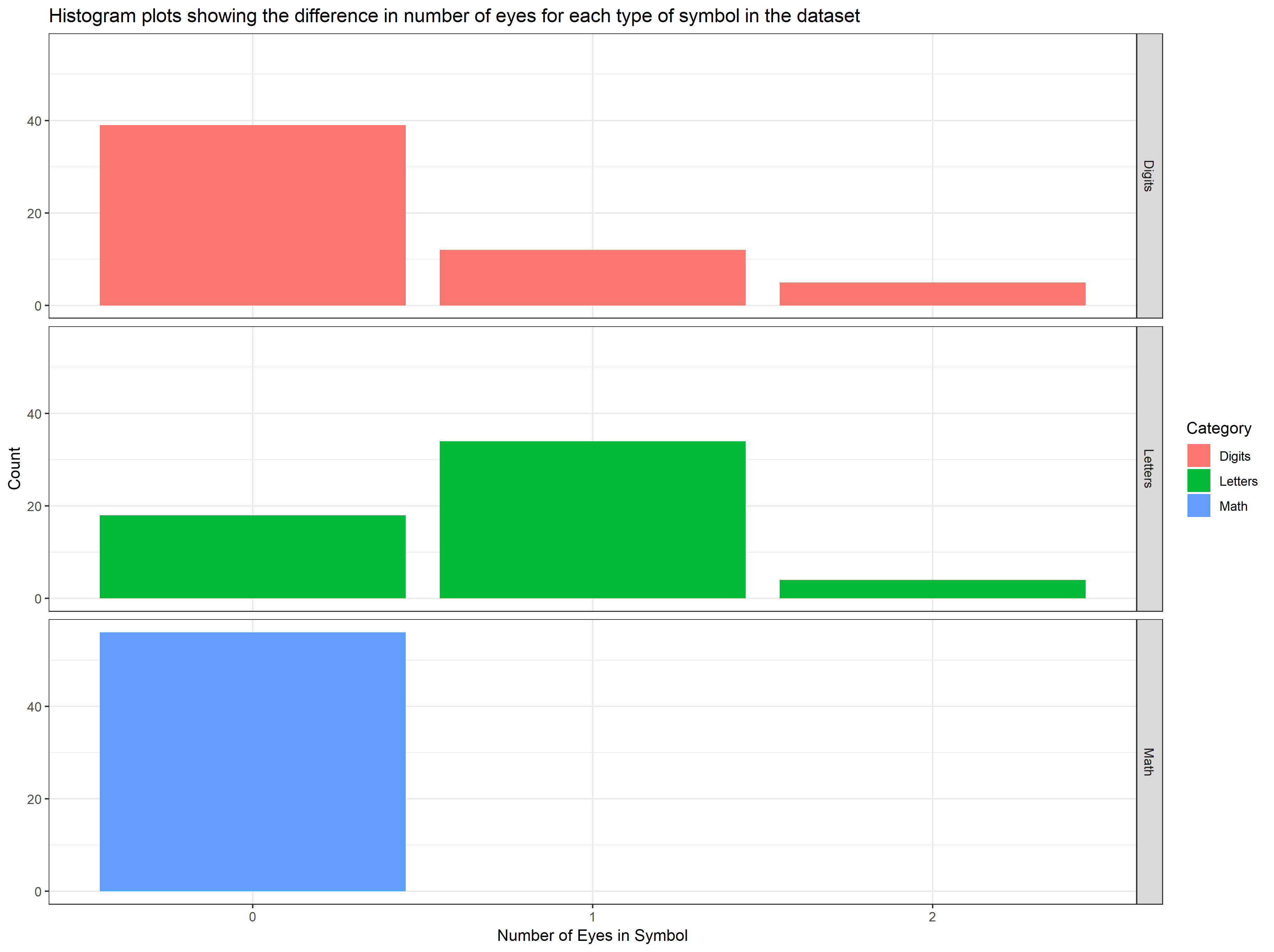
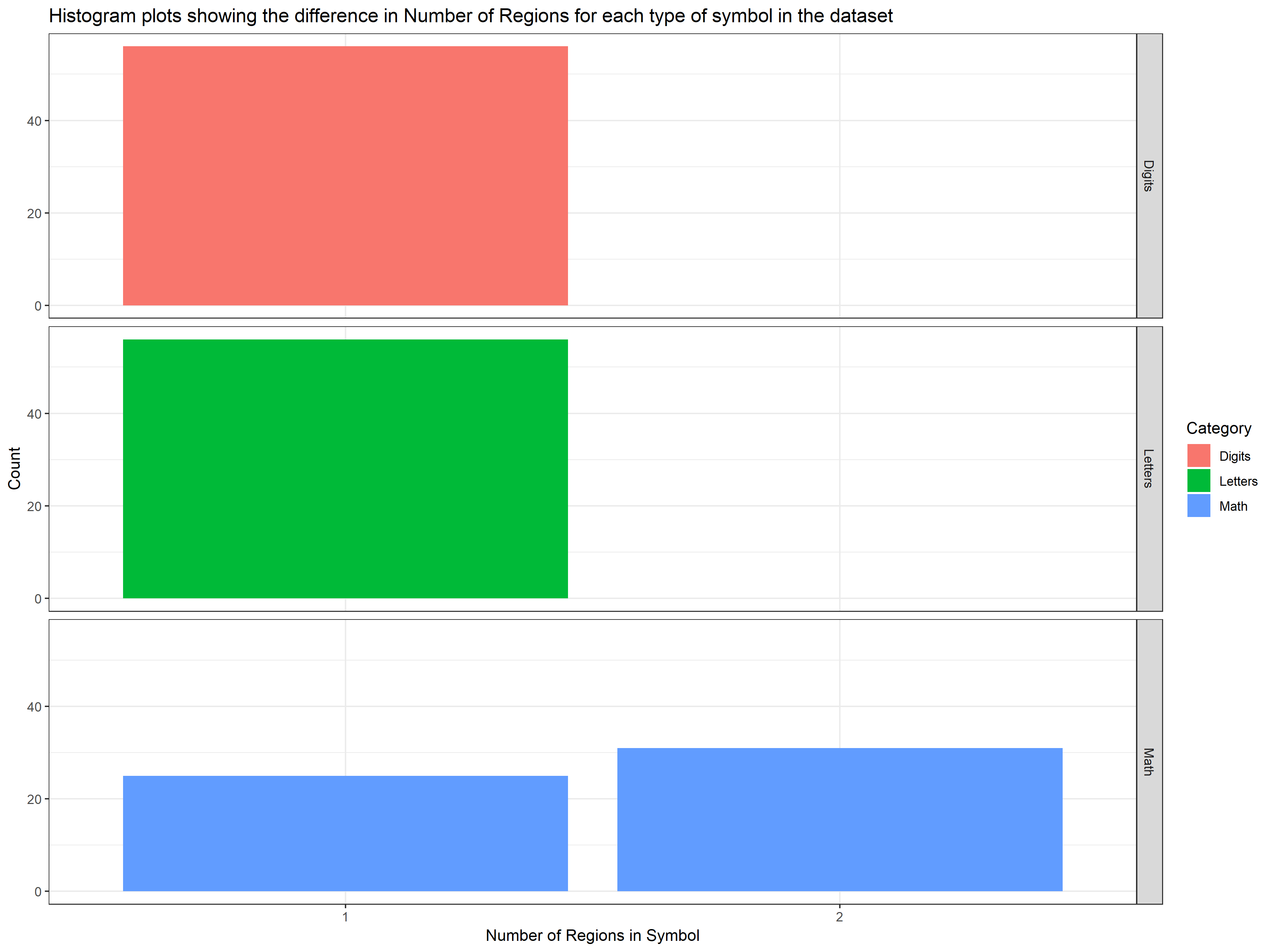
**(b): Summary table for all of the features in digits sub-group:**

**(c): Summary table for all of the features in letters sub-group:**

**(d) Summary table for all of the features in math symbols sub-group**

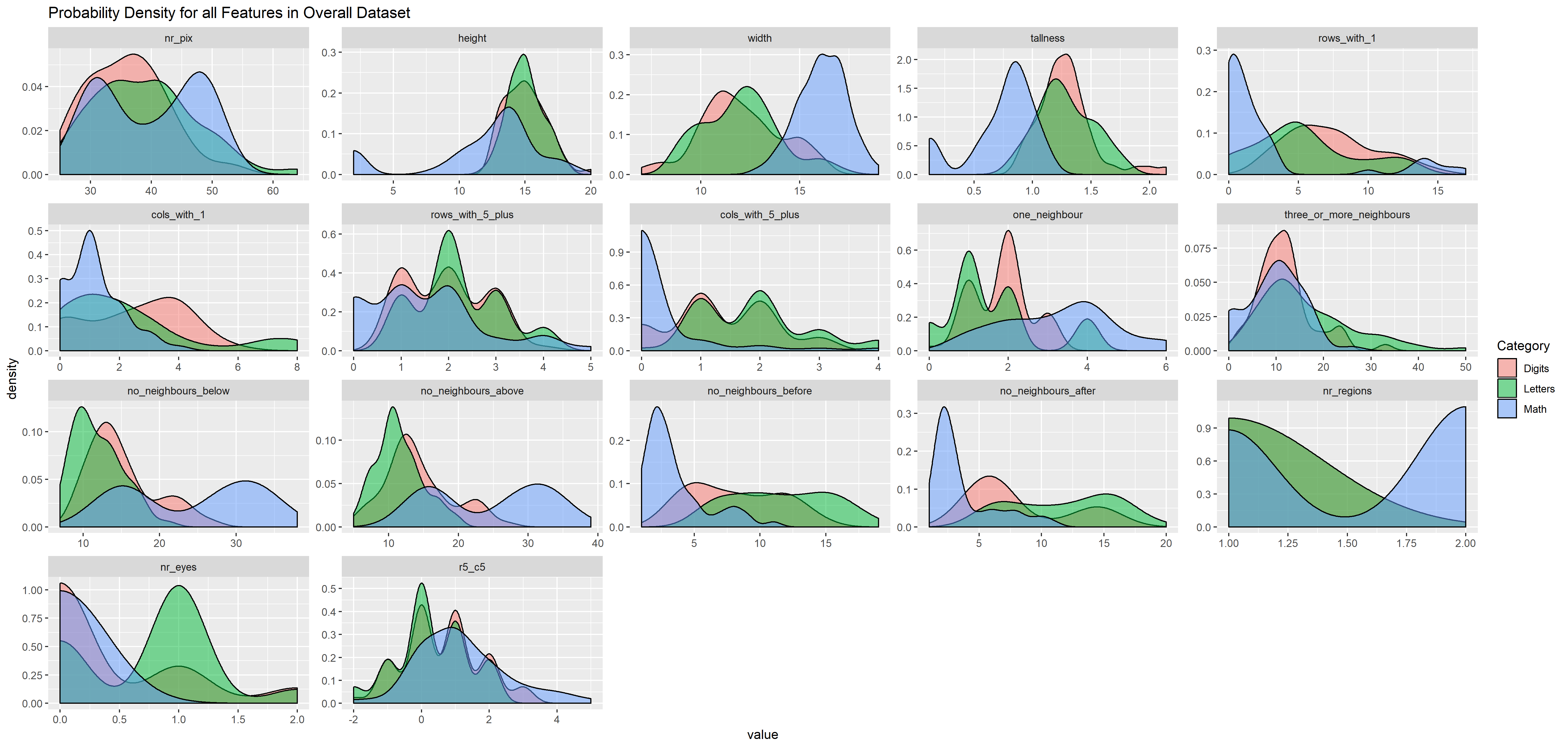
Above are the summary tables showing the mean, standard deviation, variance and skewness for all the features in my dataset. None of the skew exceeds in any of my tables the extreme value of 3 or -3 so the assumption of relative normality can be inferred for each feature. However, these tables are difficult to use to distinguish between the different groups, as the reading of numbers between each table can be tiresome. Below is a boxplot for each feature which better summarises and distinguishes the differences between the groups in my dataset:



We can already see here some noteworthy differences, namely, that math dataset is the only group that has symbols with two regions and also that no math symbol in my dataset has an eye in it. Below are some histograms which corroborate my findings:

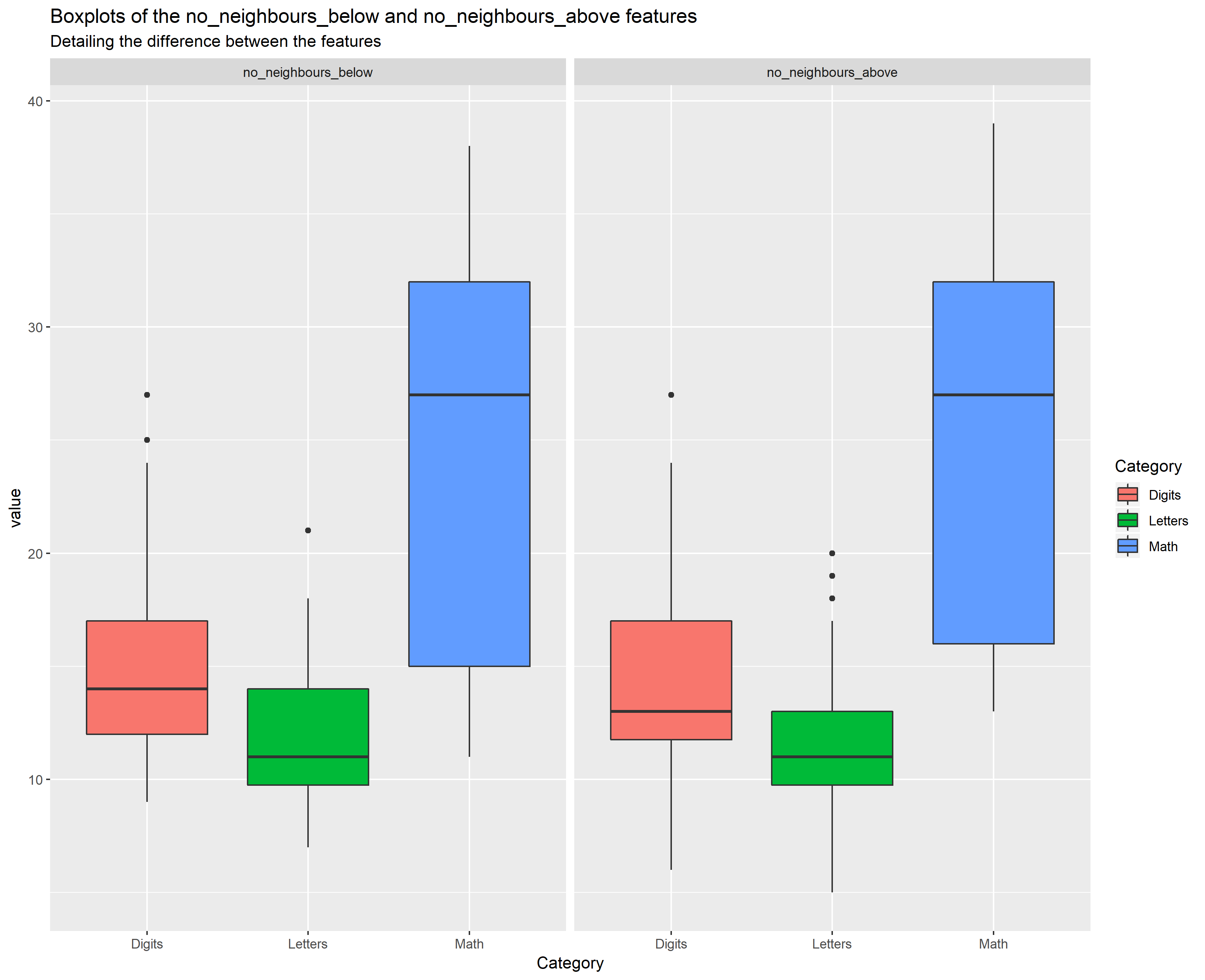
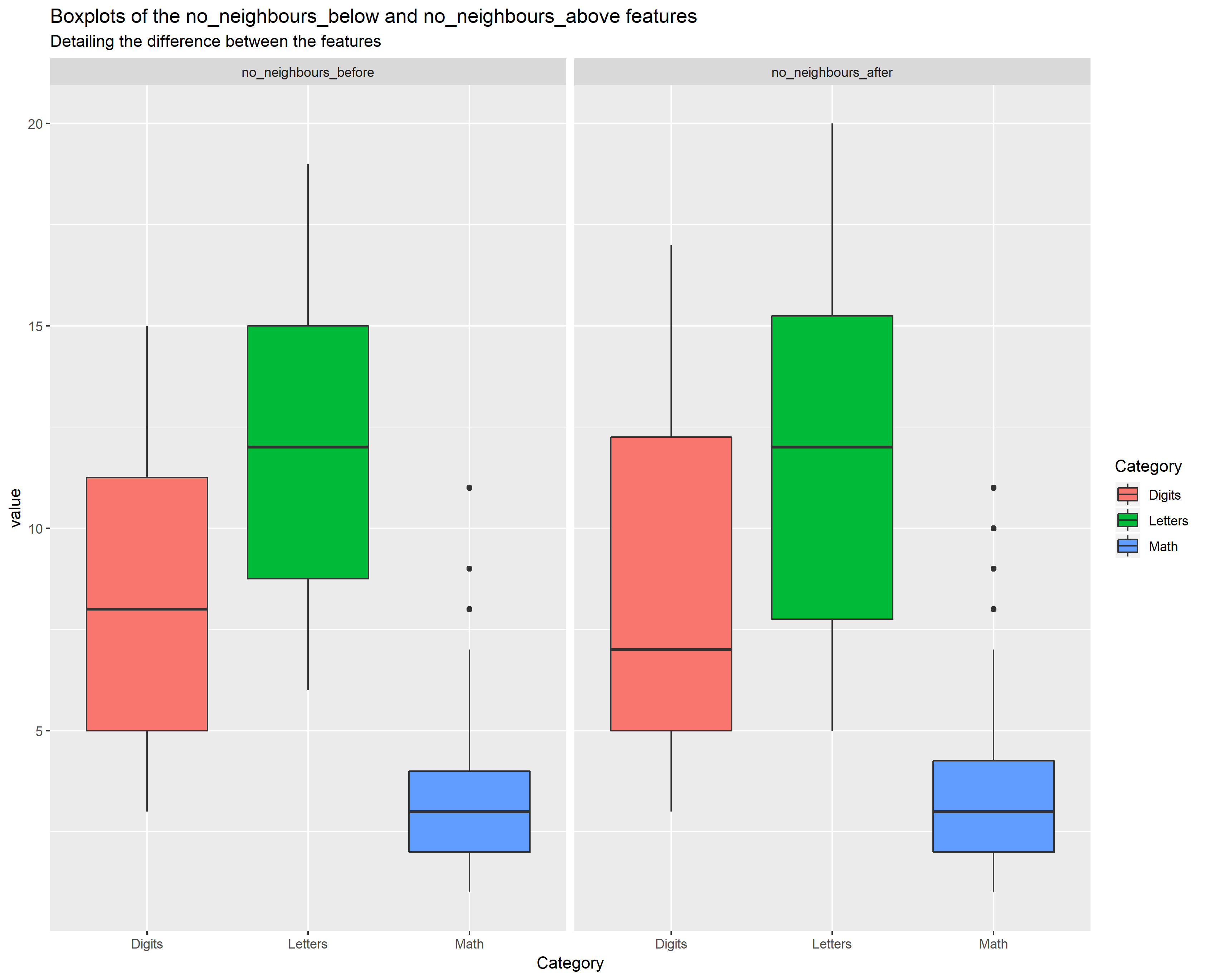
Also, there seems to be no math symbols which have a column with more than 5 black pixels in it which seems to suggest there are no symbols in my math symbols data that have a straight vertical line in it. This is further confirmed by the difference in the no\_neighbours\_before and no\_neighbours\_after feature that the math dataset has with the other groups. These feature values are calculated by analysing adjacent columns – symbols that have a vertical line in them like a 1 or an f have large values for these features, this may be a plausible explanation for the difference we see.

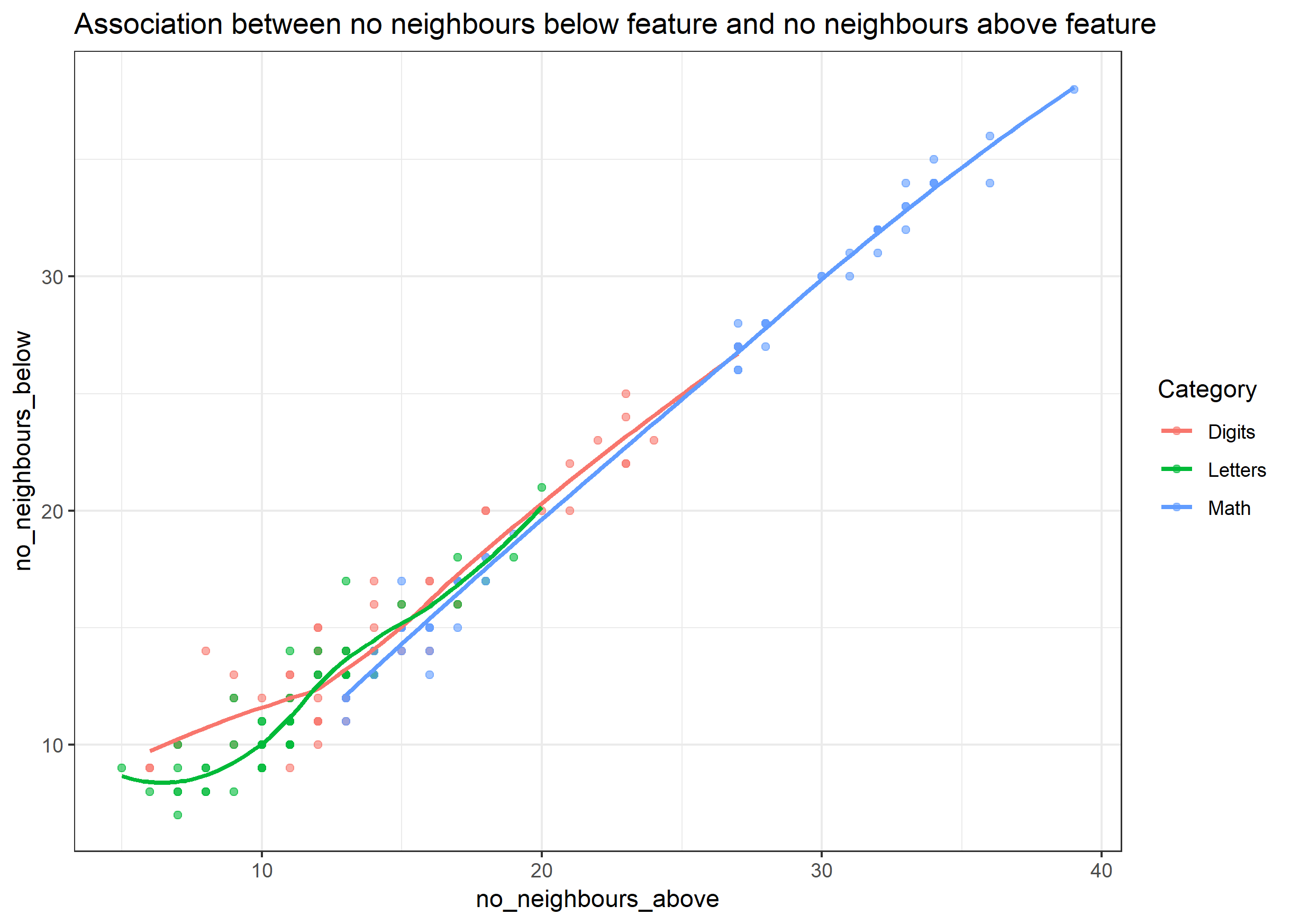
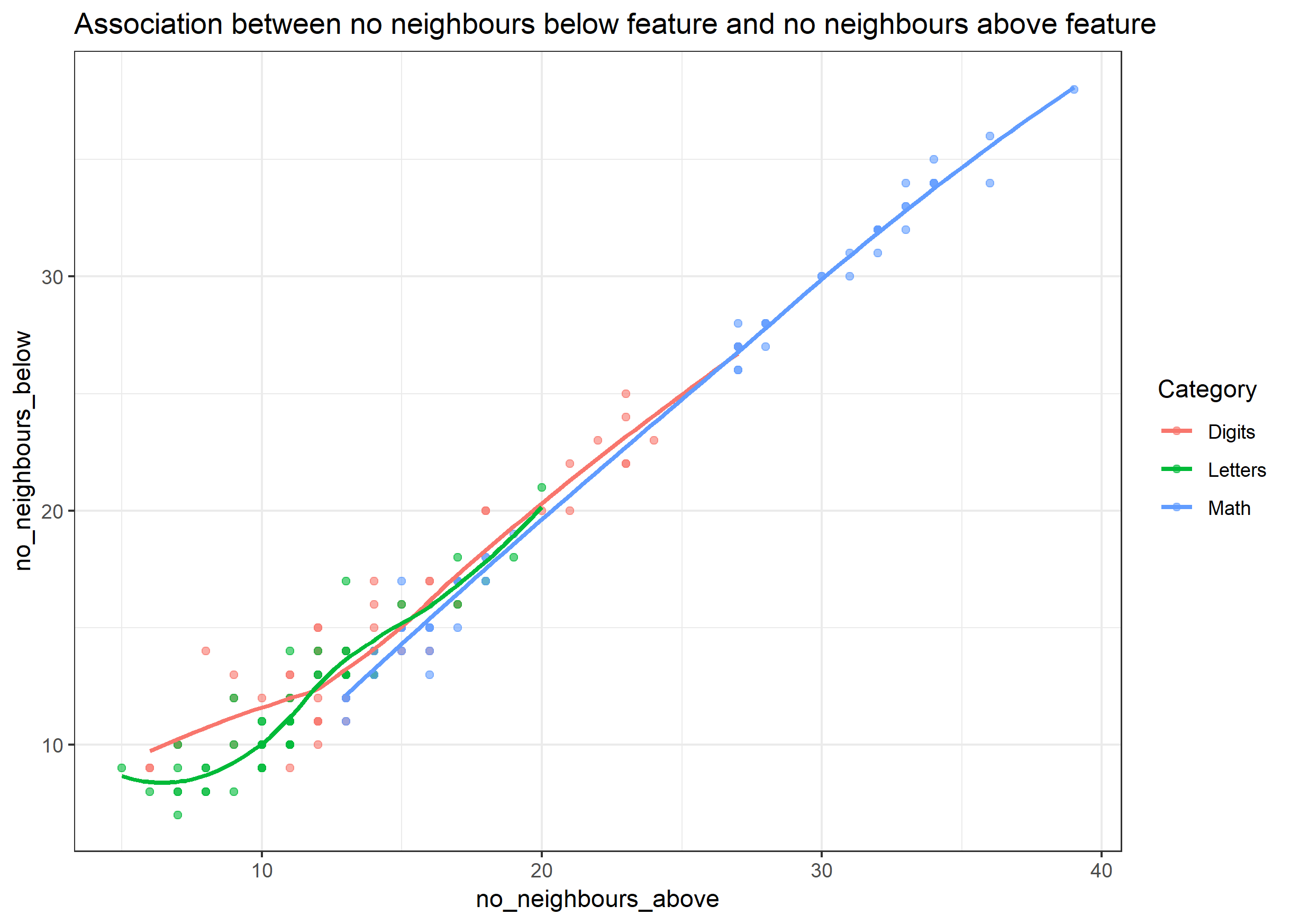
However, these boxplots do not as much to demonstrate as clear a difference between the letters and digits dataset, perhaps a probability density curve may show a clearer contrast:



The curves of digits and letters are largely in line with each other whereas often the math symbols curve is much different. A place where the letters dataset is clearly different than the digits dataset is in the nr\_regions probability density graph, however I suspect that this is an error in the way one of the handwritten letters was drawn rather than anything significant as the probability of a letter in my data having 2 regions seems very small going by the graph.

## Section 3.4: Highly associated features

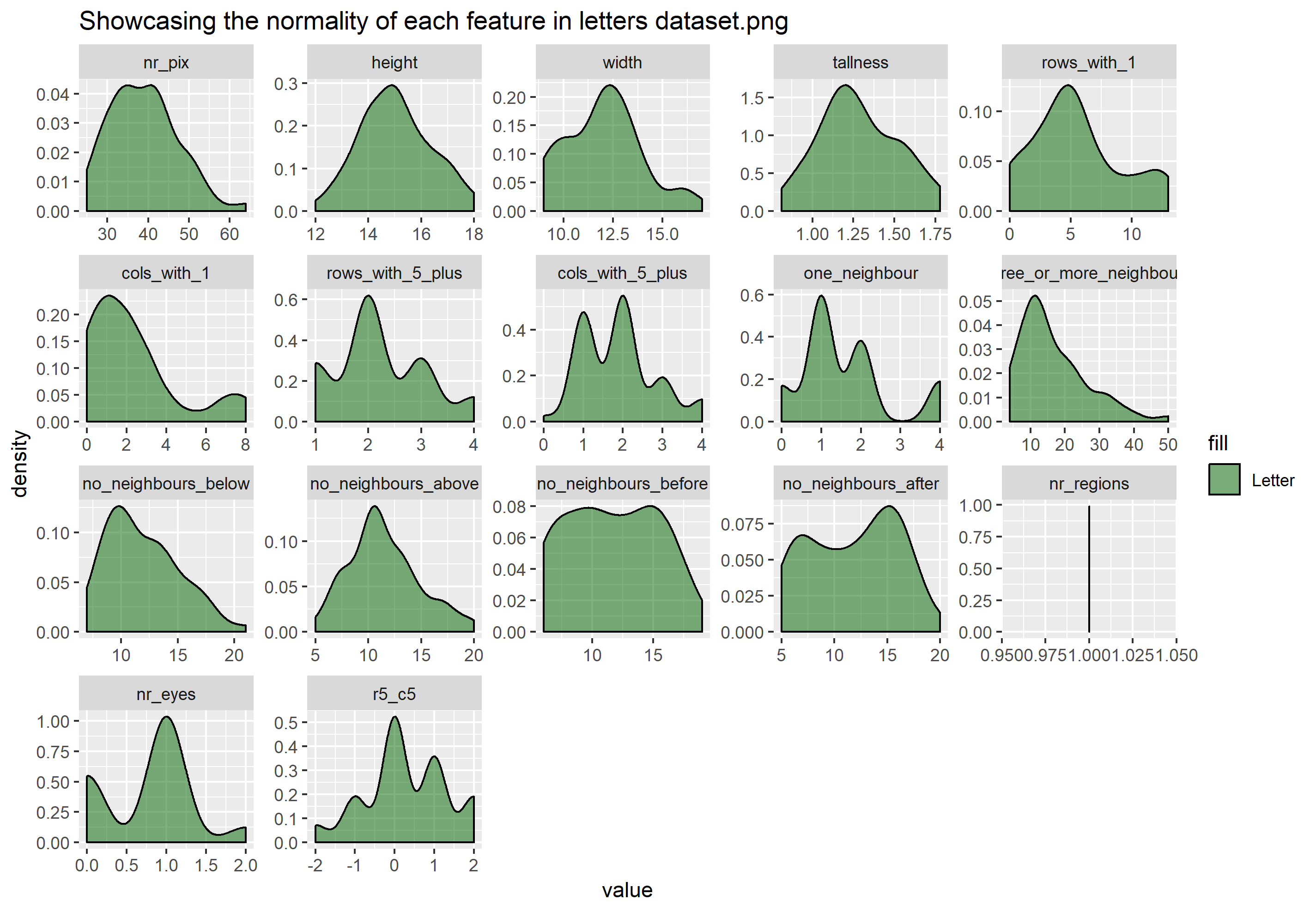
Using the boxplots I created earlier I was able to identify features that were highly associated with each other. Below they are demonstrated:

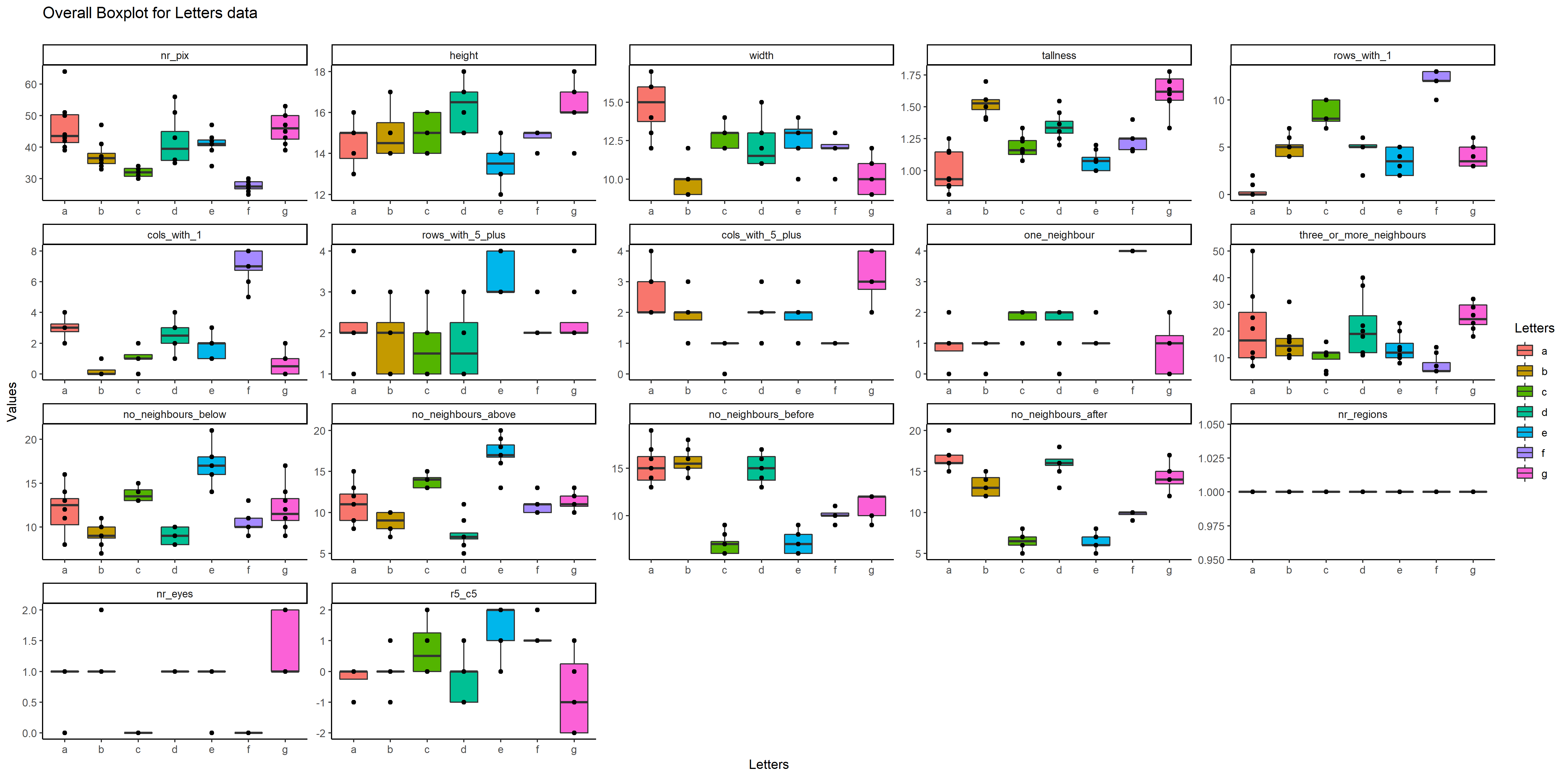
As we can see from above the no\_neighbours\_before and no\_neighbours\_after features are nearly the same in value each time and thus are highly associated, this is likewise the case for the no\_neighbours\_below and the no\_neighbours\_above features. This can be further validated by the graphs below also:

These graphs demonstrate a near identical association. Perhaps the reason for the similarities in these features can be explained by what I mentioned earlier, that being that many of the black pixels that are counted by these features, are usually either straight horizontal or vertical lines, and so are isolated on both fronts – isolated above and below in the case of horizontal lines, and isolated before and after for vertical lines. I’d be hesitant though to discard of the features from each of these associated groups as thy could be of use later possibly. A better solution would be that the features could be combined – e.g no\_neighbours\_before\_or\_after and no\_neighbours\_below\_or\_above for new features. These features could help identify symbols that have straight horizontal lines and straight vertical lines later.

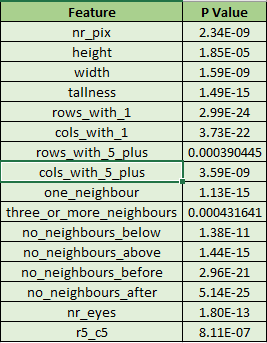
## Section 3.5: Features that are useful in discriminating between the different letters

For the next few questions – (section 3.5 – 3.7) I required a statistical method that tested for statistically significant differences between multiple groups as there are 7 different groups of symbols within the letters and digits dataset, and 3 different sub-groups in the entire features dataset. Therefore, I used a two-way ANOVA test – (not a one way test as I only had to discover if there were differences in the sample means). Before I did this, for each question, 3 conditions had to be met, these are described below. **Nearly normal observations:** As mentioned before the assumption of normality within each feature is a founded assumption. Below is a probability density graph for each feature in the letters dataset to further justify this:



Also necessary is the **independence of the samples within and between each group**, these handwritten symbols are less than 10% of the population of all handwritten symbols for each of their groups so this condition is met. To demonstrate **equal variability along groups**, as well as handily display the differences between letters I created a boxplot:

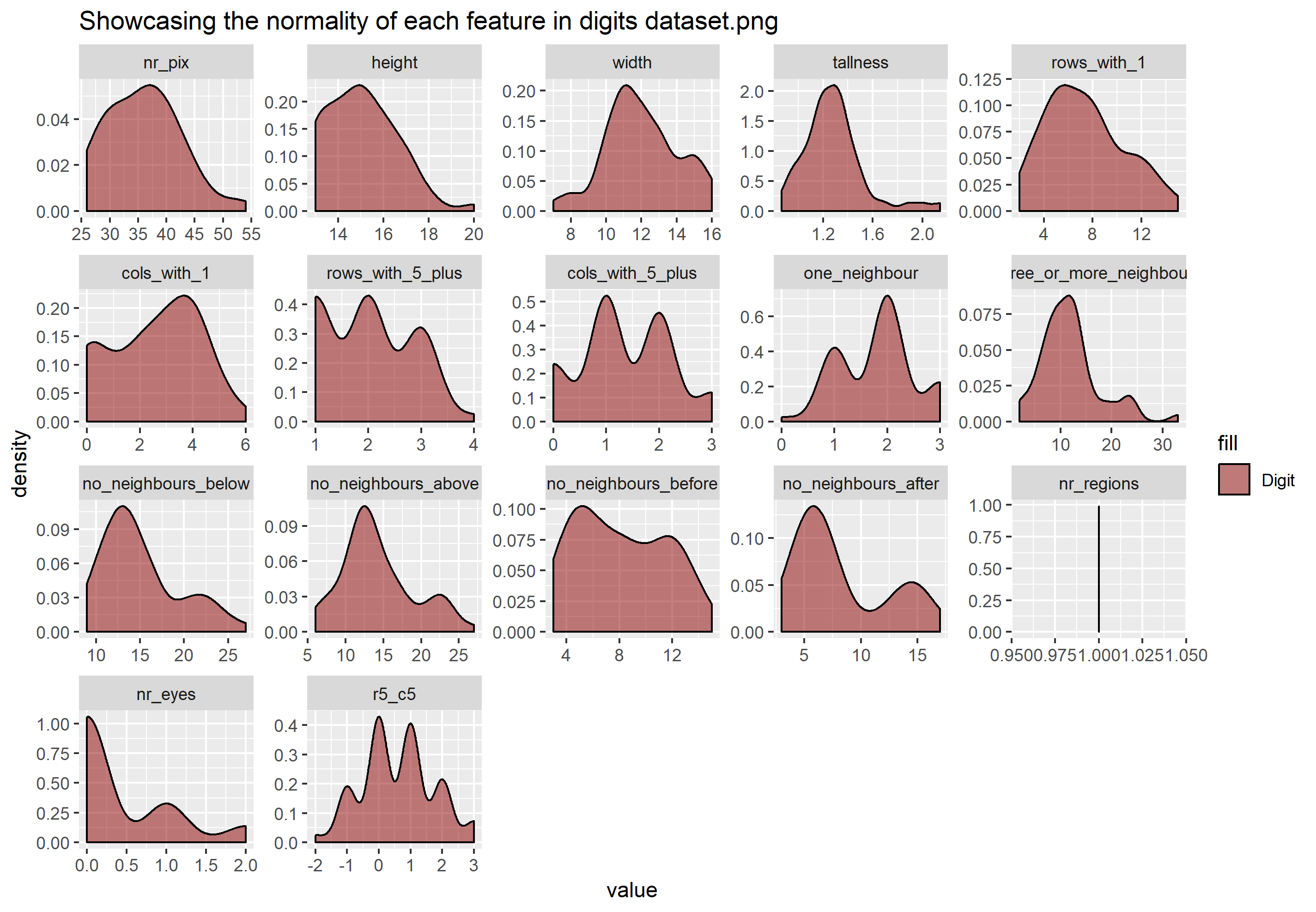
For each of the following questions I will take the steps above to demonstrate that the conditions for an ANOVA test have been met.

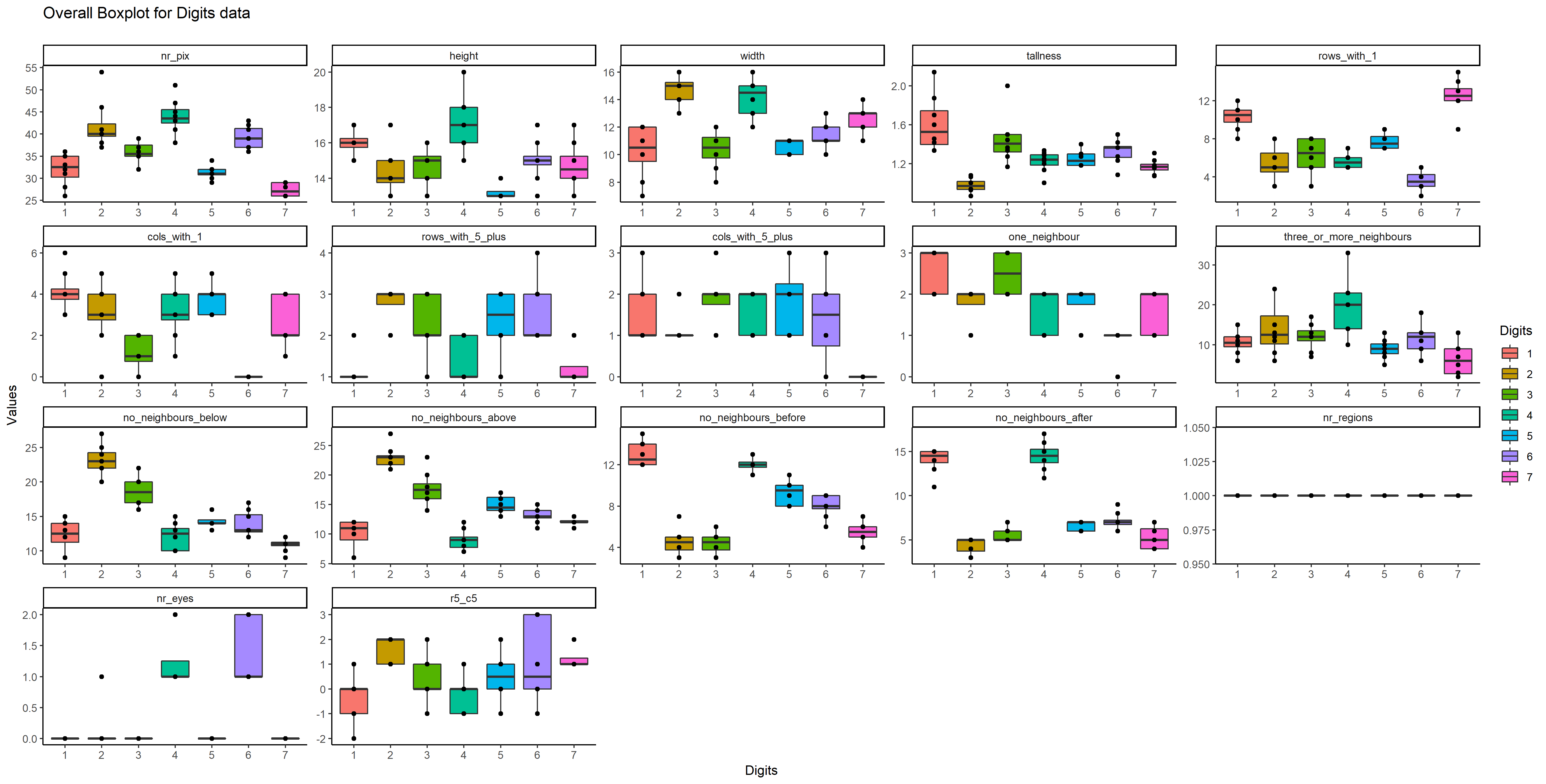
The results of my ANOVA test, with a significance value of 0.05 returned these features as being useful to discriminate between the 7 different letters, if one looks at the boxplot above, you can see that these calculations are largely accurate. The only feature that does not return statistically significant is the nr\_regions feature as it is equal to 1 for each of the letters. This ANOVA test is merely a preliminary test If the feature returns a p value that is less than our alpha, it does not mean that the feature is useful for distinguishing for all groups of letters all the time but rather it can be significant in distinguishing at least one letter from the others. The features which seem to have the smallest p-value – i.e the most useful in discriminating between letters is the no\_neighbours\_before and the no\_neighbours\_after features – this is probably because they isolate letters that have a long vertical line in them – letters like a, b, d, f and g are separated from c and e which are more horizontal in their orientation. The rows\_with\_1 feature is marked as especially significant for the same reason. I would have expected that the nr\_eyes feature would have been the largest discriminator for letters but I believe that the large variance and error within the g symbol threw off the ANOVA test. The hypothesis test for each feature is as follows:

H0 = The mean value across each letter is the same

HA = The mean value for at least one pair of letters is different

## Section 3.6: Features that are useful in discriminating between the different digits

The relative normality of the feature data for the digits dataset is demonstrated below:

The notable non-normal distribution in this data is the nr\_eyes feature, a reason for this is highlighted in the following boxplot – which also reassures us that the variability along groups is nearly equal too.

The nr\_eyes feature singles out the digits 4 and 6 from the rest of the group quite clearly, again there must be some error in the way that some 6’s were originally drawn as it’s suggesting that some of the images have two eyes

Alongside are the results of my ANOVA test, the hypothesis test for each feature in this ANOVA test is as follows:

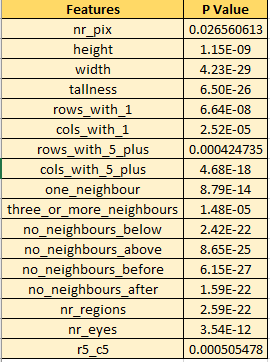
H0 = The mean value across each digit is the same

HA = The mean value for at least one pair of digits is different

Again, the only feature that does not have any statistical significance is the nr\_regions feature - as each digit has only one region. In a manner identical to the letters dataset, the most significant feature are the no\_neighbours\_before and no\_neighbours\_after features, with the no\_neighbours\_below and no\_neighbours\_above features following closely behind. The reason for this, I assume, is again identical to the reason discussed above.

## Section 3.7: Features that are useful for discriminating between the three groups

## I can use a two-way ANOVA test again to distinguish differences between the three groups, with this ANOVA test the results we obtain should be more concise and we can be more confident of their findings as we are comparing only between three groups rather than seven like before. The assumptions are like before except we are more confident in the normality of our feature data as a larger sample size usually leads to more normality in our data. The variability is roughly equal within and between the feature data for the majority of features as shown below:

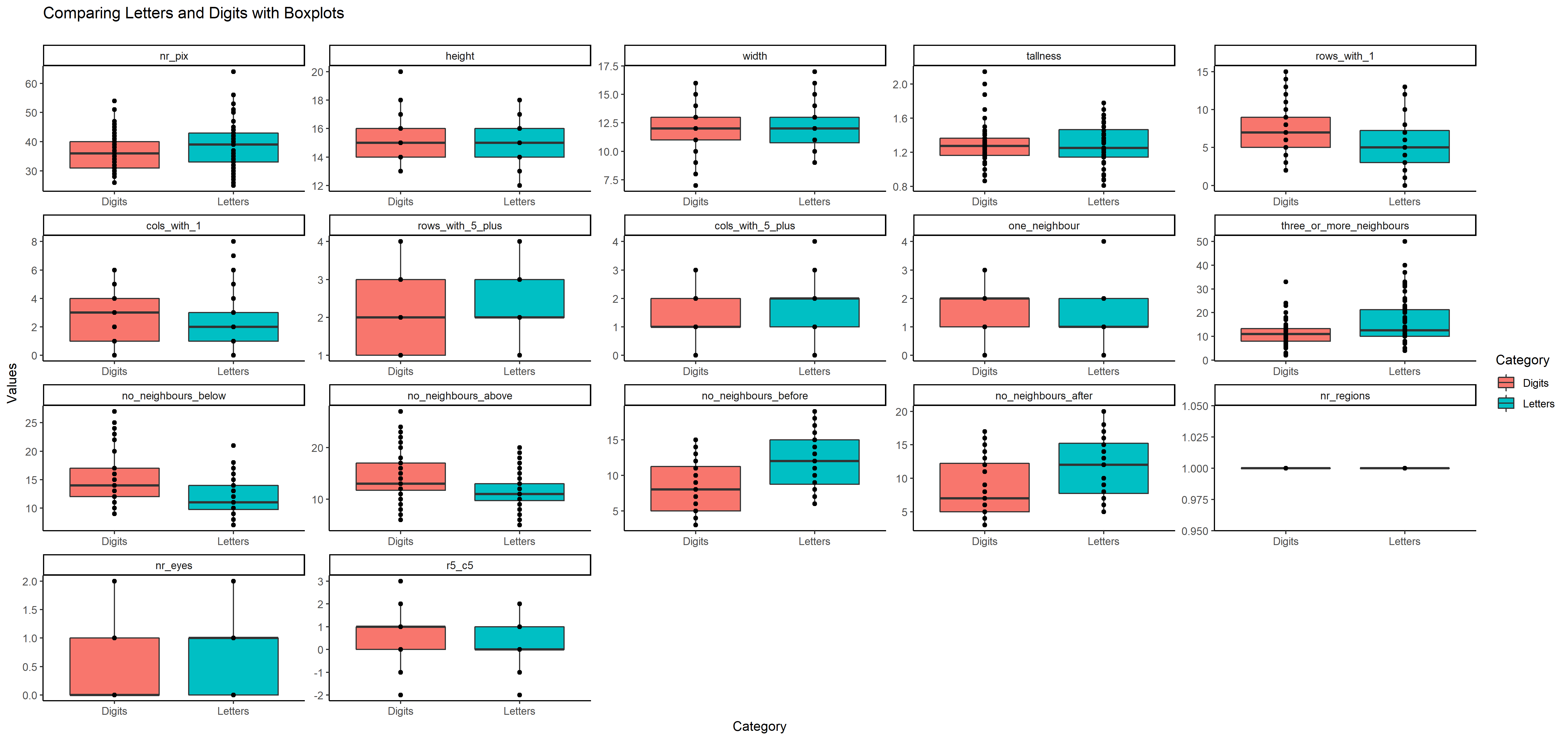
The hypothesis test for this ANOVA for each feature is as follows:

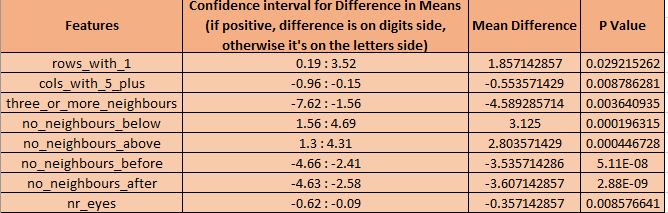
H0 = The mean value across each symbol group is the same

HA = The mean value for at least one pair of symbol groups is different

By this analysis all the features seem to be statistically significant in discriminating at least one group from the others for each feature. I suspect this is mostly to do with the difference of the Math dataset from the other two groups. The most significant values are again for the neighbour features, which was already discussed in section 3.3, to further what was said there, in the case of the symbols I believe that the math symbols are more horizontal in their orientation and less vertical than the other two sub groups of symbols and this may explain the difference. The findings of the ANOVA when it comes to the statistical significance of the r5\_c5 feature is worrying though, it does not seem by glance of the boxplot distribution to be the case. I imagine this is because the wide variance of the Math dataset has thrown the ANOVA test off.

## Section 3.8: Features that are useful for discriminating between the set of digits and the set of letters

As I was only comparing two groups this time, I found that a t-test was more suited to my needs, below is a boxplot that can help as a guide to comparing differences in the two groups alongside the p value table I got from my t-test:

The result of the t-test is below. As we can see, far less features were returned as statistically significant, this is as a result of less groups to compare and the digits and letters datasets are relatively similar for most features. Yet again the most statistically significant feature seems to be the pairing of the no\_neighbours\_before and the no\_neighbours\_after features – my estimation for why this is has already been explained before. The inclusion of the nr\_eyes being statistically significant is likely because there are possibly a couple of outlier in the letters dataset that have been erroneously drawn and couted as having two eyes and so skew our p value.

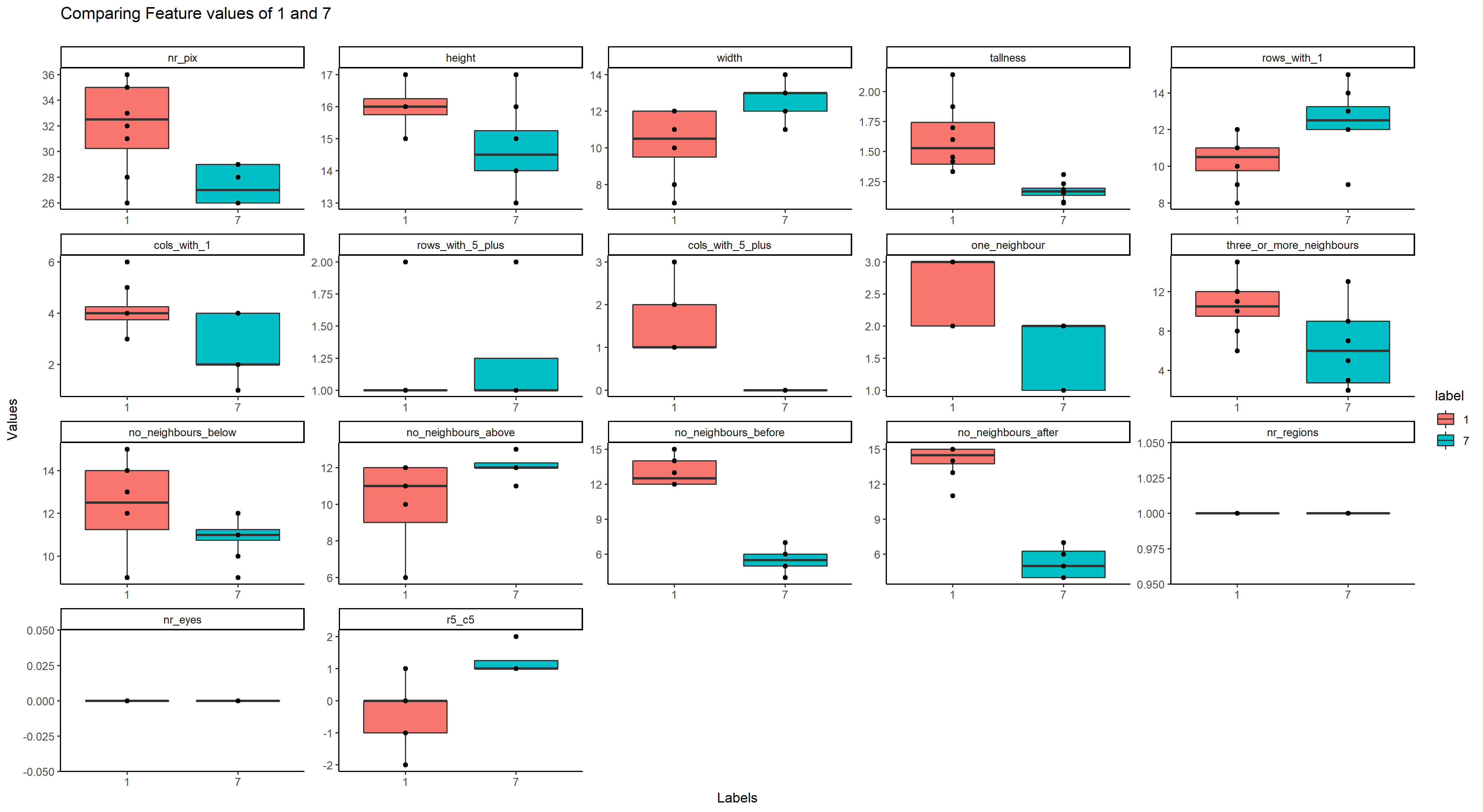
The hypothesis test for this t-test for each feature was as follows:

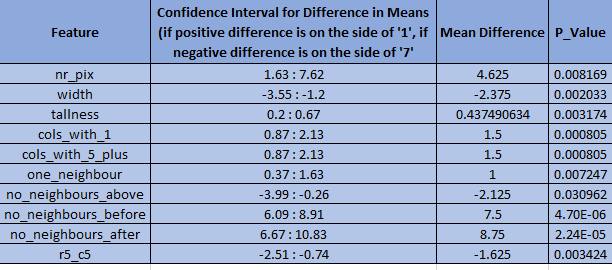
H0 = The mean value is the same for the Digits and Letters dataset

HA = The mean value is different for the Digit and letters dataset

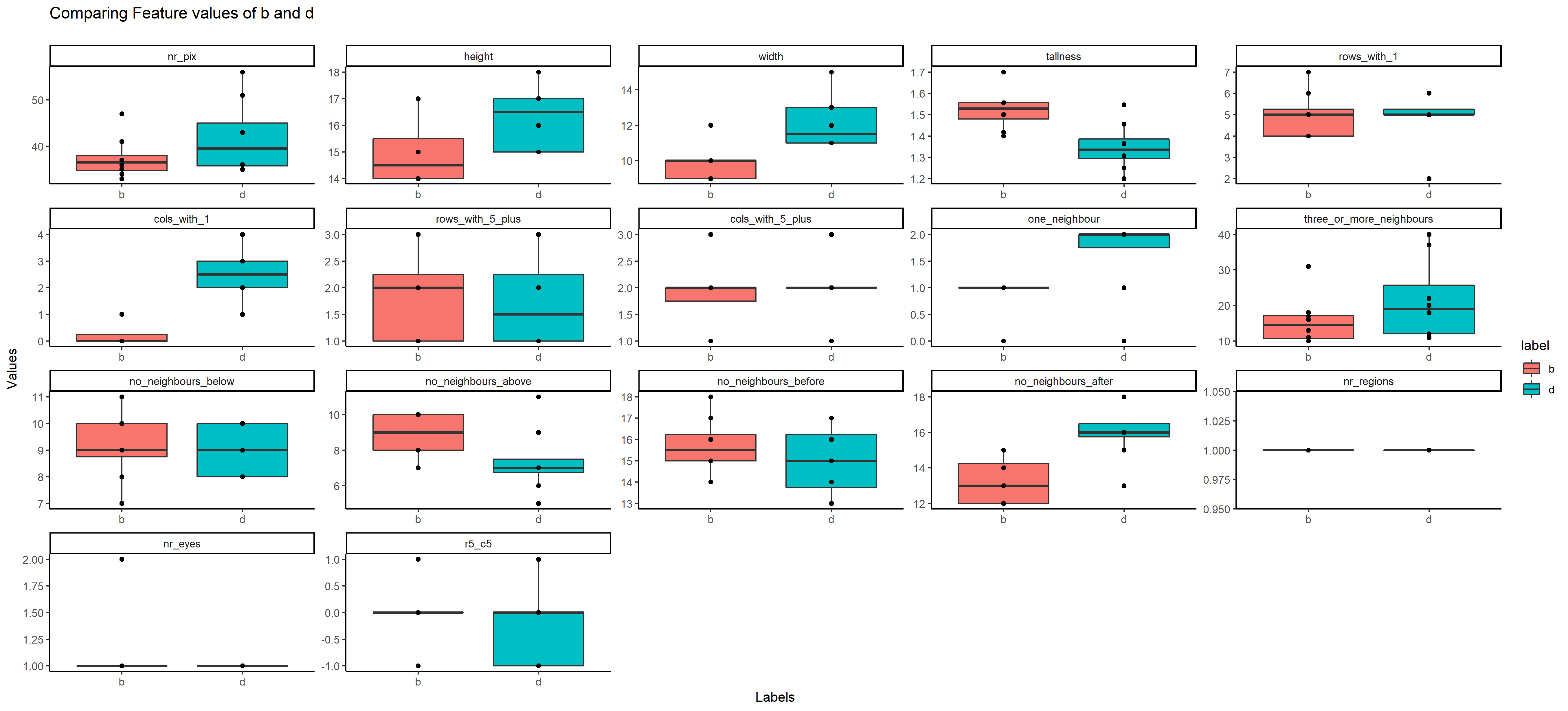
## Section 3.9: Features that are useful for discriminating between digit ‘1’ and the digit’7’

For this test, as well as the next three questions – I used a two-way t-test, as the questions only asked me to show a difference and not whether one symbol is greater or lesser in a feature than it’s counterpart. In this test as well as the next few t-tests I assumed no extreme skew in either group and an independence within and between groups. Unlike an ANOVA equal variability along groups is not necessary, however I still feel like a boxplot may be beneficial to have to corroborate the findings of my t-test:

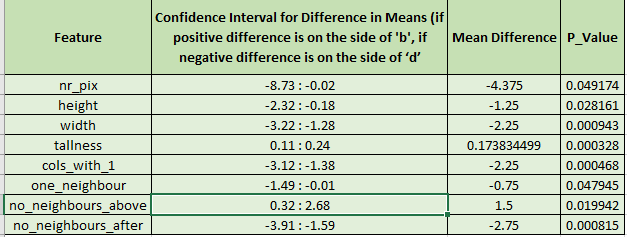


There seem to be a wide range of notable differences to choose from going by the boxplot above, although the significant features returned here are much less than before with the comparison of two groups – despite the differences seeming more notable by box-plot. Likely, this is a result of a smaller sample size having less degrees of freedom and so the t-distribution has a fatter tail. It treats outliers with more leniency than the previous tests we used. The digit 7 as a symbol, using the stats above as a guide, seems to be wider and have a more horizontal orientation – with the significance of the no\_neighbours\_before and no\_neighbours\_after features being prevalent. This is an expected result. What perhaps is not expected is the mean number of black pixels being more on the side of ‘1’ digit than the ‘7’ digit. Perhaps this is because of the way I draw my 1’s.

## Section 3.10: Features that are useful for discriminating between symbol ‘b’ and ‘d’



The differences between the symbols of b and d at a glance are far less pronounced than comparisons before. Below are the results of the t-test I performed on them

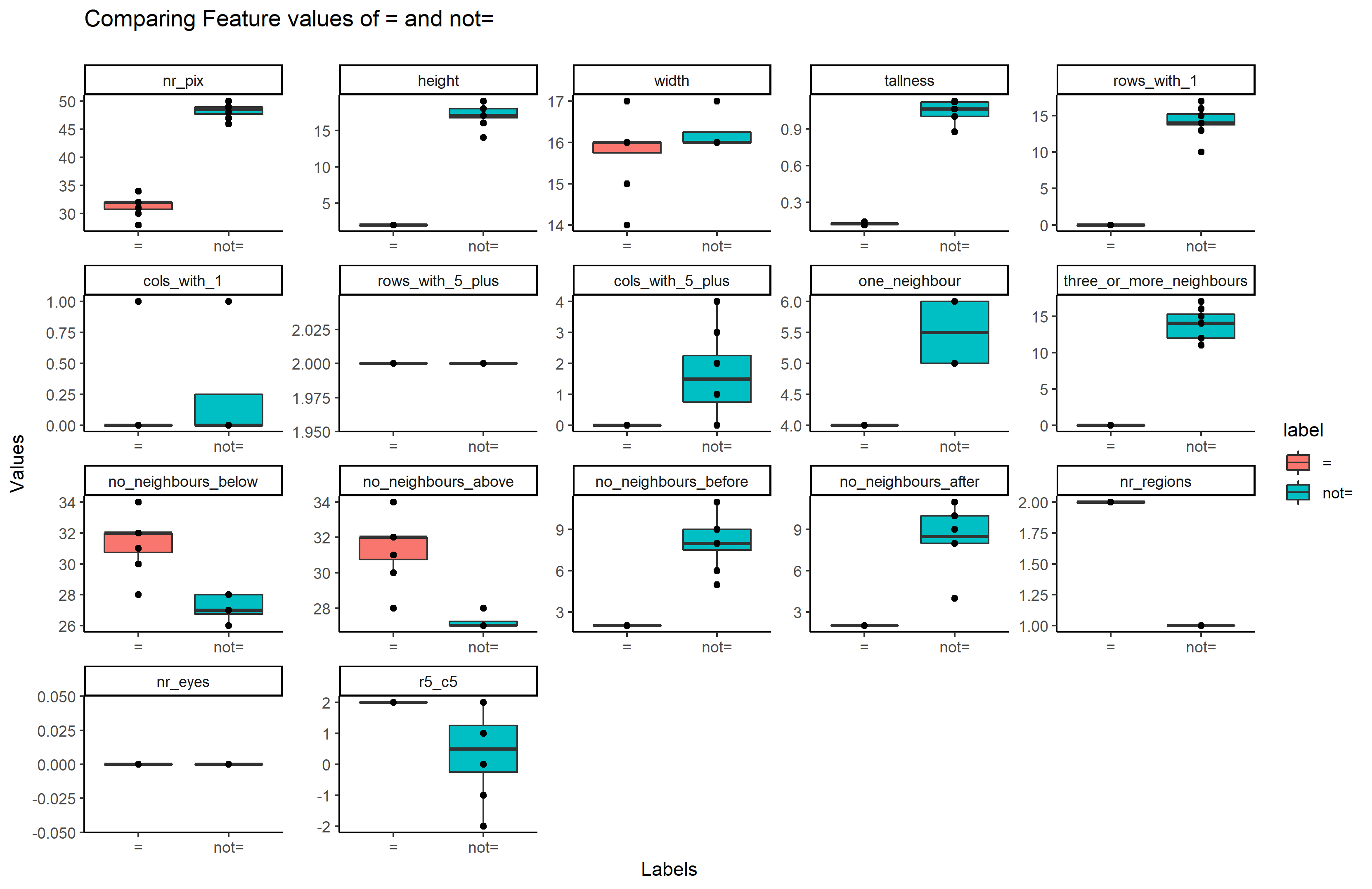


The symbol of ‘d’ seems to have a greater mean number of pixels than the b symbol. This may be related to the height difference too – I possibly draw my d’s on average taller than my b’s – this is further confirmed by the large difference the two symbols seem to have in the tallness features. - Another notable difference is the no\_neighbours\_after feature – which ‘d’ has more occurrences of – this is most likely related to the fact that the symbol ‘d’ has it’s vertical line on the right of the symbol and for each pixel in that vertical line it should increment the no\_neighbours\_after feature.

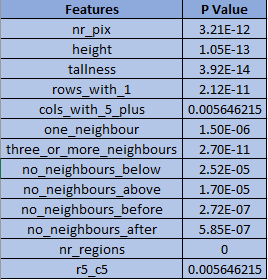
H0 = The mean value is the same for the ‘b’ sample and the ‘d’ sample

HA = The mean value is different for the ‘b’ sample and the ‘d’ sample

## Section 3.11: Features that are useful for discriminating between symbol ‘=’ and ‘≠’



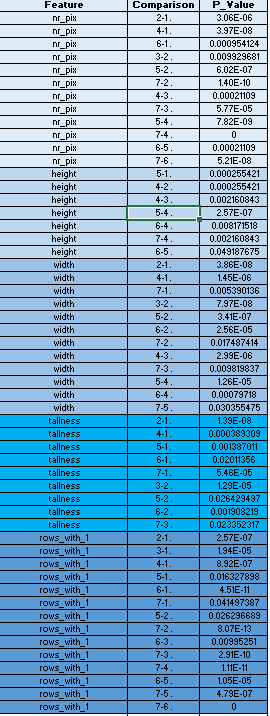
Unlike the ‘b’ and ‘d’ symbol, at a glance the ‘=’ and ‘≠’ symbols have pronounced differences. An interesting observation that can be made from the boxplot above is that the features that the samples in the ‘=’symbol group have possessed very little variance – they seem to be quite the same most of the time. Unfortunately, the standard t-test function would not work on these data and I could not diagnose the problem, so I had to use the pairwise.t.test function. Consequently, the information here is less than the information before. Anyways, below are the p-values from my dataset to illuminate further differences in the two symbols:

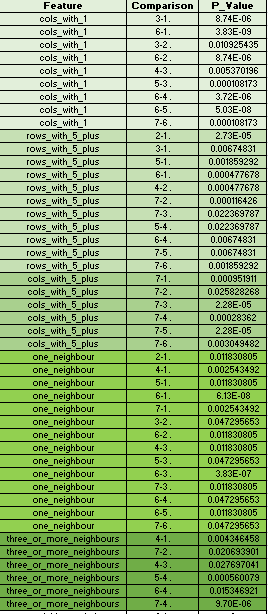
The major discriminator between the pair of symbols is the nr\_regions feature – where an = sign is always 2 regions, a ≠ is always one region. With this feature alone you could reliably distinguish one symbol from the other. Also prevalent is the difference in height, nr\_pix and tallness features which are as simply a result of ≠ being more massive shape on average. The rows\_with\_1 is also a telling feature as there are no rows\_with\_1 features being recorded on any of the handwritten = symbols in my sample. The hypothesis test for this two-way t-test for each feature was as follows:

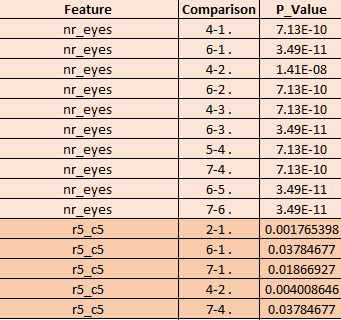
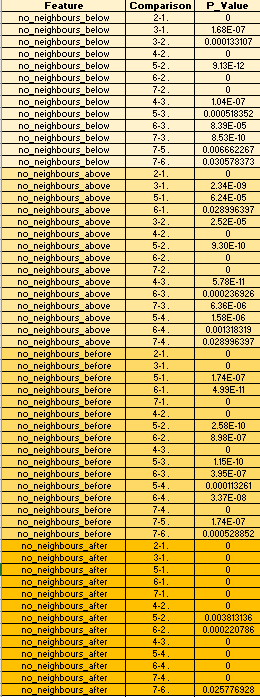
H0 = The mean value is the same for the ‘=’ sample and the ‘≠’ sample

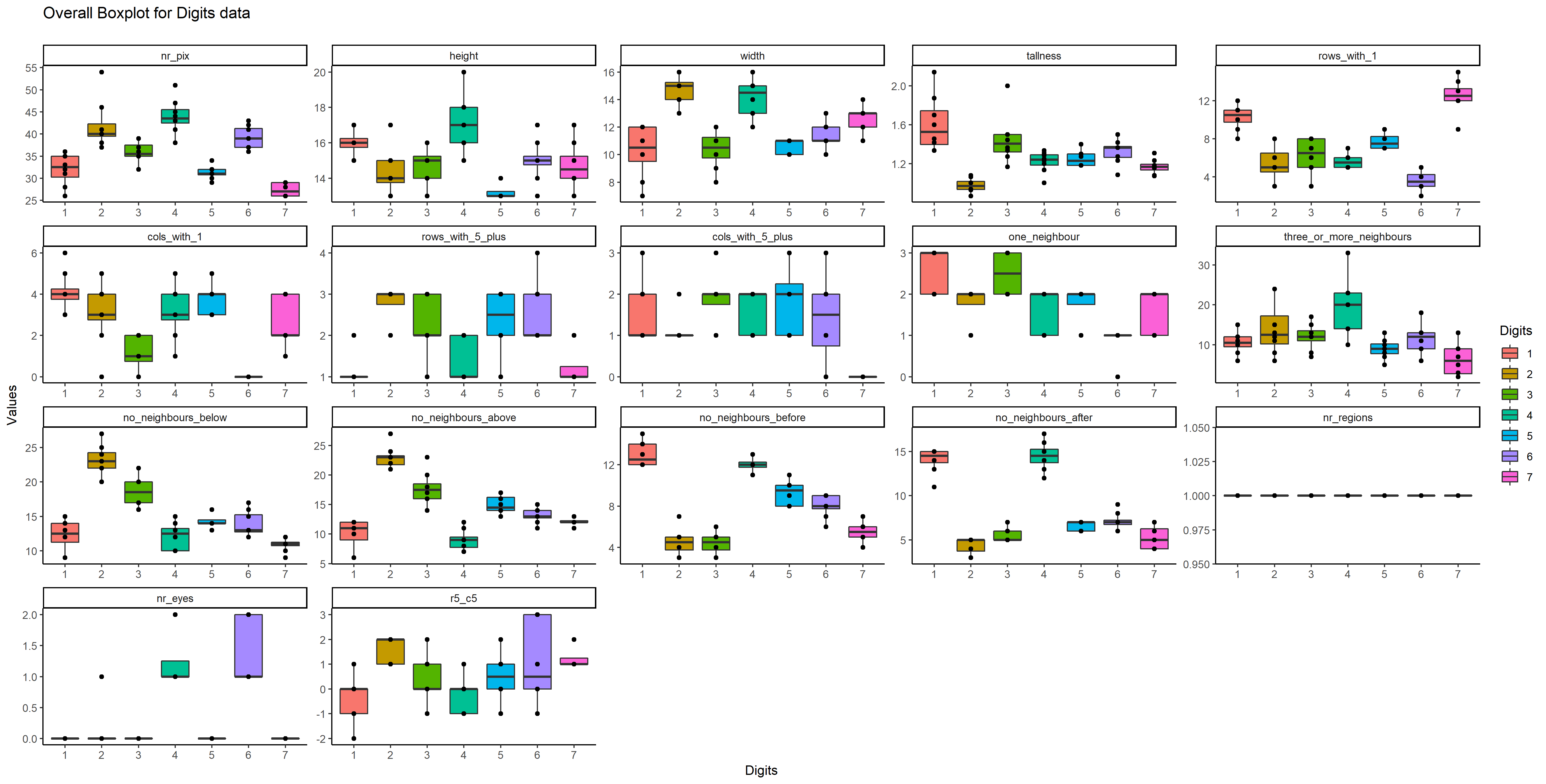
HA = The mean value is different for the ‘=’ sample and the ‘≠’ sample

## Section 3.12: For each feature, find the pairs of digits that have a statistically significant difference for that feature

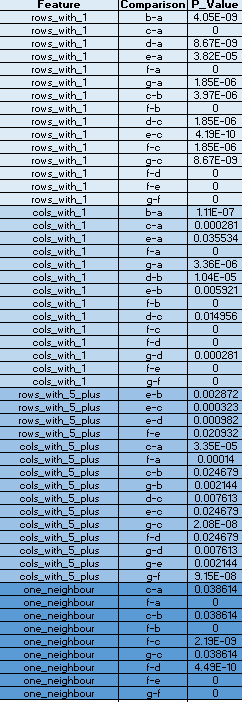
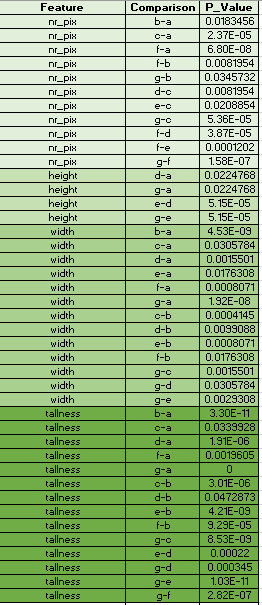
For the next three questions of this assignment, I was required to do multiple comparisons within each group of symbols in order to pinpoint where exactly the differences in means are for each feature. I had to be careful here as with multiple comparisons comes an increased probability in a type 1 error, so multiple comparison correction had to be made. For this reason, I chose to use a TukeyHSD post-hoc test, as through research it seemed more suitable to use when having a sample size with a substantial number of multiple comparisons, rather than the pairwise.t.test with bonferonni correction suggested in the lectures. Within the Tukey HSD function I made sure to specify a 98% family-wise confidence level to account for errors. Below are the results I acquired for each group of symbols along with a boxplot to verify the results.

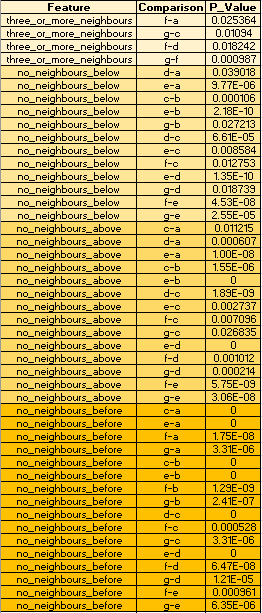
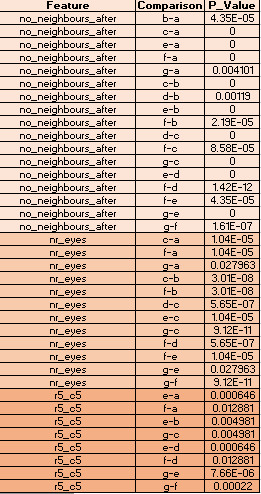


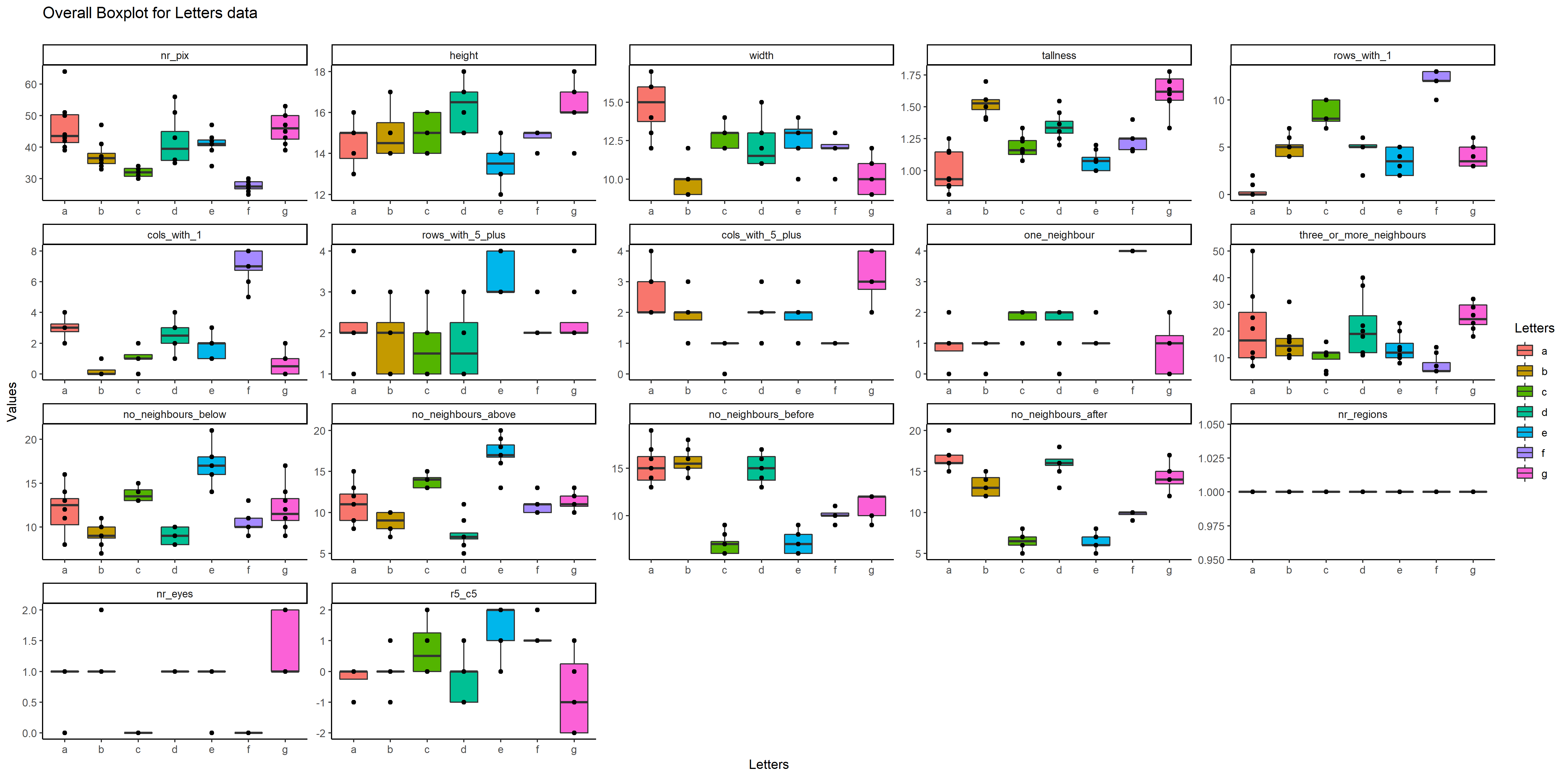




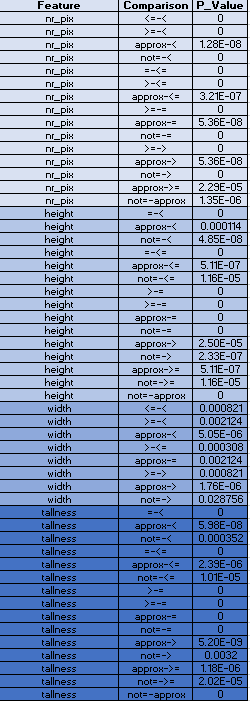
## Section 3.13: For each feature, find the pairs of letters that have a statistically significant difference for that feature

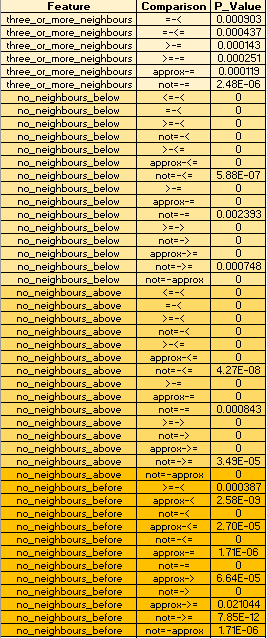
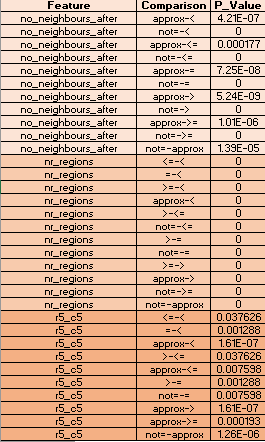


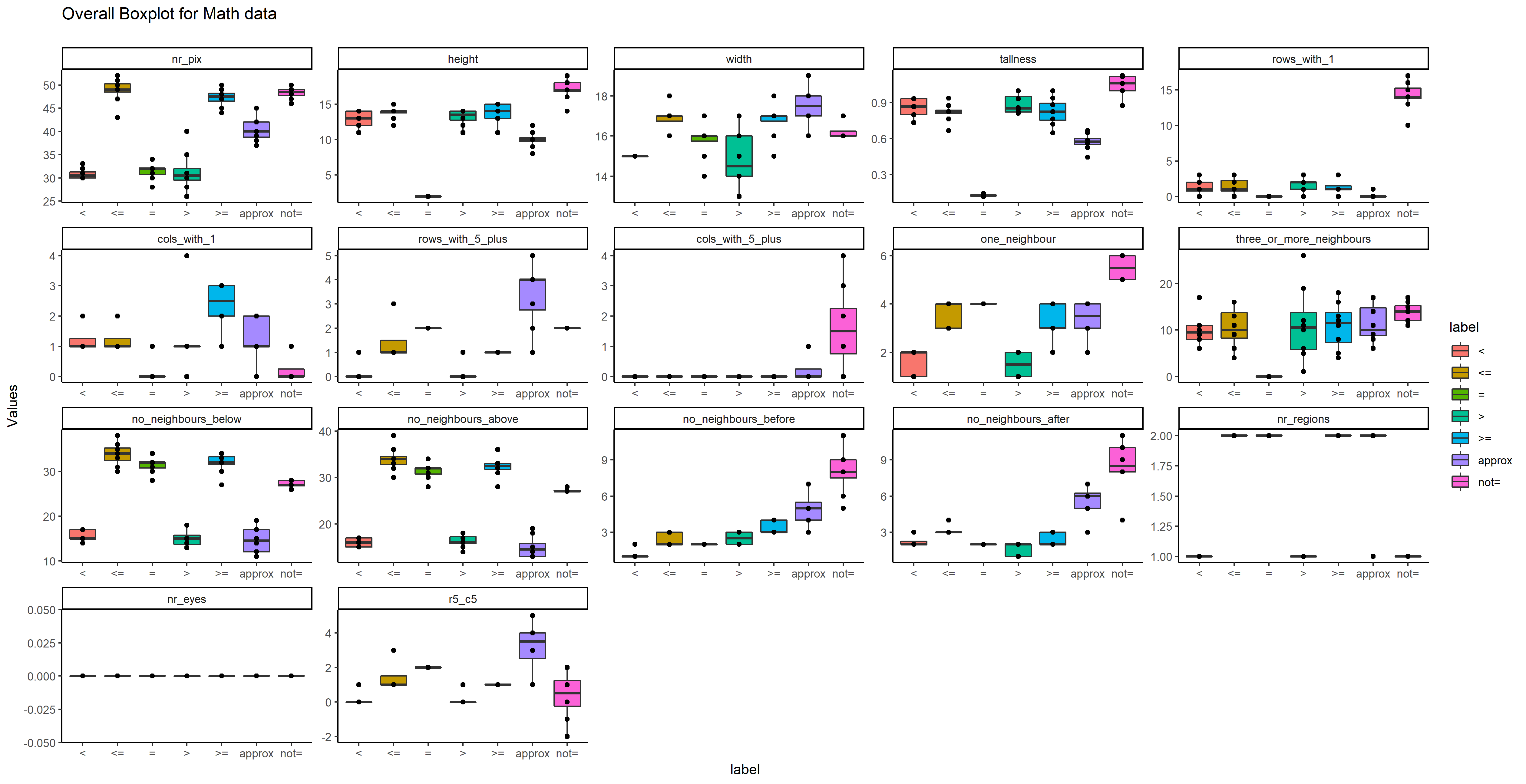




## Section 3.14: For each feature, find the pairs of Math symbols that have a statistically significant difference for that feature







**Conclusion:**

This investigation surprised me in that I assumed that nr\_eyes and nr\_regions would be by far the biggest distinguishers between symbols as they are categorical factors – a symbol either has an eye, or it doesn’t, a symbol either has 1 or 2 regions. However, this assumption was proved wrong, possibly because these features are so low in their variance that one outlier/error can throw of the entire variance of the feature data. The features that proved the most consistently important were the no neighbours before/after/above/below features as they seem the best at identifying the vertical/horizontal orientation of a symbol. If I had more time to do this project then perhaps I would attempt to create features based on identifying horizontal and vertical lines in a shape, using the code in these features as inspiration. Also, another regret I have is that I should have not assumed normality for each feature in each dataset – some of the feature data was extremely skewed in these smaller samples and I could have maybe more accurately measured the data if I had done a randomisation test for the F- scores and decided the statistical significance of the features that had skewed distributions from there. Another issue I have is with my code in section 3; I could have possibly used functions to simplify the code as much of the code to create the visualisations was often recycled, unfortunately I wasn’t familiar enough with R to be confident of going down such a route.