```
#Question 1
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```
library(plyr)
library(MLmetrics)
```

```
##
## Attaching package: 'MLmetrics'

## The following object is masked from 'package:base':
##
##     Recall
```

```
library(Hmisc)
```

```
## Loading required package: lattice

## Loading required package: survival

## Loading required package: Formula

## Loading required package: ggplot2

##
## Attaching package: 'Hmisc'

## The following objects are masked from 'package:plyr':
##
##     is.discrete, summarize

## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```
library(tidyverse);
```

```
## -- Attaching packages ----------------------------------------- tidyverse 1.3.0 --

## v tibble  3.0.3     v dplyr   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
## v purrr   0.3.4

## -- Conflicts -------------------------------------------- tidyverse_conflicts() --
## x dplyr::arrange()   masks plyr::arrange()
## x purrr::compact()   masks plyr::compact()
## x dplyr::count()     masks plyr::count()
## x dplyr::failwith()  masks plyr::failwith()
## x dplyr::filter()    masks stats::filter()
## x dplyr::id()        masks plyr::id()
## x dplyr::lag()       masks stats::lag()
## x dplyr::mutate()    masks plyr::mutate()
## x dplyr::rename()    masks plyr::rename()
## x dplyr::src()       masks Hmisc::src()
## x dplyr::summarise() masks plyr::summarise()
## x dplyr::summarize() masks Hmisc::summarize(), plyr::summarize()
```

```
library(Rtsne);
data <- read_csv("datasets_26073_33239_weight-height.csv")
```

```
## Parsed with column specification:
## cols(
##   Gender = col_character(),
##   Height = col_double(),
##   Weight = col_double()
## )
```

```
data$Gender = factor(data$Gender,
                        levels = c("Male","Female"),
                        labels = c(0,1))
data$Gender <- as.character(data$Gender)
head(data$Gender)
```

```
## [1] "0" "0" "0" "0" "0" "0"
```

```
smp_size <- floor(0.75 * nrow(data))

## set the seed to make your partition reproducible
set.seed(100)
train_ind <- sample(seq_len(nrow(data)), size = smp_size)

train <- data[train_ind, ]
test <- data[-train_ind, ]
```

```
model <- gbm(Gender ~ Height + Weight,distribution="bernoulli",data=train,
             n.trees = 100,
             interaction.depth = 2,
             shrinkage = 0.1);
pred <- predict(model, newdata = test, type="response");
```

```
## Using 100 trees...
```

```
sum(pred>0.5)/nrow(test)
```

```
## [1] 0.506
```

1. The result of the last exercise was 0.464, the accuracy of the last exercise is close to this one which is 0.506.

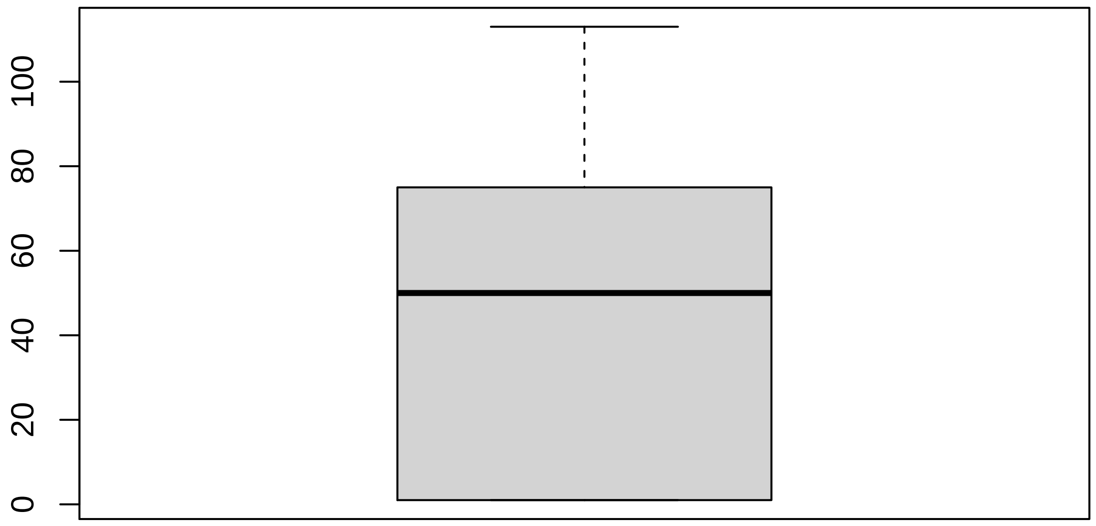
```
data2 <- read_csv("datasets_38396_60978_charcters_stats.csv")
```

```
## Parsed with column specification:
## cols(
##   Name = col_character(),
##   Alignment = col_character(),
##   Intelligence = col_double(),
##   Strength = col_double(),
##   Speed = col_double(),
##   Durability = col_double(),
##   Power = col_double(),
##   Combat = col_double(),
##   Total = col_double()
## )
```

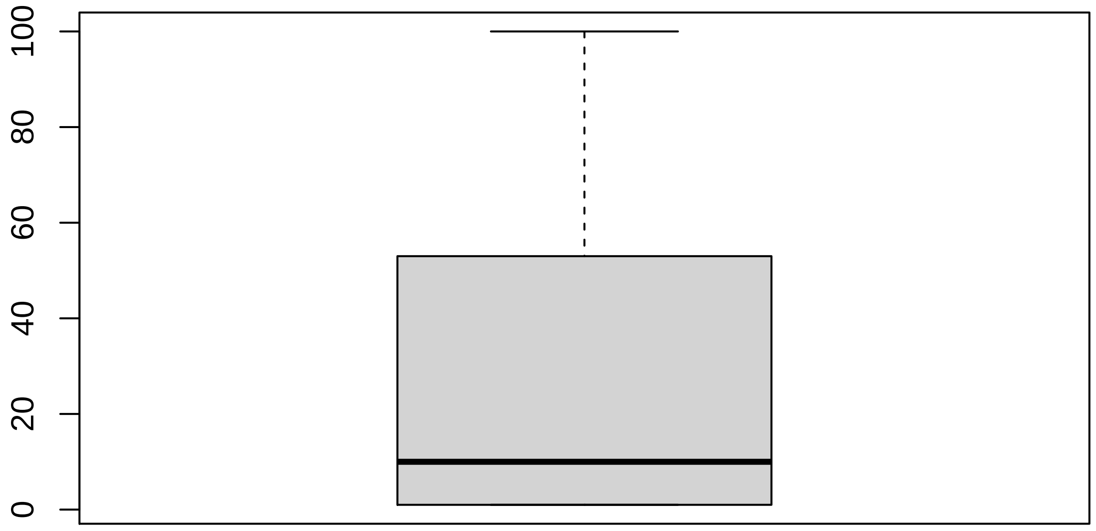
```
data2
```

```
## # A tibble: 611 x 9
##    Name      Alignment Intelligence Strength Speed Durability Power Combat Total
##    <chr>     <chr>            <dbl>    <dbl> <dbl>      <dbl> <dbl>  <dbl> <dbl>
##  1 3-D Man   good                 50       31    43         32    25     52   233
##  2 A-Bomb    good                 38      100    17         80    17     64   316
##  3 Abe Sapi~ good                 88       14    35         42    35     85   299
##  4 Abin Sur  good                 50       90    53         64    84     65   406
##  5 Abominat~ bad                  63       80    53         90    55     95   436
##  6 Abraxas   bad                  88      100    83         99   100     56   526
##  7 Adam Mon~ good                 63       10    12        100    71     64   320
##  8 Adam Str~ good                  1        1     1          1     0      1     5
##  9 Agent  13 good                  1        1     1          1     0      1     5
## 10 Agent Bob good                 10        8    13          5     5     20    61
## # ... with 601 more rows
```

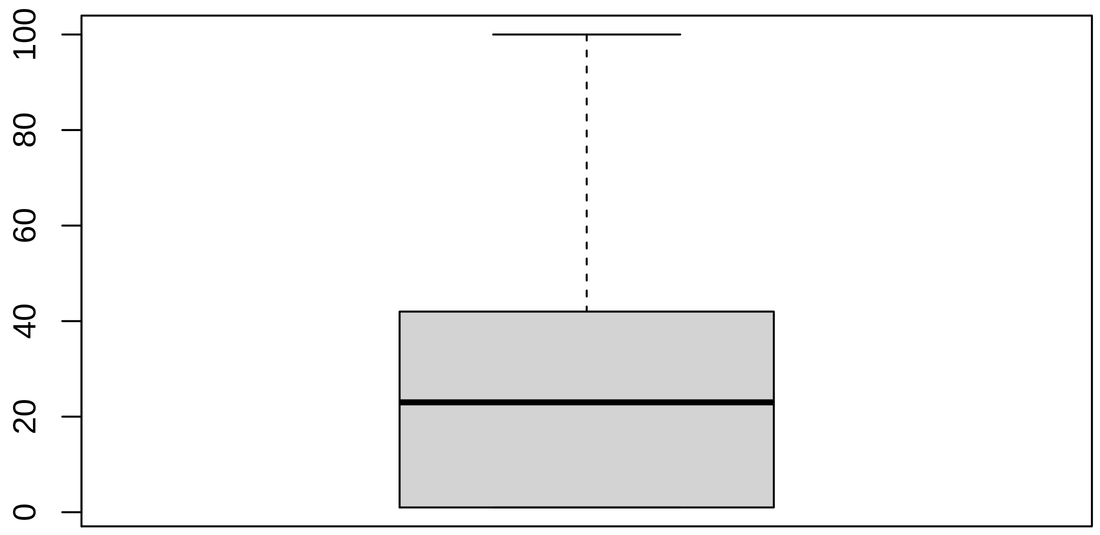
```r
boxplot(data2[3])
```



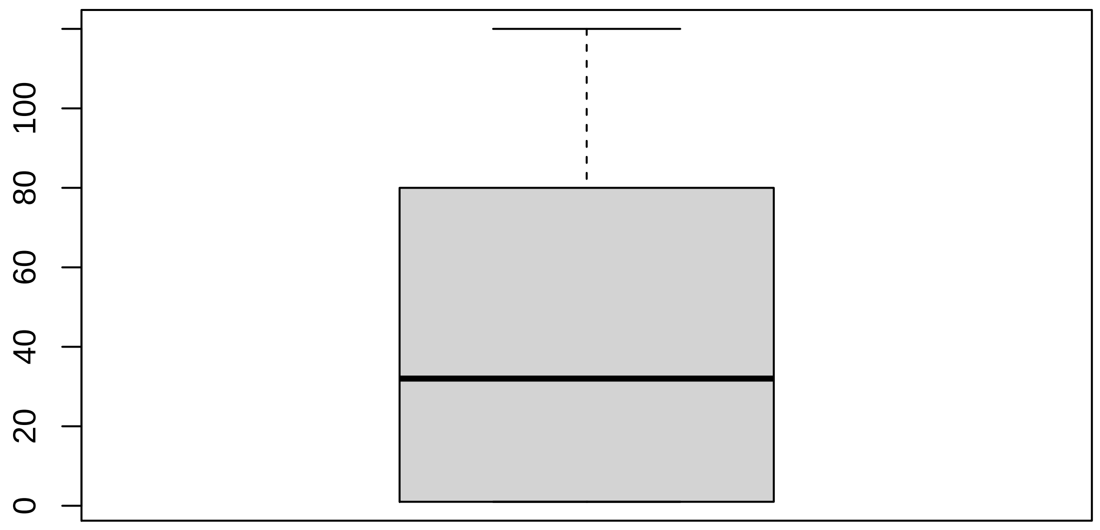
```r
boxplot(data2[4])
```



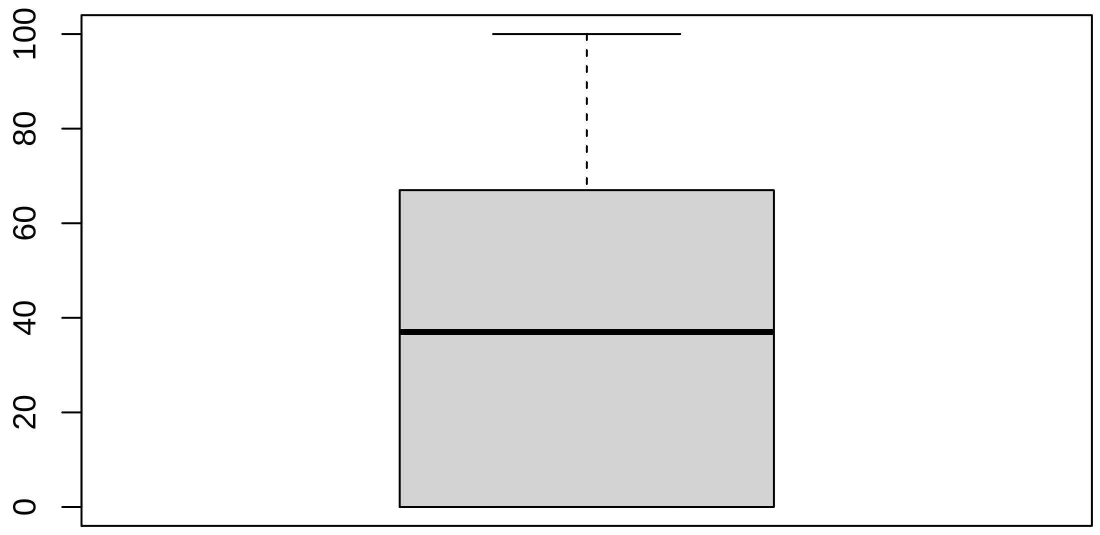
```r
boxplot(data2[5])
```



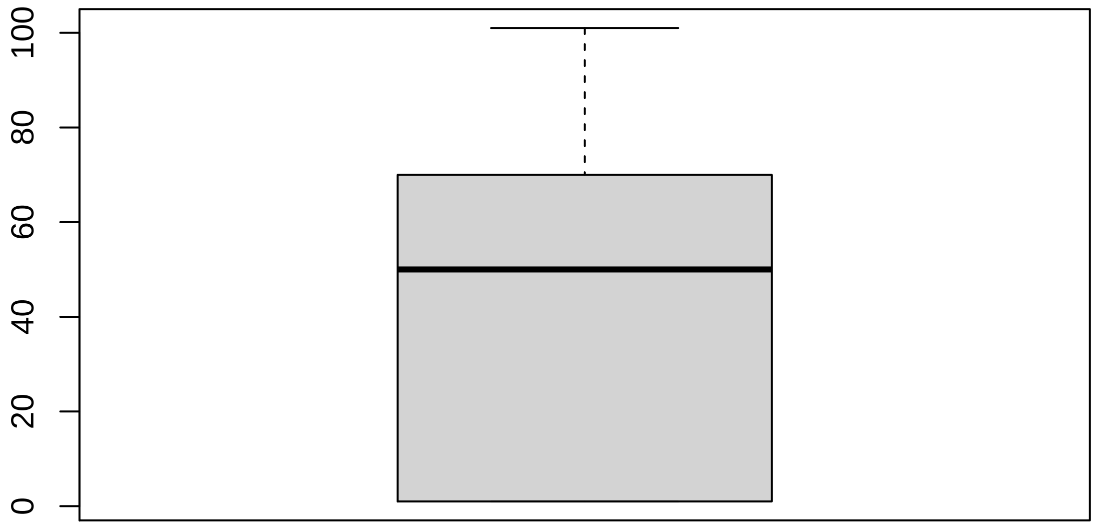
```
boxplot(data2[6])
```



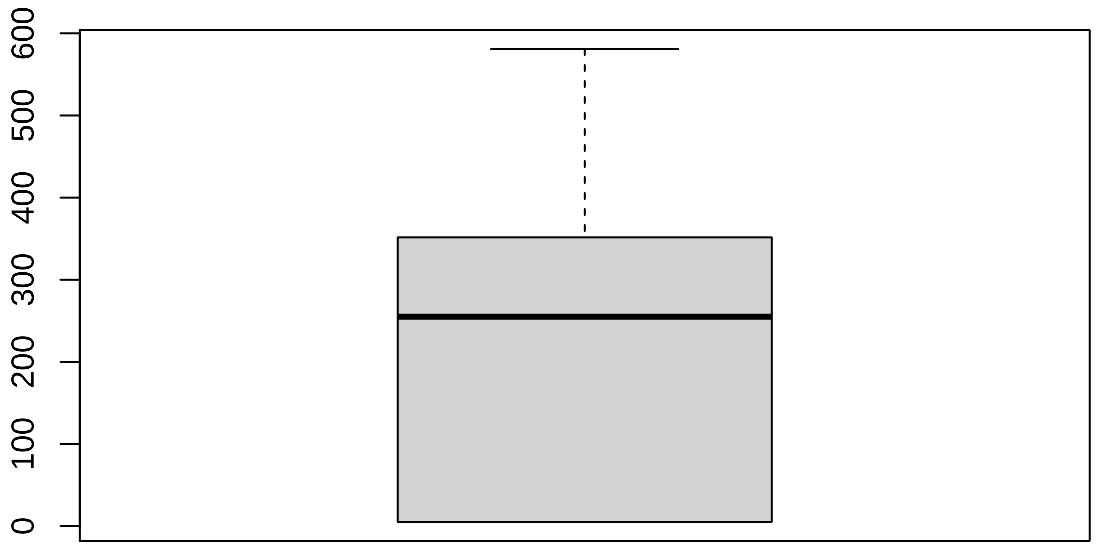
```
boxplot(data2[7])
```



```
boxplot(data2[8])
```



```
boxplot(data2[9])
```



Q2

1. Depending on the boxplot, there are not any irregualarities and I dont need to filter out a subset of rows.

```
data2.pca <- prcomp(data2[,3:9])
data2.pca
```

```
## Standard deviations (1, .., p=7):
## [1] 1.809591e+02 2.492497e+01 1.961815e+01 1.602658e+01 1.496131e+01
## [6] 1.451877e+01 2.352081e-14
##
## Rotation (n x k) = (7 x 7):
##                     PC1          PC2          PC3         PC4         PC5
## Intelligence -0.1558138 -0.512410987  0.011954768 -0.10394268  0.62945701
## Strength     -0.1485702  0.574336433  0.387331365 -0.12549242  0.42880903
## Speed        -0.1155588  0.118684676 -0.190901802  0.88752932 -0.04624362
## Durability   -0.1834677  0.356326076  0.088248520 -0.27898376 -0.44316183
## Power        -0.1668084 -0.022455199 -0.764794152 -0.32552703 -0.09544381
## Combat       -0.1544233 -0.515763934  0.469773884 -0.01091879 -0.46054378
## Total        -0.9246422 -0.001282934  0.001612583  0.04266465  0.01287301
##                     PC6        PC7
## Intelligence  0.40390193 -0.3779645
## Strength     -0.39440627 -0.3779645
## Speed         0.05841828 -0.3779645
## Durability    0.64381834 -0.3779645
## Power        -0.35892356 -0.3779645
## Combat       -0.36657410 -0.3779645
## Total        -0.01376538  0.3779645
```

```
summary(data2.pca)
```

```
## Importance of components:
##                             PC1      PC2      PC3      PC4     PC5      PC6
## Standard deviation     180.9591 24.92497 19.61815 16.02658 14.9613 14.51877
## Proportion of Variance   0.9507  0.01804  0.01117  0.00746  0.0065  0.00612
## Cumulative Proportion    0.9507  0.96875  0.97992  0.98738  0.9939  1.00000
##                              PC7
## Standard deviation     2.352e-14
## Proportion of Variance 0.000e+00
## Cumulative Proportion  1.000e+00
```

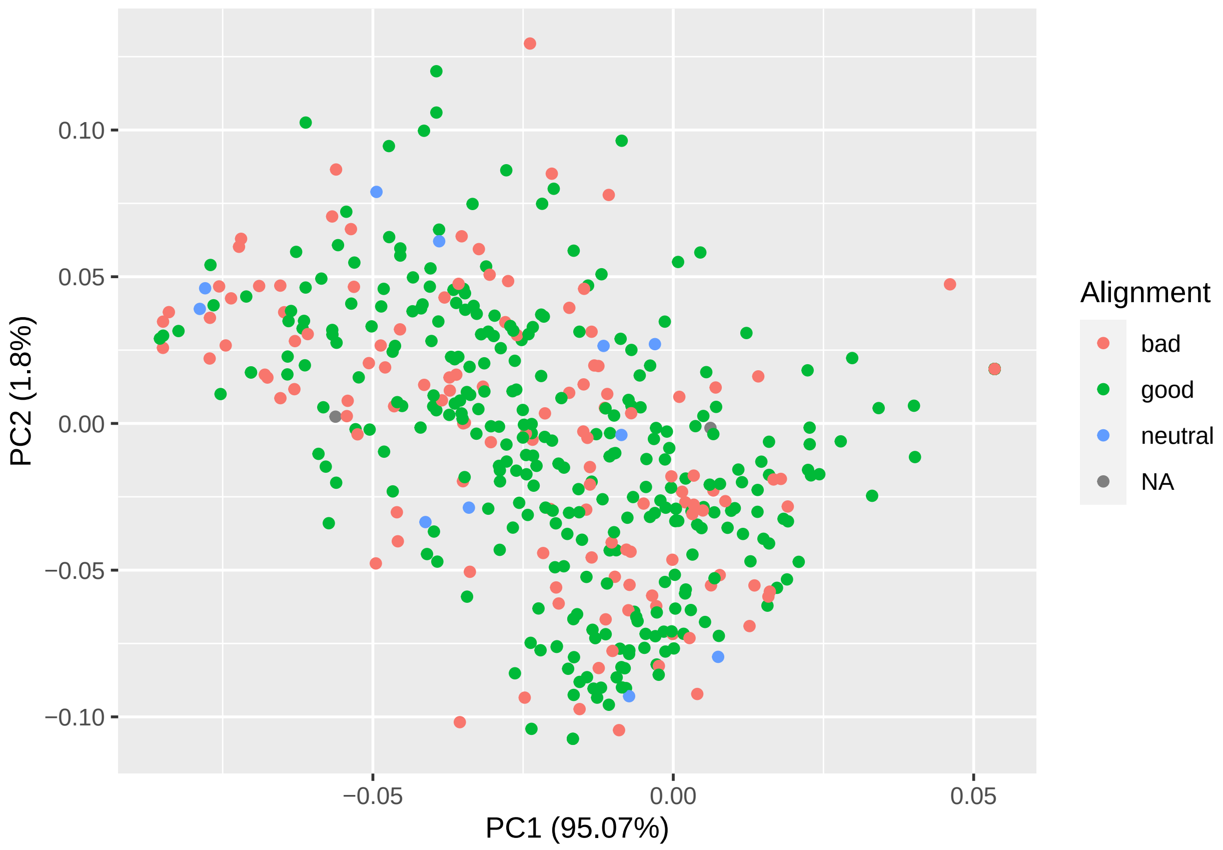
Q2:

2. According to the result of the principal component analysis, we need 1 component to get 85% of the variation in the data set.
3. We dont need to normalize these columns.
4. The total column is the total of the values in other numerical columns.
5. We shouldn't inlude that in the PCA, the largest principal components and the total column have the largest weighted values

```
library(ggfortify)
pca.plot <- autoplot(data2.pca, data = data2,colour = 'Alignment')
```

```
## Warning: `select_()` is deprecated as of dplyr 0.7.0.
## Please use `select()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
pca.plot
```



6)The points of the 4 groups are clustered for the most part; however, the three points at the right of the graph may be outliers. The data does not appear to depart widely from multivariate normality.

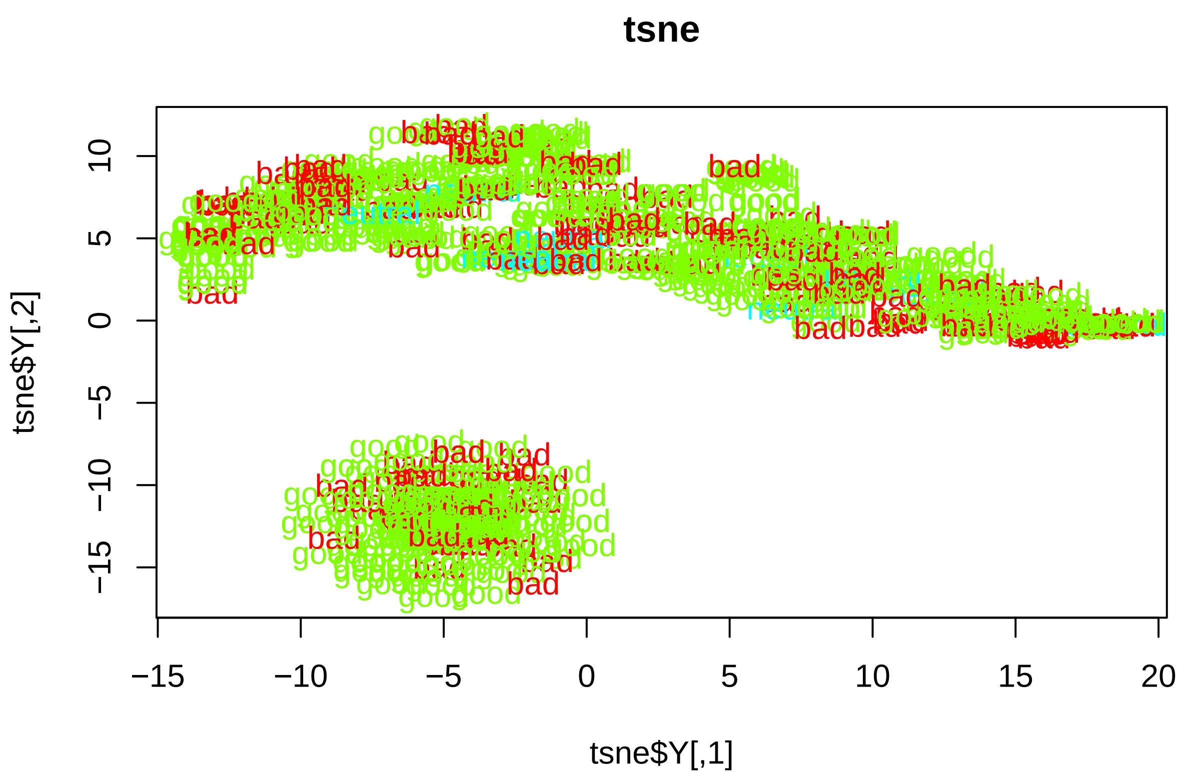
```
library(Rtsne);
Labels <- data2$Alignment
data2$Alignment <- as.factor(data2$Alignment)
colors = rainbow(length(unique(data2$Alignment)))
names(colors) = unique(data2$Alignment)
tsne <- Rtsne(data2[,3:9], dims = 2, perplexity=30, verbose=TRUE, max_iter = 500,check_duplicates = FALS
```

```
## Performing PCA
## Read the 611 x 7 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 30.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
## Done in 0.07 seconds (sparsity = 0.193383)!
## Learning embedding...
## Iteration 50: error is 54.248133 (50 iterations in 0.10 seconds)
## Iteration 100: error is 49.383369 (50 iterations in 0.09 seconds)
## Iteration 150: error is 48.987868 (50 iterations in 0.08 seconds)
## Iteration 200: error is 48.816828 (50 iterations in 0.07 seconds)
## Iteration 250: error is 48.726268 (50 iterations in 0.08 seconds)
## Iteration 300: error is 0.713593 (50 iterations in 0.09 seconds)
## Iteration 350: error is 0.653791 (50 iterations in 0.09 seconds)
## Iteration 400: error is 0.635313 (50 iterations in 0.08 seconds)
## Iteration 450: error is 0.626903 (50 iterations in 0.07 seconds)
## Iteration 500: error is 0.620394 (50 iterations in 0.07 seconds)
## Fitting performed in 0.84 seconds.
```

```
exeTimeTsne<- system.time(Rtsne(data2[,3:9], dims = 2, perplexity=30, verbose=TRUE, max_iter = 500,chec
```

```
## Performing PCA
## Read the 611 x 7 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 30.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
## Done in 0.06 seconds (sparsity = 0.193383)!
## Learning embedding...
## Iteration 50: error is 52.810262 (50 iterations in 0.09 seconds)
## Iteration 100: error is 49.291739 (50 iterations in 0.08 seconds)
## Iteration 150: error is 48.948248 (50 iterations in 0.08 seconds)
## Iteration 200: error is 48.793637 (50 iterations in 0.08 seconds)
## Iteration 250: error is 48.705934 (50 iterations in 0.09 seconds)
## Iteration 300: error is 0.724315 (50 iterations in 0.09 seconds)
## Iteration 350: error is 0.646306 (50 iterations in 0.08 seconds)
## Iteration 400: error is 0.629444 (50 iterations in 0.08 seconds)
## Iteration 450: error is 0.617899 (50 iterations in 0.07 seconds)
## Iteration 500: error is 0.610791 (50 iterations in 0.08 seconds)
## Fitting performed in 0.81 seconds.
```

```
plot(tsne$Y, t='n', main="tsne")
text(tsne$Y, labels=data2$Alignment, col=colors[data2$Alignment])
```



Q3:

The points within the individual clusters are highly similar to each other and close to points in other clusters. The same pattern likely holds in a high-dimensional original dataset. In the digits dataset, t-SNE didn't separate clusters of each digit class. In the context of alignment, these clusters represent the alignment with similar associated characters.

Q4: The plot produced by python is in the same directory that was named as 'tsne_2d'.

```
data3 = data2[,2:9]
data3
```

```
## # A tibble: 611 x 8
##    Alignment Intelligence Strength Speed Durability Power Combat Total
##    <fct>            <dbl>    <dbl> <dbl>      <dbl> <dbl>  <dbl> <dbl>
##  1 good                50       31    43         32    25     52   233
##  2 good                38      100    17         80    17     64   316
##  3 good                88       14    35         42    35     85   299
##  4 good                50       90    53         64    84     65   406
##  5 bad                 63       80    53         90    55     95   436
##  6 bad                 88      100    83         99   100     56   526
##  7 good                63       10    12        100    71     64   320
##  8 good                 1        1     1          1     0      1     5
##  9 good                 1        1     1          1     0      1     5
## 10 good                10        8    13          5     5     20    61
## # ... with 601 more rows
```

```
library(caret);
```

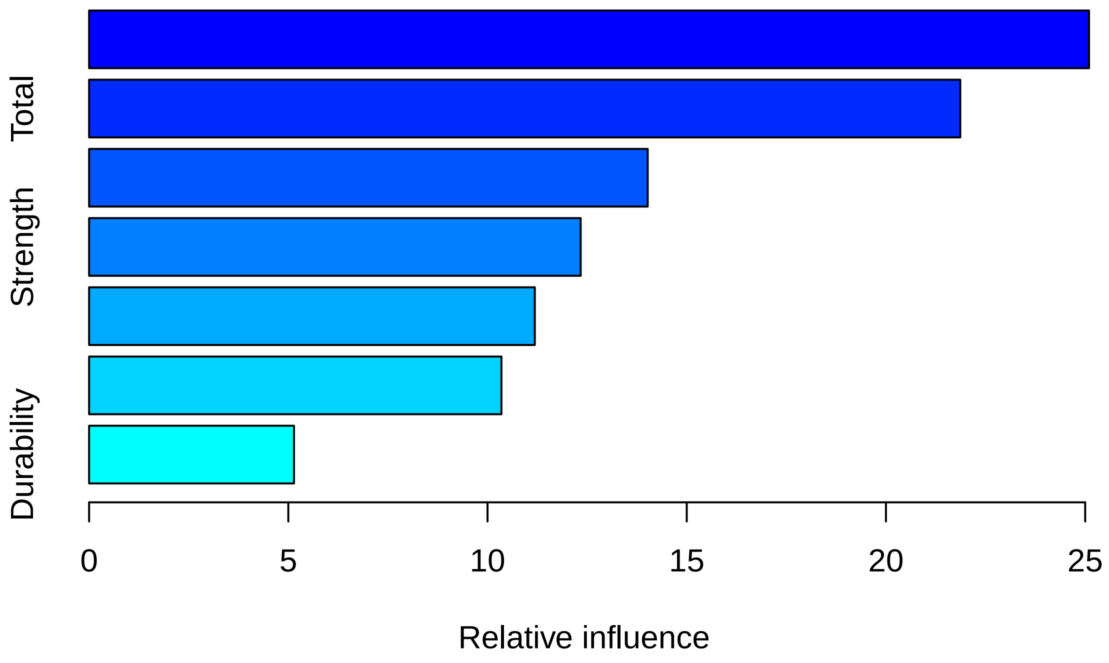
```
##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift

## The following object is masked from 'package:survival':
##
##     cluster

## The following objects are masked from 'package:MLmetrics':
##
##     MAE, RMSE
```

```
control <- trainControl(method="repeatedcv", number=10, repeats=3)
set.seed(7)
modelGbm <- train(Alignment~., data=data3, method="gbm",na.action=na.exclude, trControl=control, verbos
summary(modelGbm)
```



```
##                     var   rel.inf
## Intelligence Intelligence 25.097294
## Total               Total 21.865455
## Speed               Speed 14.020442
## Strength         Strength 12.338157
## Combat             Combat 11.186908
## Power               Power 10.349115
## Durability     Durability  5.142628
```

```
modelGbm
```

```
## Stochastic Gradient Boosting
##
## 611 samples
##   7 predictor
##   3 classes: 'bad', 'good', 'neutral'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 545, 547, 547, 548, 546, 548, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7132108  0.09383176
##   1                  100      0.7110793  0.10979519
##   1                  150      0.7044758  0.11139626
##   2                   50      0.7066983  0.11038054
##   2                  100      0.7001369  0.11865352
##   2                  150      0.6913041  0.10528900
##   3                   50      0.7089882  0.13808781
##   3                  100      0.6893035  0.10321287
##   3                  150      0.6931471  0.13589523
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
```

```
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 50, interaction.depth =
##  1, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
set.seed(7)
data4 = data3
data4 = data4 %>% relocate(Alignment, .after = last_col())
data4 <- na.omit(data4)

control <- rfeControl(functions=rfFuncs, method="cv", number=10)
results <- rfe(data4[,1:7], data4[[8]], sizes=c(1:8),rfeControl=control)
```

```
print(results)
```

```
##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (10 fold)
##
## Resampling performance over subset size:
##
##  Variables Accuracy   Kappa AccuracySD KappaSD Selected
##          1   0.6942 0.06296    0.03482 0.08919
##          2   0.7123 0.13521    0.03364 0.10764
##          3   0.7089 0.11555    0.04470 0.13908
##          4   0.7189 0.17224    0.04758 0.13336
##          5   0.7123 0.15976    0.03929 0.11315
##          6   0.7189 0.15603    0.04250 0.12807
##          7   0.7270 0.16442    0.02841 0.09814        *
##
## The top 5 variables (out of 7):
##    Intelligence, Combat, Durability, Power, Total
```

```
predictors(results)
```

```
## [1] "Intelligence" "Combat"       "Durability"   "Power"        "Total"
## [6] "Strength"     "Speed"
```

Q5: According to the results generated above, the best parameters are Intelligence, Combat, Durability, Power, Total.

Q6: k-fold Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.

The reasons why we need to use k-fold cross-validation are it is simple to understand and it is generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split. We are able to 1)make predictions on all of our data using k-fold cross-validation. 2) get more metrics and draw important conclusion both about our algorithm and our data.3)work with dependent/grouped data. 4) Do Parameters Fine-Tuning

if we just report a single number for the accuracy of our model, this accuracy is more likely to be biased and inaccurate.

Q7: Recursive Feature Elimination, or RFE, is a popular feature selection algorithm. it is easy to configure and it is effective at selecting the best features.

RFE works by searching for a subset of features by starting with all features in the training dataset and removing features until the desired number remains. In other words, RFE works as follows: fitting the given machine learning algorithm used in the core of the model, ranking features by importance, discarding the least important features, and re-fitting the model. This process is repeated until a specified number of features remains.