Classifying digits & letters

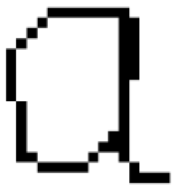
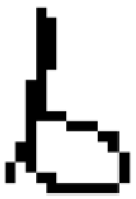
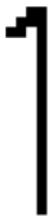
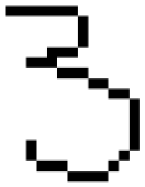
Zauad Shahreer Abeer

Introduction:

This project demonstrates *feature engineering, statistical analysis and machine learning* skills, all in one. The main aim is to classify digits (1, 3, 4, 5, 7) and letters (a, b, c, d, e)

1. **The Data:** For each letter/digit, there are 10 csv files, each csv file represents an image.

Each image is represented by a black & white matrix with size 16 rows by 16 columns. In the matrix, the number 1 represents black pixels and 0 represents white pixels. As such, one image can be stored in a csv file containing the matrix (and no headers), as in these examples:

Class	*a*	*b*	*1*	*3*
Example Image				
Image Matrix CSV file	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0 0,0,0,0,0,1,1,1,0,0,0,0,1,1,0,0 0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0 0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0 0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0 0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0 0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0 0,0,0,0,1,0,0,0,0,0,0,0,1,1,0,0 0,0,0,0,1,0,0,0,0,0,0,0,1,1,0,0 0,0,0,0,0,1,0,0,0,0,0,0,1,1,0,0 0,0,0,0,0,1,0,0,0,0,0,0,1,1,0,0 0,0,0,0,0,0,1,1,0,1,1,1,0,1,0,0 0,0,0,0,0,0,0,1,1,1,0,0,0,1,1,0 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0 0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0 0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0 0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0 0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0 0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0 0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0 0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0 0,0,0,1,1,0,0,1,1,1,0,0,0,0,0,0 0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0 0,0,1,0,1,0,0,0,0,0,0,1,1,0,0,0 0,0,1,0,1,0,0,0,0,0,0,1,1,0,0,0 0,0,1,0,0,1,1,1,0,0,0,1,0,0,0,0 0,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0 0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0 0,0,0,0,1,1,1,0,1,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0 0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0 0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0 0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0 0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0 0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0 0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0 0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0 0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0 0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0 0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0 0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

Feature Engineering

Using each 16x16 matrix obtained from an image as above, we create an array of characteristics that describe some features of the image. Each feature will be a number. There are 17 features in total. In the feature definitions that follow, a pixel has 8 neighbors, which will be referred to as follows:

upper-left	**upper**	**upper-right**
left	**[pixel]**	**right**
lower-left	**lower**	**lower-right**

The features are:

Feature Index	**Feature Short Name**	**Feature Description**
0	**label**	The true symbol in the image (represented by one of {1,3,5,7,a,b,d,e}). Note that the label is not a true feature, and should not be used as a feature during for statistical tests or during model training. You may, if you wish, encode the labels for 'a', 'b', 'c', 'd', and 'e' as 101, 102, 103, 104, and 105, respectively.
1	**nr_pix**	The number of black pixels in the image.
2	**height**	Number of rows containing at least one black pixel
3	**width**	Number of columns containing at least one black pixel
4	**tallness**	Ratio of height to width; i.e. feature 2 / divided by feature 3
5	**rows_with_1**	Number of rows with exactly one black pixel
6	**cols_with_1**	Number of columns with exactly one black pixel
7	**rows_with_5+**	Number of rows with five or more black pixels
8	**cols_with_5+**	Number of columns with five or more black pixels
9	**1neigh**	Number of black pixels with exactly 1 neighbouring pixel
10	**3+neigh**	Number of black pixels with 3 or more neighbours
11	**none_below**	Number of black pixels with no neighbours in the lower-left, lower, or lower-right positions
12	**none_above**	Number of black pixels with no neighbours in the upper-left, upper, or upper-right positions
13	**none_before**	Number of black pixels with no neighbours in the upper-left, left, or lower-left positions
14	**none_after**	Number of black pixels with no neighbours in the upper-right, right, or lower-right positions
15	**nr_eyes**	In a written character, an "eye" is a region of whitespace that is completely surrounded by lines of the character. For example, "A" contains one eye, "B" contains two eyes, and "C" contains no eyes. A region of whitespace is an eye if there is a ring of black pixels surrounding it which are all connected (i.e. they form a chain of neighbours). This feature is the number of eyes in the image.
16	**bd**	Design your own feature which you think may be useful for distinguishing between "b" and "d" character images.
17	**custom**	Design any other feature you like, which you think may be useful for distinguishing between characters.

This is done in **Python**.

A list is used to store all the images. Each element of the list is a numpy ndarray. Each ndarray contains 10 16×16 images. This made the calculations rather easy to perform as all the images could be handled at once.

The first few features are pretty straightforward and calculated using numpy functions. The logic of neighbours of pixels was a bit tricky and it took some thinking to code it. Nonetheless the features involving neighbours have been calculated perfectly. One feature, namely *nr_eyes* is not calculated. It is rather hard to express the logic in Pythonic terms and so a random number from 1 to 5 is used as instructed.

The feature *bd* is calculated in the following way. At first the column with the highest number of black pixels is obtained. This represents the **|** of b and d. Then, we take the difference of the total black pixels to the right of that column with the total black pixels to the left of the column. So, in case of *'b'*, this feature would be a positive value and in case of *'d'* it would be negative. This will hopefully be a good feature to distinguish between b and d.
The custom feature is sparsity, the product of height and width divided by the total number of black pixels. Some characters, like 1, will be less sparse. On the other hand, 7 should be more sparse. So this feature should be able to distinguish between these two. Combinded with the other features, this should work quite well.

Statistical Analysis

In this section, we will perform statistical analyses of the feature data, in order to explore which features are important for distinguishing between different kinds of characters. It is done in **R**.

3.1

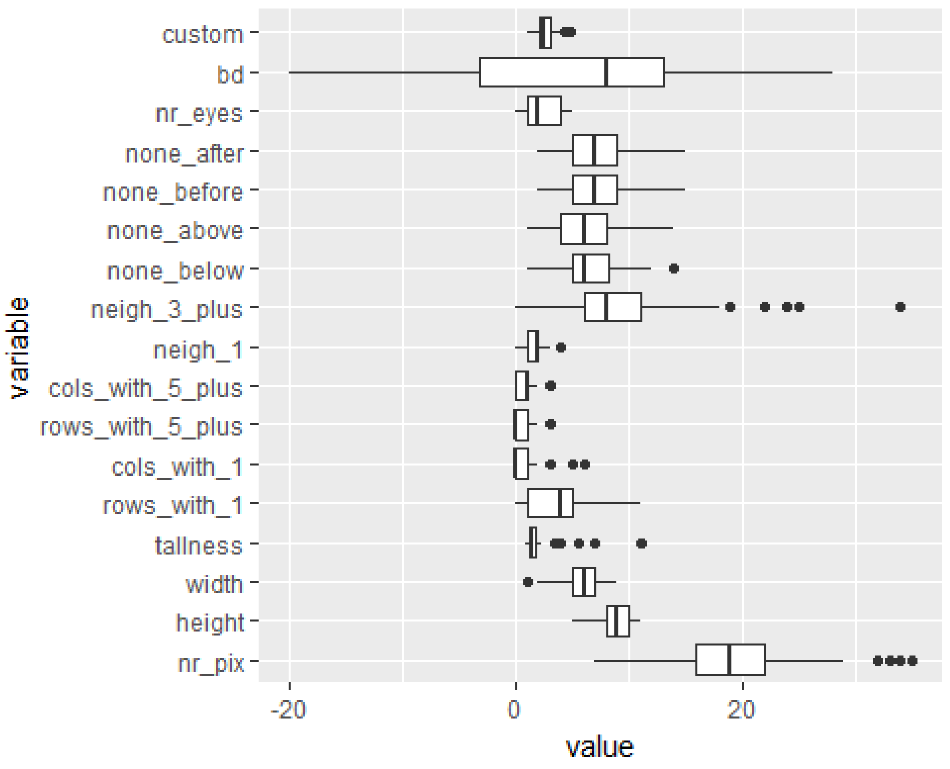
At first, we have a look at the descriptive statistics of all the features.

Table 1: Descriptive Statistics of the features

feature	mean	sd
nr_pix	19.66	5.119
height	8.75	1.5
width	5.78	1.495
tallness	1.741	1.251
rows_with_1	3.8	2.782
cols_with_1	0.84	1.308
rows_with_5_plus	0.48	0.7032
cols_with_5_plus	0.74	0.6908
neigh_1	1.6	1.064
neigh_3_plus	8.98	5.961
none_below	6.39	2.752
none_above	6.12	3.173
none_before	7.17	2.948
none_after	7.11	2.689
nr_eyes	2.48	1.673
bd	5.64	10.52
custom	2.598	0.7241

It is seen that the average number of black pixels is about 20 but varies greatly. Let us have a look at the boxplots to understand the distribution of the features a bit more clearly.

Figure 1: Boxplot of the features



The descriptive statistics and boxplots for the letters and digits seperated should reveal some interesting information.

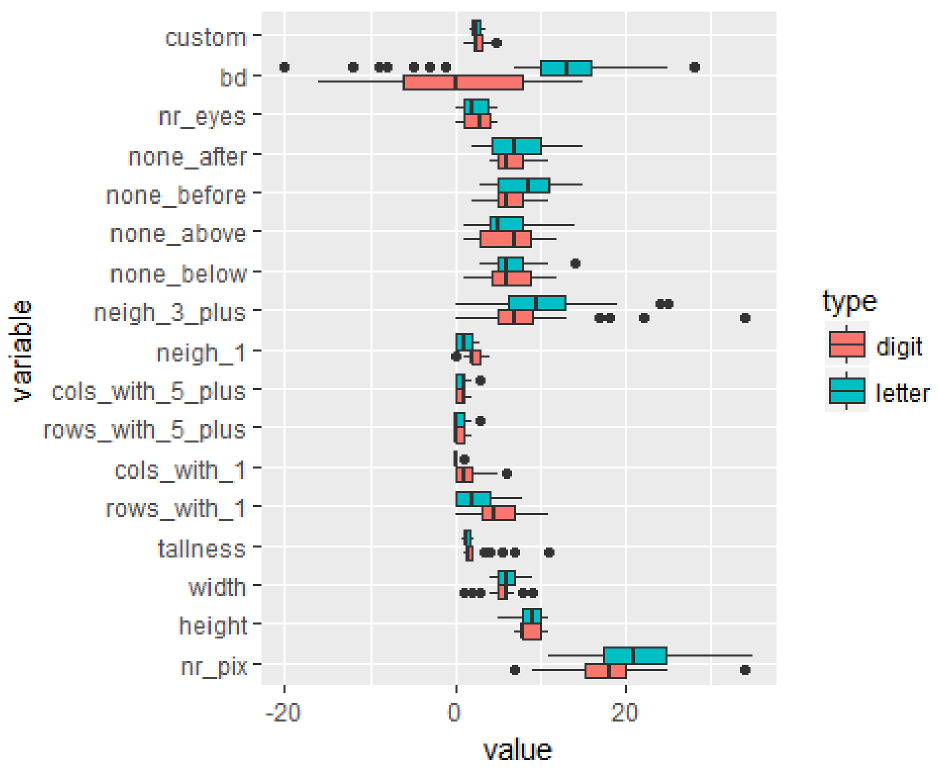
Table 2: Descriptive Statistics by digits and letters

feature	mean_digits	sd_digits	mean_letters	sd_letters
bd	0.56	8.814	10.72	9.664
cols_with_1	1.5	1.555	0.18	0.3881
cols_with_5_plus	0.66	0.5928	0.82	0.7743
custom	2.72	0.8717	2.476	0.5187
height	8.88	1.394	8.62	1.602
neigh_1	2.06	0.9564	1.14	0.9691
neigh_3_plus	8	5.785	9.96	6.03
none_above	6.14	3.289	6.1	3.086
none_after	6.78	2.102	7.44	3.157
none_before	6.26	2.146	8.08	3.355
none_below	6.18	2.768	6.6	2.748
nr_eyes	2.68	1.647	2.28	1.691
nr_pix	17.82	4.346	21.5	5.211

rows_with_1	5.2	2.483	2.4	2.339
rows_with_5_plus	0.52	0.6465	0.44	0.7602
tallness	2.024	1.692	1.458	0.3705
width	5.48	1.776	6.08	1.085

A great difference between the letters and digits can be seen in case of the feature *bd*. There are also some notable differences among *number of pixels, neigbours with 3 plus* etc.

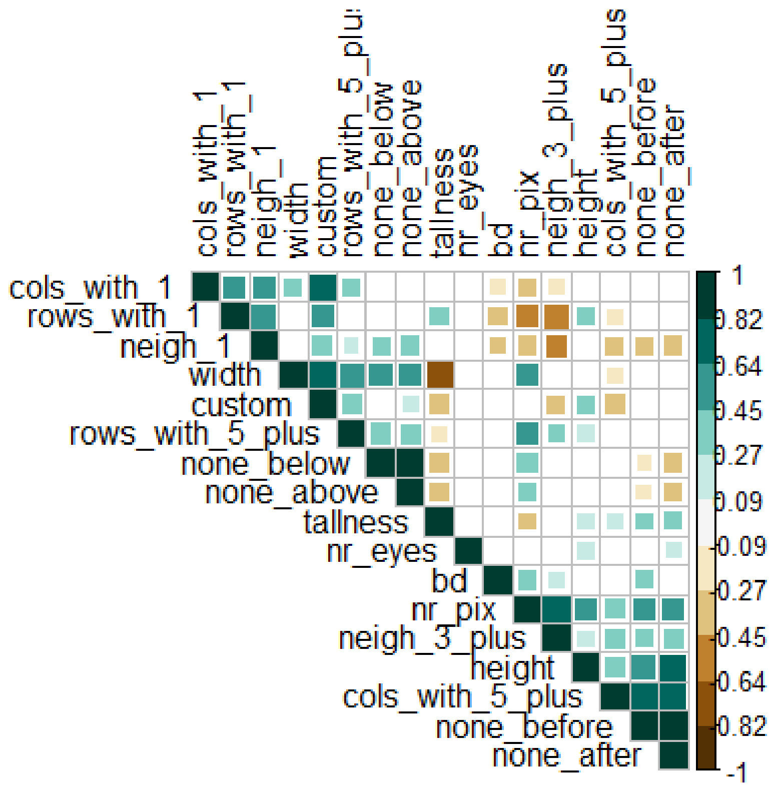
Figure 2: Boxplots by type (digit/ letter)



3.2

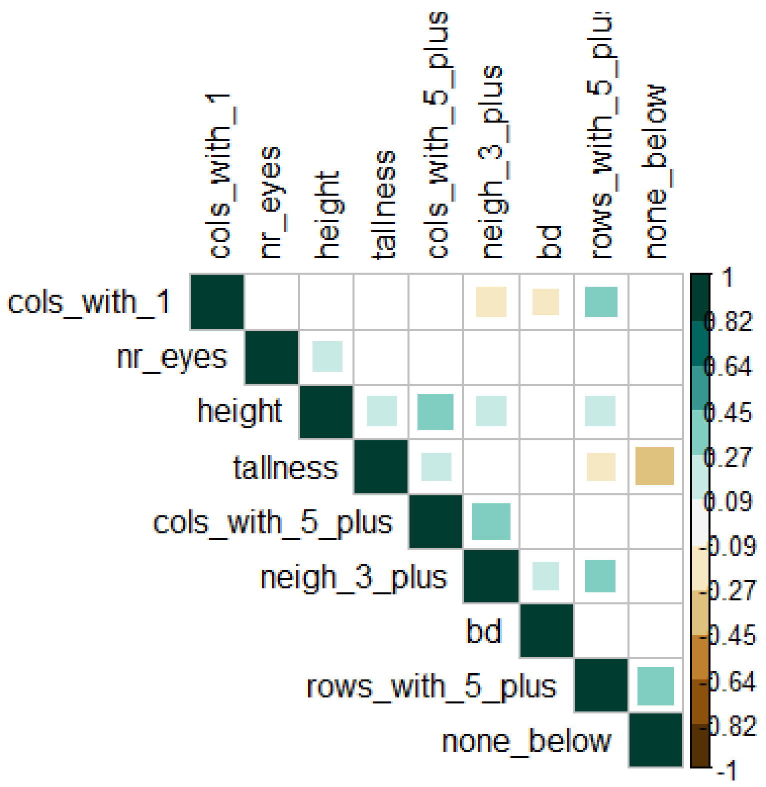
The plot of correlation matrix shows the correlations of the variables with each other.

Figure 3: Correlation plot of the features.



It is observed that there are some highly correlated features. The correlation matrix is showing only the significant ones. So, there is a strong significant correlation between (width & tallness), (none_before & none_after), (height and none_after) and some other pairs as well. Correlated variables will introduce multicollinearity. If two variables are highly correlated, it is a burden to keep both of them in the analysis. So we remove feature that are highly correlated with others or have a moderate correlation with some of the features. In such a case, the omitted features are *nr_pix*, *width*, *rows_with_1*, *neigh_1*, *none_above*, *none_before*, *none_after and custom*. The rest of the analysis will be carried on without these features. Let's now have a look at the correlation matrix.

Figure 4: Correlation plot of the reduced data set



So the correlations are weak among these features. These features can now be used to build models in order to do classification.

3.3

We now look for features that can identify the different letters i.e. the featues will have different values for different letters. For example, let us consider the feature *height*.

Table 3: Mean heights of the letters

letter	height
a	8.5
b	9.3
c	7.3
d	9.6
e	8.4

It is observed that *'a'* has a mean height of 8.5, *'b'* has mean height of 9.3 and so on. We want to find out whether the difference between the heights of the letters is significant or is it just by chance. For this purpose, we may use one-way ANOVA since there are more than two groups to compare.

We are testing the hypothesis:

H_0: There is no difference in the mean heights of the letters vs
H_1: There is difference among the mean heights of the letters

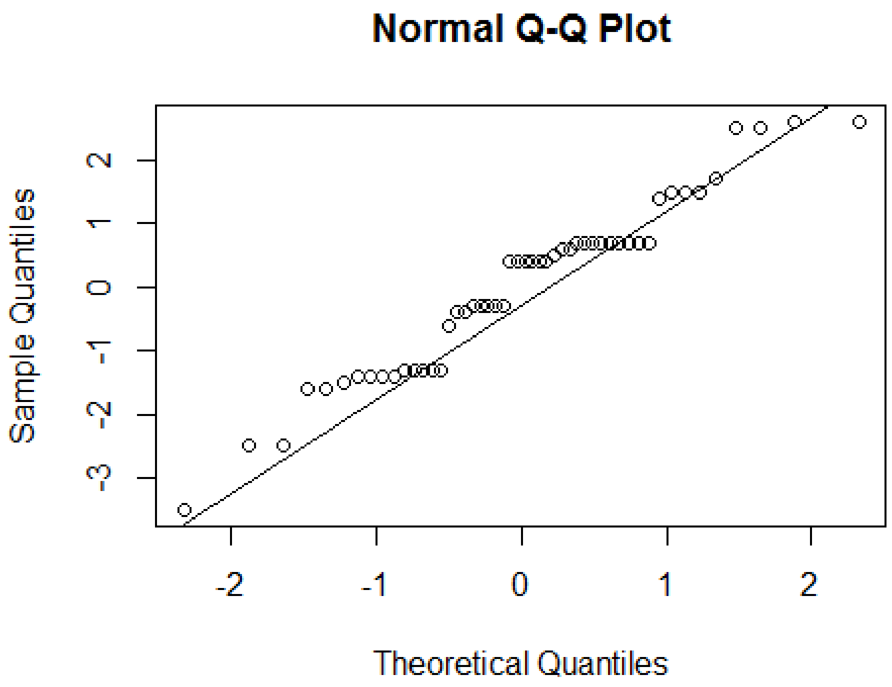
In order to use ANOVA it needs to satisfy some assumptions,

1. Samples are independent.
2. Experimental errors i.e residuals are normally distributed.
3. Homogeneity of variance, homoscedasticity.

We need to test whether these assumptions are met before using the test.

The first assumption can be understood intuitively, the samples are independent, there is no dependency of ‘a’ with ‘b’. We need to draw a qqplot to check whether the distribution of the residuals is normal to check the second assumption.

Figure 5: Normal Q-Q plot of the residuals



The more points on the line, the more normal the distribution. The qqplot suggests that the distribution of the residuals may not be normal. We need to perform a test, namely *Shapiro-Wilk’s normality test* to test for normality.

Table 4: Shapiro-Wilk normality test

	statistic	**p.value**
W	0.9602	0.09053

The test gives a p-value of .09 which indicates that the distribution of the residuals can be considered normal at the 5% level. The third assumption can be checked using the *Levene's Test for homogeneity of variance(based on median)*

Table 5: Levene's Test for Homogeneity of Variance

	Df	F.value	Pr..F.
group	4	2.984	0.02871
	45	NA	NA

This test gives a p-value of .03 indicating that the variance is not homogeneous among the samples and thus the assumption of homogeneity of ANOVA is violated. We cannot perform ANOVA if any one of the assumptions is violated. An alternative to ANOVA is the *Kruskal-Wallis rank sum test* which is a non-parametric test and thus it doesn't have the drawback of assumptions like in ANOVA. We will use this test.

Table 6: Kruskal-Wallis rank sum test

	statistic	parameter	p.value
Kruskal-Wallis chi-squared	13.14	4	0.01059

This gives the p-value of .011 and thus we may reject the null hypothesis and conclude that there is significant difference between the mean heights of the letters.

Table 7: Tests of normality and homogeneity for each feature

feature	test.rejected
nr_pix	Levene Test of homogeneity
height	Levene Test of homogeneity
width	Shapiro-Wilk's normality
tallness	None
rows_with_1	Shapiro-Wilk's normality
cols_with_1	Shapiro-Wilk's normality
rows_with_5_plus	Shapiro-Wilk's normality
cols_with_5_plus	Shapiro-Wilk's normality
neigh_1	Shapiro-Wilk's normality
neigh_3_plus	Shapiro-Wilk's normality
none_below	Shapiro-Wilk's normality
none_above	Shapiro-Wilk's normality
none_before	None
none_after	None
nr_eyes	Shapiro-Wilk's normality

bd	both
custom	None

It is observed that all but 4 of the features get rejected in either or both of the tests. Thus the non-parametric *Kruskal-Wallis test* is used to check for differences among the letters. It is summarized in the following table. We are testing:

H_0: There is no difference among the values of feature$_i$ for the five letters vs
H_1: There is difference among the values

Table 8: Significant features to distinguish among letters

feature	**sig**
nr_pix	0.0003076
height	0.01059
width	0.002257
tallness	1.663e-05
rows_with_1	0.005028
cols_with_5_plus	0.0002808
neigh_1	7.201e-05
neigh_3_plus	5.171e-05
none_below	1.316e-05
none_above	1.807e-05
none_before	0.0001875
none_after	1.382e-06
custom	0.01218

So we have 13 features that have significant differences and may be used to identify the letters.

3.4

We perform a similar test as in the previous section just with digits instead of letters. So our hypothesis is:

H_0: There is no difference among the values of feature$_i$ for the five digits vs
H_1: There is difference among the values

Table 9: Significant features to distiguish among digits

feature	**sig**
height	0.0001145
width	1.015e-05
tallness	2.918e-05
rows_with_1	3.352e-05
cols_with_1	2.72e-06

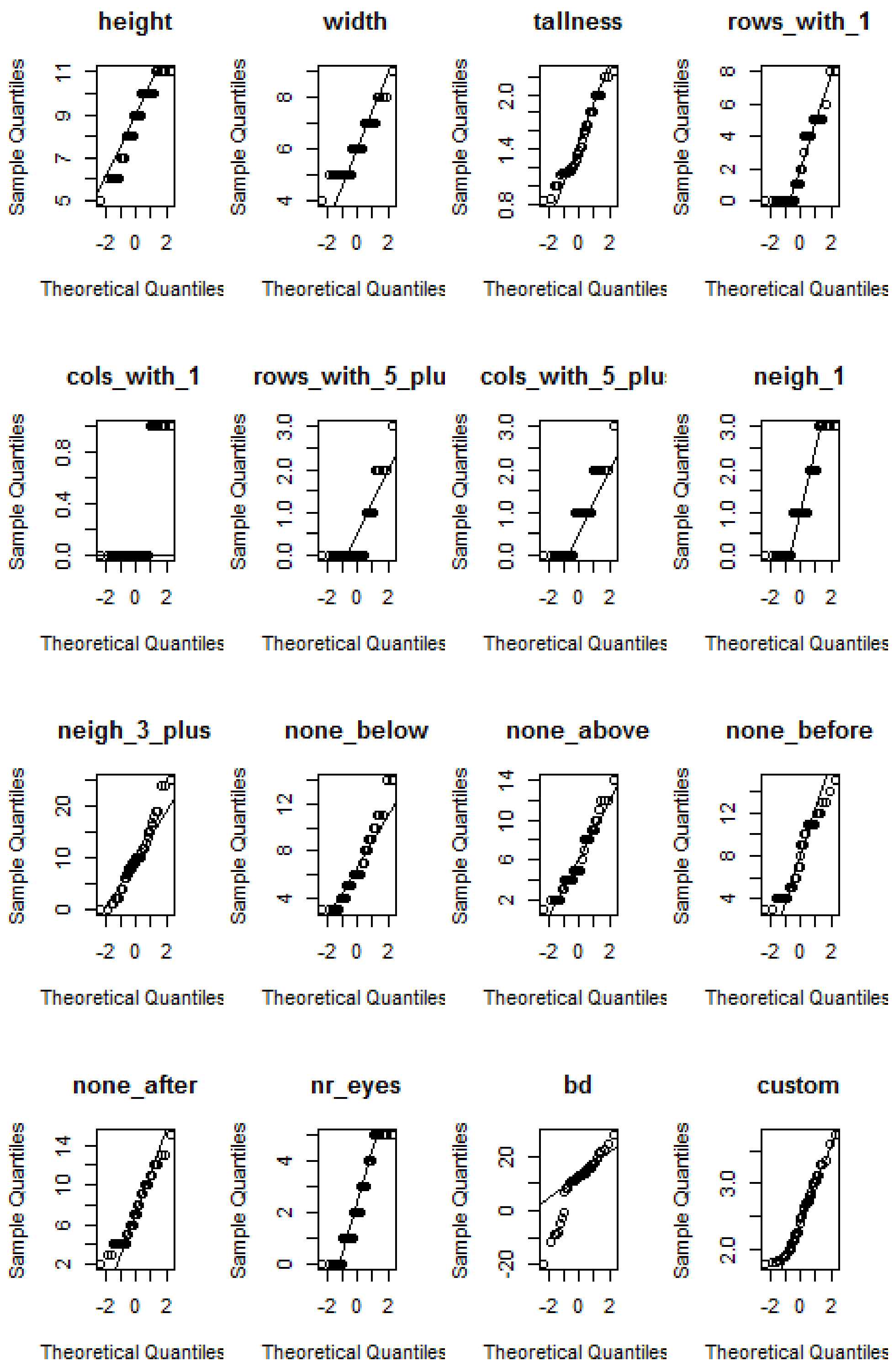
rows_with_5_plus	0.0002574
cols_with_5_plus	0.005212
neigh_1	0.005569
none_below	6.616e-06
none_above	5.208e-07
none_before	4.068e-06
none_after	3.941e-06
bd	0.001039
custom	6.597e-06

We have 14 features with significant differences that can be used to build models to identify the digits seperately.

3.5

In this case we are to perform a two sample comparison between digits and letters for each of the features. We are trying to observe if there is any difference in the features of digits and letters. This could have been accomplished by the two-sample t-test, but there is the assumption of normality. Let us the check the normality of the features of the letters using qqplot.

Figure 6: Normal Q-Q plots of all the features



Most of the features deviate from the assumption of normality. Therefore, the *Mann-Whitney U test*, a non-parametric counterpart of the two sample t-test will be used. This test holds even when the assumption of normality is violated.

The hypothesis we are testing is:

H_0: There is no difference between the features of digits and letters vs
H_1: There is difference between the features

Table 10: Significant features to distiguish between digits and letters

feature	sig
nr_pix	0.0004071
tallness	0.02843
rows_with_1	3.27e-06
cols_with_1	1.368e-07
neigh_1	1.151e-05
neigh_3_plus	0.02797
none_before	0.01037
bd	4.788e-08

There are 8 features, *bd* being the most significant.

3.6

We treat the two digits ‘1’ and ‘7’ as two samples and perform a two-sample test, the *Mann-Whitney test*.

Table 11: Features to differentiate between 1 and 7

feature	sig
width	0.0002206
tallness	0.0001707
cols_with_1	0.001458
rows_with_5_plus	0.0008584
cols_with_5_plus	0.001443
neigh_1	0.01184
none_below	0.005871
none_above	0.0001941
none_before	0.0028
none_after	0.001877
custom	0.0003197

So we see a large number of features to differentiate between ‘1’ and ‘7’.

3.7

A similar test is performed for ‘b’ and ‘d’.

Table 12: Features to differentiate between b and d

feature	sig
none_below	0.002218
bd	0.03699

Only two features have a significant difference among ‘b’ and ‘d’. One of which is the custom feature *bd*.

3.8

It is required to test the pairs of digits that have a significant difference for each feature. At first, we need to perform *Kruskal-Wallis rank sum test* to check whether there are differences among the digits for a particular feature(done in section 3.4). If the test is rejected, then we will do multiple comparisons using *Mann-Whitney U test* keeping in mind the correction for multiple comparisons. We will be using the *Bonferroni* adjustment in this case. For example, let’s consider the feature height. We know from 3.4 that the digits differ in case of height. But we don’t know which digits differ. So the following table gives the pairwise test result:

Table 13: Pairwise comparisons using Mann Whitney U test for the feature height

	1	3	4	5
3	0.1642	NA	NA	NA
4	0.04266	0.5575	NA	NA
5	0.03494	0.5544	1	NA
7	1	0.1218	0.01204	0.01291

So we observe that the pairs *(1, 4), (1, 5), (4, 7), (5, 7)* are significant, since the p-value is less than .05 Thus these pairs differ in case of height.

This is done for all the features and the results are printed in a single table as given below. Note that only the significant pairs for each feature are shown in the table.

We are testing the hypotheses:

H_0: There is no difference between the pairs of digits for feature_i vs
H_1: There is difference between paris of digits for feature_i

Table 14: Pairs of digits that have significant differences for each feature

feature	pairs	feature	pairs

height	(1 4)	none_below	(1 3)
	(1 5)		(1 5)
	(4 7)		(3 4)
	(5 7)		(4 5)
width	(1 3)		(4 7)
	(1 7)	none_above	(1 3)
	(3 7)		(1 5)
	(5 7)		(1 7)
tallness	(1 3)		(3 4)
	(1 4)		(4 5)
	(1 5)		(4 7)
	(1 7)	none_before	(1 3)
rows_with_1	(3 7)		(1 5)
	(4 7)		(1 7)
	(5 7)		(3 4)
cols_with_1	(1 4)		(3 5)
	(1 7)		(3 7)
	(3 4)	none_after	(1 3)
	(3 7)		(1 4)
	(4 5)		(1 5)

	(5 7)		(1 7)
rows_with_5_plus	(1 7)		(4 5)
	(3 7)	bd	(3 5)
	(5 7)		(3 7)
cols_with_5_plus	(1 3)	custom	(1 7)
	(1 7)		(3 7)
			(4 7)
			(5 7)

So let’s take an example. For the *height* feature, significant differences are seen between digits *1 and 4*, *1 and 5*, *3 and 7*, and *4 and 7*. The rest of the table is to be interpreted in a similar fashion.

3.9

A similar test as above is performed in case of the letters as well. The hypothesis stays almost the same, we are doing this for the letters in place of the digits. The results are as follows:

Table 15: Pairs of letters for that have significant difference for each feature

feature	**pairs**	**feature**	**pairs**
nr_pix	(a c)	neigh_3_plus	(a c)
	(b c)		(a e)
	(c d)		(b c)
	(c e)		(b e)

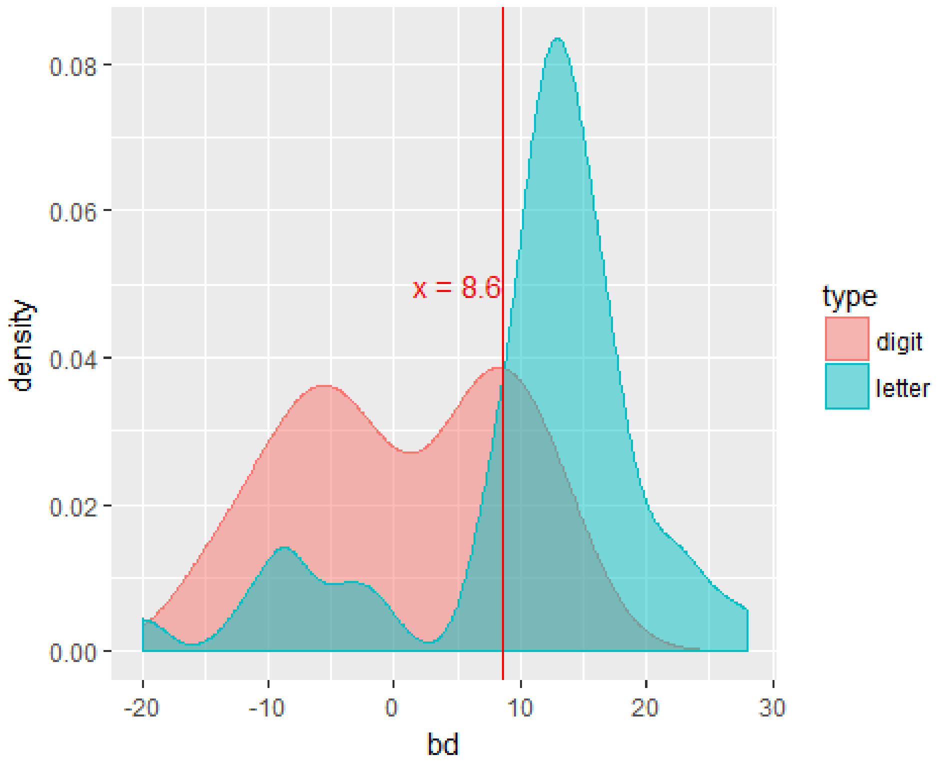
height	(b c)		(c d)
	(c d)	none_below	(a e)
width	(a b)		(b d)
	(a d)		(c d)
tallness	(a b)		(d e)
	(a d)	none_above	(a e)
	(b c)		(b e)
	(c d)		(c d)
rows_with_1	(a c)		(c e)
	(a e)		(d e)
cols_with_5_plus	(b c)	none_before	(b c)
	(b e)		(c d)
	(c d)	none_after	(a c)
	(d e)		(a e)
neigh_1	(b c)		(b c)
	(b e)		(b e)
	(c d)		(c d)
	(d e)		(d e)
		custom	(b c)

So the letters *b, c* and *c, d* differ in case of height, which is pretty intuitive.

3.10

The best feature for distinguishing between the letters and digits is *bd* as observed in section 3.5

Figure 7: Density plots of digits and letters for the feature bd



The above figure is a density plot showing the density of feature bd for the digits and letters simultaneously. The two densities intersect at $bd = 8.6$
We may select this intersecting point as the threshold to predict digits and letters. Any character having a bd value less than 8.6 would be classified as a digit and any character with value 8.6 or greater would be a letter. Since the two densities are somewhat divided at the intersecting point, so this value is chosen for the purpose.
Let us see how this classification performs.

Table 16: Confusion Matrix of classification based on bd

	digit	**letter**
digit	39	12
letter	11	38

We observe that out of the 50 digits, our algorithm classified 39 correctly and out of 50 letters, it classified 38 correctly. This gives an accuracy score of $(39 + 38)/100 = .77$ or 77%!
This is quite good given the pure simplicity of the model.

Section 4: Machine Learning

In this section we will use the features developed and analysed in the previous sections to solve classification problems. We will fit classifiers to the image data in order to build and evaluate models that can predict the classes.

4.1

The best feature we selected is *bd*. A logistic regeression model is fit in order to categorize digits and letters based on the feature bd.
The model is as follows:

```
$call
glm(formula = type ~ bd, family = binomial(link = "logit"), data = data)
```

Table 17: Coefficients of the model

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-0.6975	0.291	-2.397	0.01652
bd	0.1148	0.02626	4.37	1.244e-05

The output shows that *bd* has a significant p-value that is it is indeed a feature based on which the digits and letters can be sorted.

Table 18: Contrasts

	letter
digit	0
letter	1

The contrasts are so arranged that 'digit' is denoted by 0 and 'letter' by 1. So the coefficient is 0.115 indicates that for an unit increase in *bd*, the log odds of the probability that the character is a letter increses by 0.115

A 5-fold cross validation is used to evaluate the accuracy of the model.

Table 19: Confusion Matrix of 5-fold cross validated logit model(single feature)

	digit	letter
digit	32	8
letter	18	42

The accuracy of the model is reported as .74 and thus the model predicts the correct outcome 74% of the time. On 18 cases, it has predicted a digit falsely as a letter. And on 8 instances, the model has wrongly classified a letter as a digit.

4.2

We select the five features from the reduced data from 3.2
We select the features for which there is significant difference between the digits and letters. Also, there should be no correlation among the features themselves. Let us run our test from 3.5 to get the features.

Table 20: Features that can distinguish between digits and letters

feature	sig
height	0.02843
width	1.368e-07
cols_with_1	0.02797
neigh_1	4.788e-08

We select the feature *bd* even though its not in the list, as it was the best feature to distinguish between digits and letters. Thus the selected features are *bd, height, width, cols_with_1 and neigh_1*
We fit the logistic regression model with these five features. The model and the summary is given below

```
$call
train.formula(form = type ~ bd + height + cols_with_1 + neigh_1 +
    width, data = data, method = "glm", family = "binomial",
    trControl = ctrl)
```

Table 21: Summary of the model

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-5.009	2.873	-1.743	0.08128
bd	0.1004	0.03311	3.032	0.002425
height	-0.07136	0.2195	-0.3251	0.7451
cols_with_1	-2.443	0.6486	-3.767	0.0001653
neigh_1	-0.6045	0.3378	-1.789	0.07354
width	1.302	0.3957	3.289	0.001004

Table 22: Confusion Matrix of 5-fold cross validated logit model(five features)

	digit	letter
digit	41	6
letter	9	44

We observe that the coefficients of height and neigh_1 are insignificant at the 5% level of significance. Even then model has an accuracy of 85%, not bad considering out training set consists only of 100 observations.

4.3

The simple model that classifies everything as ‘digit’ would be right 50% of the times and wrong 50% of the times. So it would have an accuracy score of .5

The models in 4.1 and 4.2 have accuracy scores of .74 and .85 respectively. So we may say that they perform a lot better than the simple baseline model.
Although the model with five features is a bit complicated and it takes some time to compute, the performance is so better compared to the simple baseline model that we can overlook the complication. And the model with the single feature also performs a lot better than the baseling model, but it is not that complicated. Of course the 5-fold cross validation takes some time, but it can be overlooked if we consider the gain in accuracy.

4.4

Let us design a simulation to find the answer to this question.
At first we generate a character randomly with all 10 being equally likely and note down if its a 'digit' or a 'letter'
Next, we generate one of the types i.e. 'digit' or 'letter' randomly with equal probability.
If the type of the randomly generated character(i.e 'digit' or 'letter') matches the generated type, we note that down. We do this a large number of times. Finally, we return the correct guesses divided by the total number of iterations, which is actually the accuracy score.

```
[1] "n = 100000"
```

Table 23: Simulated model with accuracy scores

iteration	accuracy
1	0.4979
2	0.4966
3	0.501
4	0.5012
5	0.5022

So the mean accuracy is $.499$ which is approximately .5
So this model has an accuracy of 50%

4.5

The accuracy of the model by chance is 50% i.e. we will get about 50 correct predicitons out of 100 if we use the chance model. Whereas the model in 4.2 identified 85 chracters correctly. So this is a certainly much greater than the chance model. What remains to see is that if the accuracy of the logit model with five features is significantly greater than the chance model.
Here, the hypothesis we are testing is:
H_0: The distribution of the logit model and the chance model is the same vs
H_1: The distributions of the two models arer not same

We may use the binomial distribution for the purpose. If X ~ binomial(100, 0.5) then $P(X >= 85)$ gives be the probability that the number of correct answers exceeds 85, which is almost 0. This works as the p-value for the test.

The meaning of this is that it is highly unlikely that we will see an accuracy score of 85 or more if the distribution of the logit models is $binomial(100, .5)$
Thus we may conclude that the logit model is significantly better than the chance model.

4.6

We can view the confusion matrix from 4.2 for this section

Table 24: Confusion Matrix of 5-fold cross validated logit model(five features)

	digit	**letter**
digit	41	6
letter	9	44

It is seen from the Confusion matrix that out of 50 digits 41 are classified correctly and out of 50 letters 44 are classified correctly.
It is also observed that 9 of the digits are incorrectly classified as letters. And 6 of the letters are incorrectly classified as digits.
One interesting pattern we may be observing in this accuracy data is that the letters are more classified more correctly than the digits.

4.7

At first, we find the features for which there is significant difference among the 10 characters. We are using the reduced data set, the data set obtained after removing the highly correlated features in 3.2

Table 25: Features that show sig difference between the characters.

feature	sig
height	1.797e-05
tallness	2.287e-09
cols_with_1	4.989e-10
rows_with_5_plus	0.0005586
cols_with_5_plus	1.748e-05
neigh_3_plus	3.696e-05
none_below	2.162e-09
bd	2.939e-07

So we select *cols_with_1*, *none_below*, *tallness*, *bd and height* as the features to fit the model.
This time we will fit a *k-nearest-neighbor* model with 5-fold cross validation. The features are scaled before fitting the model.

Table 26: Accuracy of the knn model for diff values of k

k	Accuracy

1	0.53
3	0.48
5	0.51
9	0.55

4.8

Based on the accuracy results, we choose the model with k = 9

Thus our final selected model is **k-nearest-neighbor** with **k = 9** and the features in the model are **cols_with_1, none_below, tallness, bd annd height*.
Since this value of k yields the greates accuracy, so it is chosen. The features are chosen keeping in mind that they had the most significance when tested for difference among the character. The more the difference of a feature from character to character, the better it can predict. Also they are not much correlated among themselves. These properties led us to the final model. Let us fit the final model to the data.

Table 27: Confusion Matrix of the final model

	1	**3**	**4**	**5**	**7**	**a**	**b**	**c**	**d**	**e**
1	8	0	0	0	0	1	0	0	0	0
3	0	8	0	2	0	0	0	0	1	0
4	0	0	10	0	1	0	0	0	0	0
5	0	1	0	4	0	0	0	1	0	2
7	1	0	0	0	8	0	0	0	0	0
a	0	0	0	0	0	2	0	0	0	0
b	1	0	0	1	0	1	8	1	2	1
c	0	0	0	1	1	2	0	6	0	2
d	0	1	0	0	0	4	1	0	7	0
e	0	0	0	2	0	0	1	2	0	5

Table 28: Accuracy of the final model

Model	**Accuracy**
label~cols_with_1+none_below+ tallness+bd+height	0.66

We observe that the final model identifies 8 ‘1’s correctly, 8 ’3’s correctly, all 10 ’4’s correctly and so on. It had problems classifying the letter ’a’, being correct only twice. So the accuracy of the final model is .66
This is a good accuracy even thought it is working on seen data, given that there are only 100 training observations and there are 10 classes to classify. So we can say that the overall performance of the final model is quite satisfactory.

Conclusion:

This was the project. It had multiple goals. We have done Feature Engineering, Statistical Analysis and Machine Learning in different sections of the assignment. The goal was to classify images of 10 characters, 5 digits and 5 letters based on features that were engineered by us.

We calculated some interesting features, tested them statistically which finally led to building a machine learning model that predicted the characters.

References:

1. Kabacoff, RI 2015, *R in Action*, 2nd Ed, Manning, New York, US
2. Zumel N, Mount J, 2014, *Practical Data Science with R*, Manning, New York, Us

Codes:

Please find the codes in a github repository here