Assignment 3 PCA

Nicholas Jacob and Yechang Qi

2024-09-05

Glass Data

1.a.i I'll load the data directly from my machine. I wanted to load it directly from the web but seems as though those links don't exist anymore. I've added the names from the info included in the zip file with the data.

```
names <- c('Id', 'RI','Na','Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe', 'Type')
Glass <- read.csv('glass.data',header = FALSE)
names(Glass)<- names
Glass$Type = as.character(Glass$Type)
head(Glass)
```

```
##   Id      RI    Na   Mg   Al    Si    K   Ca Ba   Fe Type
## 1  1 1.52101 13.64 4.49 1.10 71.78 0.06 8.75  0 0.00    1
## 2  2 1.51761 13.89 3.60 1.36 72.73 0.48 7.83  0 0.00    1
## 3  3 1.51618 13.53 3.55 1.54 72.99 0.39 7.78  0 0.00    1
## 4  4 1.51766 13.21 3.69 1.29 72.61 0.57 8.22  0 0.00    1
## 5  5 1.51742 13.27 3.62 1.24 73.08 0.55 8.07  0 0.00    1
## 6  6 1.51596 12.79 3.61 1.62 72.97 0.64 8.07  0 0.26    1
```

Let's double check that there isn't a duplicate since there was a warning about a possible dup.

```
any(!duplicated(Glass))
```

```
## [1] TRUE
```

Does not appear to be any duplicates in this dataset. Now we'll create the correlation matrix. We should remove the 'Id' and 'Type' from the data before we do the correlation. Originally we saw strong correlation between Id and Type only because the data was organized by type

```
corMat <- cor(Glass[,2:10])
corMat
```

```
##               RI          Na           Mg          Al          Si             K
## RI  1.0000000000 -0.19188538 -0.122274039 -0.40732603 -0.54205220 -0.289832711
## Na -0.1918853790  1.00000000 -0.273731961  0.15679367 -0.06980881 -0.266086504
## Mg -0.1222740393 -0.27373196  1.000000000 -0.48179851 -0.16592672  0.005395667
## Al -0.4073260341  0.15679367 -0.481798509  1.00000000 -0.00552372  0.325958446
## Si -0.5420521997 -0.06980881 -0.165926723 -0.00552372  1.00000000 -0.193330854
## K  -0.2898327111 -0.26608650  0.005395667  0.32595845 -0.19333085  1.000000000
## Ca  0.8104026963 -0.27544249 -0.443750026 -0.25959201 -0.20873215 -0.317836155
## Ba -0.0003860189  0.32660288 -0.492262118  0.47940390 -0.10215131 -0.042618059
## Fe  0.1430096093 -0.24134641  0.083059529 -0.07440215 -0.09420073 -0.007719049
##            Ca            Ba           Fe
## RI  0.8104027 -0.0003860189  0.143009609
## Na -0.2754425  0.3266028795 -0.241346411
```

```
## Mg -0.4437500 -0.4922621178  0.083059529
## Al -0.2595920  0.4794039017 -0.074402151
## Si -0.2087322 -0.1021513105 -0.094200731
## K  -0.3178362 -0.0426180594 -0.007719049
## Ca  1.0000000 -0.1128409671  0.124968219
## Ba -0.1128410  1.0000000000 -0.058691755
## Fe  0.1249682 -0.0586917554  1.000000000
```

1.a.ii. Compute the eigen values and vectors of this matrix.

```
eMat <- eigen(corMat)
eMat
```

```
## eigen() decomposition
## $values
## [1] 2.511163726 2.050072185 1.404843994 1.157862446 0.914002247 0.527635193
## [7] 0.368958443 0.063852948 0.001608818
##
## $vectors
##             [,1]        [,2]          [,3]        [,4]         [,5]        [,6]
##  [1,]  0.5451766 -0.28568318 -0.0869108293  0.14738099  0.073542700 -0.11528772
##  [2,] -0.2581256 -0.27035007  0.3849196197  0.49124204 -0.153683304  0.55811757
##  [3,]  0.1108810  0.59355826 -0.0084179590  0.37878577 -0.123509124 -0.30818598
##  [4,] -0.4287086 -0.29521154 -0.3292371183 -0.13750592 -0.014108879  0.01885731
##  [5,] -0.2288364  0.15509891  0.4587088382 -0.65253771 -0.008500117 -0.08609797
##  [6,] -0.2193440  0.15397013 -0.6625741197 -0.03853544  0.307039842  0.24363237
##  [7,]  0.4923061 -0.34537980  0.0009847321 -0.27644322  0.188187742  0.14866937
##  [8,] -0.2503751 -0.48470218 -0.0740547309  0.13317545 -0.251334261 -0.65721884
##  [9,]  0.1858415  0.06203879 -0.2844505524 -0.23049202 -0.873264047  0.24304431
##              [,7]        [,8]        [,9]
##  [1,]  0.08186724  0.75221590 -0.02573194
##  [2,]  0.14858006  0.12769315  0.31193718
##  [3,] -0.20604537  0.07689061  0.57727335
##  [4,] -0.69923557  0.27444105  0.19222686
##  [5,]  0.21606658  0.37992298  0.29807321
##  [6,]  0.50412141  0.10981168  0.26050863
##  [7,] -0.09913463 -0.39870468  0.57932321
##  [8,]  0.35178255 -0.14493235  0.19822820
##  [9,]  0.07372136  0.01627141  0.01466944
```

As expected for the correlation matrix, all values are positive. Eigen vectors don't show me much yet.

1.a.iii. Next we'll do the principle component decomposition

```
pMat <- prcomp(Glass[,2:10],scale = TRUE)
pMat
```

```
## Standard deviations (1, .., p=9):
## [1] 1.58466518 1.43180731 1.18526115 1.07604017 0.95603465 0.72638502 0.60741950
## [8] 0.25269141 0.04011007
##
## Rotation (n x k) = (9 x 9):
##           PC1         PC2           PC3         PC4          PC5         PC6
## RI -0.5451766  0.28568318  0.0869108293  0.14738099 -0.073542700  0.11528772
## Na  0.2581256  0.27035007 -0.3849196197  0.49124204  0.153683304 -0.55811757
## Mg -0.1108810 -0.59355826  0.0084179590  0.37878577  0.123509124  0.30818598
## Al  0.4287086  0.29521154  0.3292371183 -0.13750592  0.014108879 -0.01885731
```

```
## Si  0.2288364 -0.15509891 -0.4587088382 -0.65253771  0.008500117  0.08609797
## K   0.2193440 -0.15397013  0.6625741197 -0.03853544 -0.307039842 -0.24363237
## Ca -0.4923061  0.34537980 -0.0009847321 -0.27644322 -0.188187742 -0.14866937
## Ba  0.2503751  0.48470218  0.0740547309  0.13317545  0.251334261  0.65721884
## Fe -0.1858415 -0.06203879  0.2844505524 -0.23049202  0.873264047 -0.24304431
##            PC7         PC8         PC9
## RI -0.08186724 -0.75221590 -0.02573194
## Na -0.14858006 -0.12769315  0.31193718
## Mg  0.20604537 -0.07689061  0.57727335
## Al  0.69923557 -0.27444105  0.19222686
## Si -0.21606658 -0.37992298  0.29807321
## K  -0.50412141 -0.10981168  0.26050863
## Ca  0.09913463  0.39870468  0.57932321
## Ba -0.35178255  0.14493235  0.19822820
## Fe -0.07372136 -0.01627141  0.01466944
```

1.a.iv. These are different. The correlation is a scaled version of the covariance matrix so essentially, these are built off of similar matrices up to a rescaling. If before building the correlation e-values, we did a z-score rescaling they would be exactly the same. Instead as we will see in the next section, they both form the same ortho-normal basis.

1.a.v. Show that these vectors are orthogonal to each other.

First, I needed to see how to access the vectors

```
eMat$vectors[,1] #
```

```
## [1]  0.5451766 -0.2581256  0.1108810 -0.4287086 -0.2288364 -0.2193440  0.4923061
## [8] -0.2503751  0.1858415
```

We did not find a function for doing the inner product but since the formula is

$$x \cdot y = \sum_{i=1}^{n} x_i y_i$$

We simply asked R to do that with base packages.

```
sum(pMat$rotation[,1]*eMat$vectors[,1])
```

```
## [1] -1
```

We see that this one is indeed 1 as expected. Let's build a matrix of all 9^2 possible combinations. Yes some are repeated but that does not increase runtime significantly for this data.

```
idMat <- data.frame(matrix(nrow = 9, ncol = 9)) #initialize matrix
for(i in 1:9){
  for (j in 1:9){
    idMat[i,j] <- sum(pMat$rotation[,i]*eMat$vectors[,j]) #add data to matrix
  }
}
idMat #print matrix
```

```
##              X1            X2            X3            X4            X5
## 1 -1.000000e+00 -2.801578e-15  3.397348e-16 -3.504141e-16  8.760354e-17
## 2  2.862294e-15 -1.000000e+00  3.794708e-18  1.821460e-17 -1.101549e-16
## 3 -1.521678e-16 -3.878191e-16 -1.000000e+00  4.054374e-16  1.808449e-16
## 4 -1.908196e-16 -1.353084e-16  4.637675e-16  1.000000e+00 -2.583003e-15
## 5 -2.498002e-16 -4.295609e-16 -7.232171e-16 -1.645168e-15 -1.000000e+00
## 6 -1.700029e-16 -6.852158e-17 -3.948393e-16  5.347285e-16 -5.667125e-16
```

```
## 7  5.030698e-17 -3.729655e-17  4.270943e-16  6.470519e-16 -1.214523e-15
## 8 -1.379105e-16 -3.035766e-17 -1.542820e-16 -1.153591e-16  1.539567e-16
## 9  8.803722e-17  1.116728e-17  2.501254e-16  4.289104e-16 -3.170207e-16
##              X6            X7            X8            X9
## 1  2.879641e-16 -1.977585e-16  3.187554e-16  1.964574e-16
## 2  4.753142e-16 -4.683753e-16  1.170938e-16  7.220786e-17
## 3  3.613917e-16  9.528782e-17  9.031404e-17  3.894454e-16
## 4  1.098947e-15  3.903128e-16 -3.859760e-17 -7.095019e-16
## 5  4.254409e-16  1.854203e-15 -1.986258e-16 -2.569559e-16
## 6 -1.000000e+00 -5.551115e-17  3.001072e-16 -2.298509e-16
## 7  6.192963e-16 -1.000000e+00 -2.168404e-16 -5.169476e-16
## 8 -3.773024e-16  1.214306e-16 -1.000000e+00  1.846803e-15
## 9 -1.838807e-16  3.838076e-17  1.813979e-15  1.000000e+00
```

We this is essentially an identity matrix but it has the overflow errors. Let's force those to be zero.

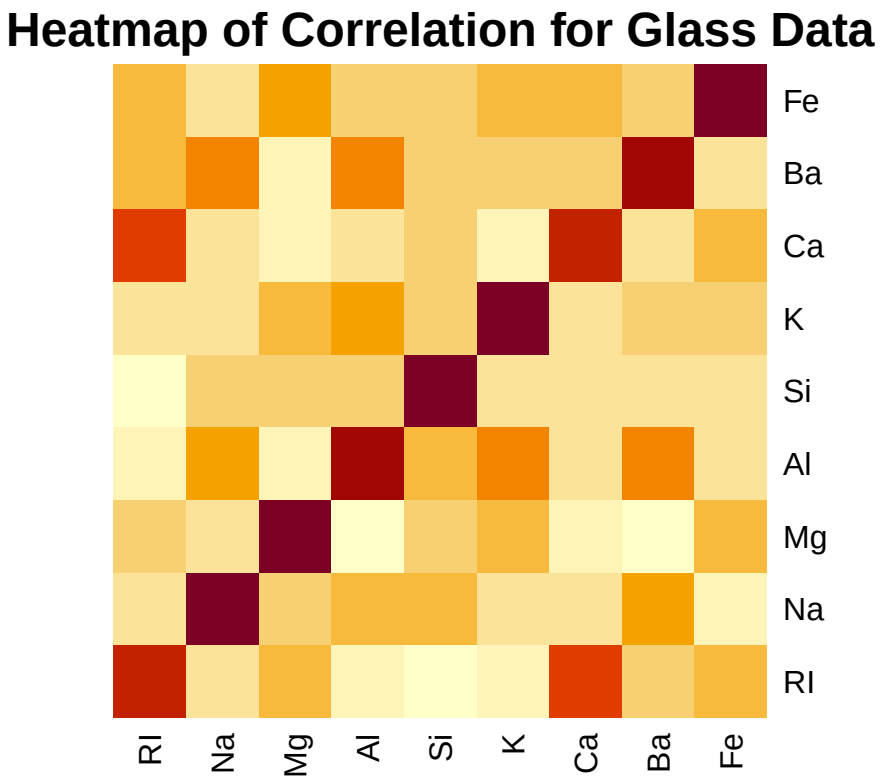
```
idMat[abs(idMat)<0.00000001] = 0 #force really small values to be zero so you can inspect
idMat
```

```
##   X1 X2 X3 X4 X5 X6 X7 X8 X9
## 1 -1  0  0  0  0  0  0  0  0
## 2  0 -1  0  0  0  0  0  0  0
## 3  0  0 -1  0  0  0  0  0  0
## 4  0  0  0  1  0  0  0  0  0
## 5  0  0  0  0 -1  0  0  0  0
## 6  0  0  0  0  0 -1  0  0  0
## 7  0  0  0  0  0  0 -1  0  0
## 8  0  0  0  0  0  0  0 -1  0
## 9  0  0  0  0  0  0  0  0  1
```

We might be worried about the negative ones, but we recall that the inner product is negative one if the vectors point opposite each other but otherwise the same direction. So we do see that these two matrix have a orthonormal column space.

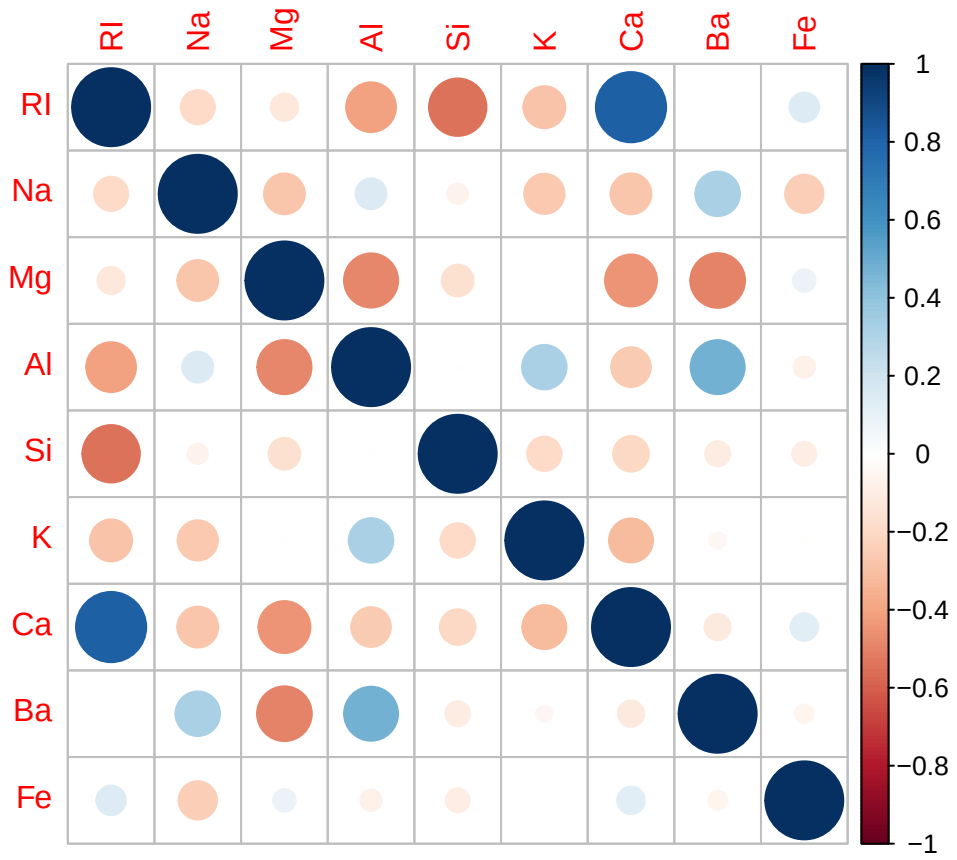
1.b.i.

```
heatmap(corMat, main = "Heatmap of Correlation for Glass Data", scale = "column", Rowv = NA, Colv = NA)
```



I tried for a while to get a scale for the colors on this graph and failed to find it. I am going to use the package suggestion of `corrplot` and of course it works great and has the scale that I wanted in the defaults. Yeah open source for the win!

```
corrplot(corMat)
```

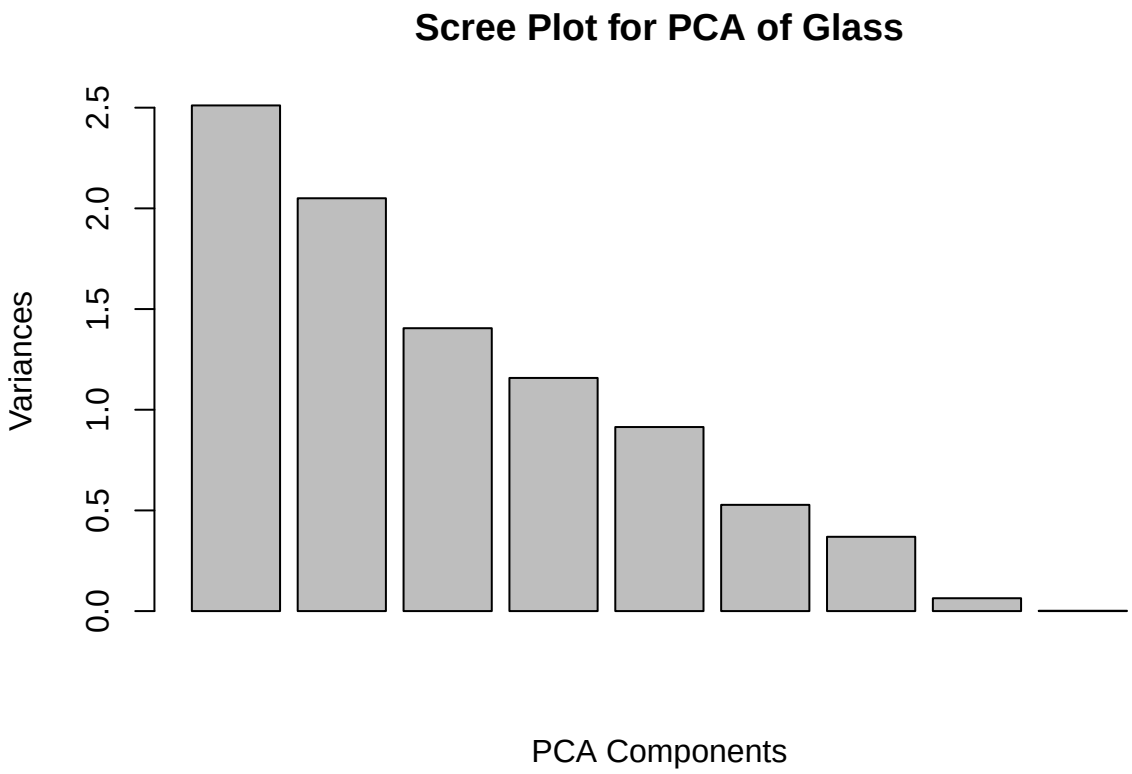


Yes there are two solutions here but I find it important to show that sometimes the base package just doesn't have what you want.

1.b.ii.

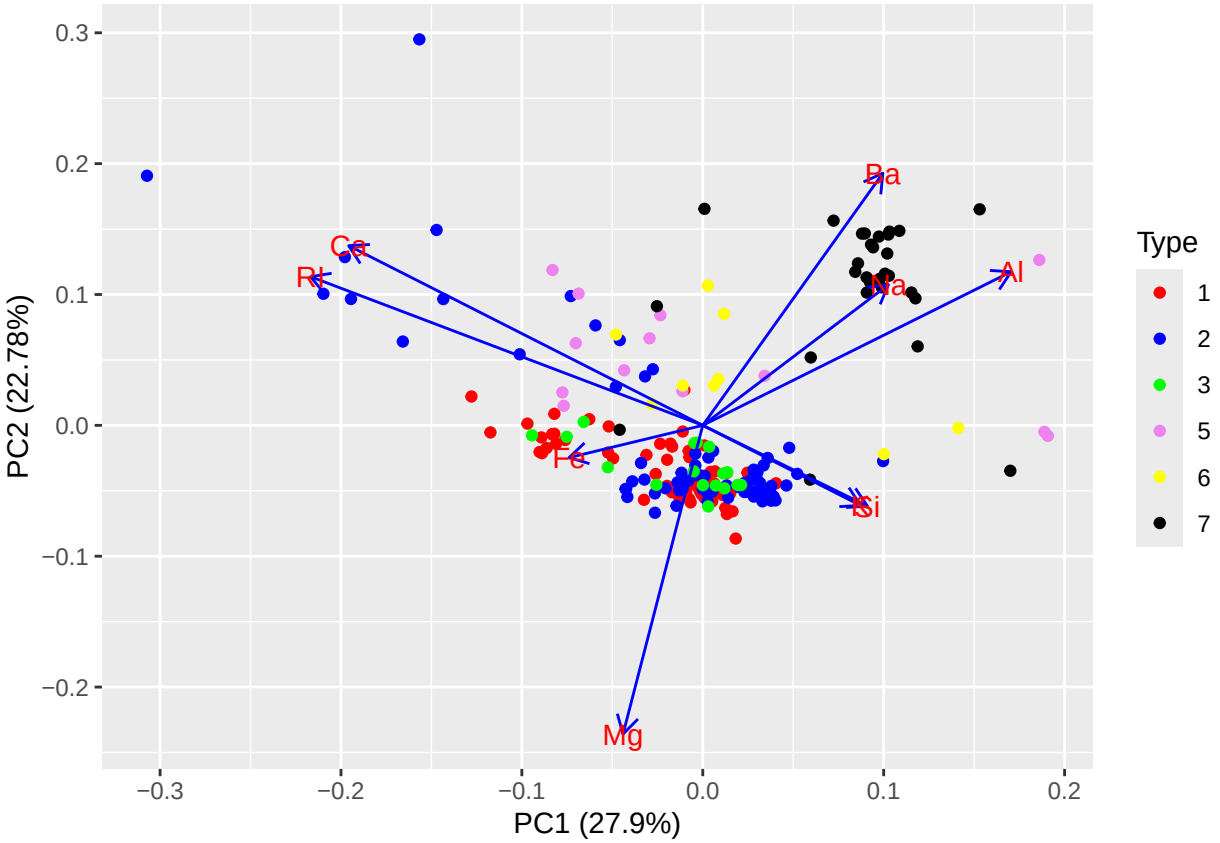
To look at the PCA, I'll start with the overloaded `plot` from the S3 method included with the `prcomp` function. The scree plot follows

```
plot(pMat, main = "Scree Plot for PCA of Glass", xlab = "PCA Components")
```



I don't quite see the hockey stick I expected. I also expected this to be normalized to one but here we see variances above 1 which is odd and not what I recalled in scree plots.

```
autoplot(pMat, data = Glass,
         color = "Type",
         loadings = TRUE,
         loadings.colour = 'blue',
         loadings.label = TRUE)+
  scale_color_manual(breaks = c("1","2","3","5","6","7"),
                     values =c("red","blue","green","violet","yellow","black"))
```



1.b.iii.

The first two components of the PCA provide the direction in the 9 dimensional space in which the most variance is explained. We see that 27.9% and 22.7% of the variance are explained in these two directions. We also see that iron (fe) and silicon (si) contribute negligibly to the first two PCAs. This first direction of the PCA is dominated by Al, Ca, and RI. The second component is dominated by Mg and Ba.

1.b.iv.

There is some value to this PCA but only covering about 50% of the variation has some drawbacks. The lack of the hockeystick in the scree plot also makes me hesitant to use this PCA. Besides all this, the dataset is relatively small dimensional at 9 dimensions. Reducing to two or three dimensions would allow for slightly faster computations but overall, I do not think it is worth it here only getting 50% of the variation. No, I would not suggest using the PCA in this case.

1.c.i. First, I will mean center the data.

```
preproc.param <- Glass[2:11] %>% preProcess(method = c("center","scale"))

transformed <- preproc.param %>% predict(Glass[2:11])
```

Now we will apply the LDA transformation on the transformed data.

```
ldaData <- lda(Type~., transformed)
ldaData
```

```
## Call:
## lda(Type ~ ., data = transformed)
##
## Prior probabilities of groups:
```

```
##          1          2          3          5          6          7
## 0.32710280 0.35514019 0.07943925 0.06074766 0.04205607 0.13551402
##
## Group means:
##            RI         Na         Mg          Al          Si           K
## 1  0.11619394 -0.2027480  0.6016993 -0.56292106 -0.04104563 -0.07609341
## 2  0.08335312 -0.3626484  0.2201683 -0.07360481 -0.06830876  0.03679371
## 3 -0.13233756  0.0357681  0.5955297 -0.48817322 -0.31790076 -0.13889393
## 5  0.18514882 -0.7104526 -1.3246507  1.17960228 -0.36767449  1.49180633
## 6 -0.29960679  1.5170350 -0.9560244 -0.15670866  0.71749416 -0.76213169
## 7 -0.41134992  1.2664879 -1.4879681  1.35768734  0.40659635 -0.26354770
##            Ca         Ba            Fe
## 1 -0.11219935 -0.3264806 -0.0000959146
## 2  0.08201617 -0.2509629  0.2332491727
## 3 -0.12227876 -0.3343056  0.0005077832
## 5  0.81992810  0.0254326  0.0385871830
## 6  0.28085801 -0.3520514 -0.5850790848
## 7 -0.32714905  1.7395812 -0.4470612770
##
## Coefficients of linear discriminants:
##            LD1        LD2       LD3          LD4        LD5
## RI -0.94656386 0.08925658 1.0811807 -0.749671704 -2.4436288
## Na -1.94450927 2.58461557 0.3753751 -5.654449980  1.9588285
## Mg -1.06793259 4.30684515 2.2687400 -9.880261458  4.0391676
## Al -1.66643308 0.86111013 1.0996248 -3.204929873  0.4678828
## Si -1.89891675 2.32855629 1.3187565 -5.841781630  0.7406973
## K  -1.02491652 1.21439159 0.8387922 -5.267175041  1.8398285
## Ca -1.43213378 3.37701884 0.9215204 -9.530340500  5.2814448
## Ba -1.15061274 1.71202472 1.2910288 -3.201342577  2.1915962
## Fe  0.04983573 0.02110900 0.1171805  0.004360319 -0.1269549
##
## Proportion of trace:
##    LD1    LD2    LD3    LD4    LD5
## 0.8145 0.1169 0.0413 0.0163 0.0111
```

1.c.ii. We see that the first LD1 projection picks about 81.45% of the differences of the means. We see that Na, Al, Si and Ca are the most important variables in our determining which class the class belongs to. We can use the following formula to make our predictions with the LDA1.

$$0.94 * RI + 1.94 * Na + 1.06 * Mg + 1.66 * Al + 1.90 * Si + 1.02 * K + 1.43 * Ca + 1.15 * Ba - .05 * Fe$$

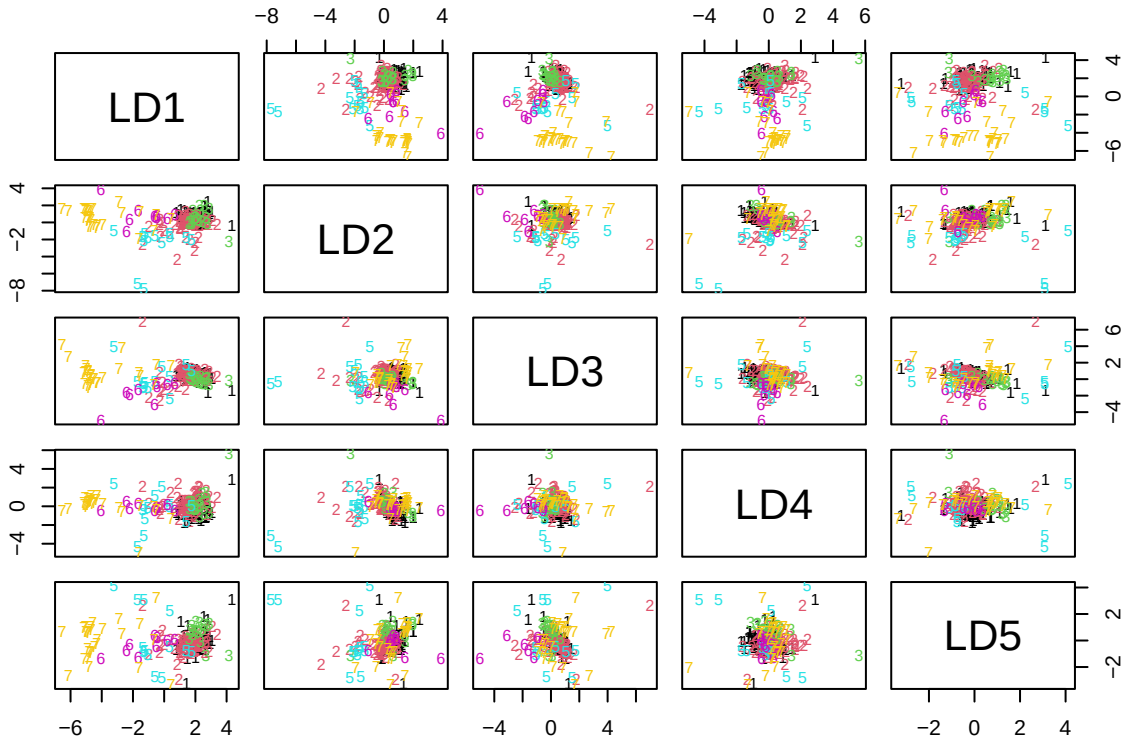
1.c.iii.

We make our predictions here

```
values <- predict(ldaData, transformed)
#values$x[,1]
#ldahist(data = values$x[,1], g = Glass$Type)
```

This is the overloaded plot for the lda. I am always curious to see these for new methods I use.

```
plot(ldaData, col = as.numeric(Glass$Type))
```



A confusion table to check out the results

```
table(Orignial = Glass$Type, Predicted = values$class)
```

```
##          Predicted
## Orignial  1  2  3  5  6  7
##        1 52 15  3  0  0  0
##        2 17 54  0  3  2  0
##        3 11  6  0  0  0  0
##        5  0  5  0  7  0  1
##        6  1  2  0  0  6  0
##        7  1  2  0  1  0 25
```

We see in our confusion table that we did well but not perfect. Let's see what it actually did

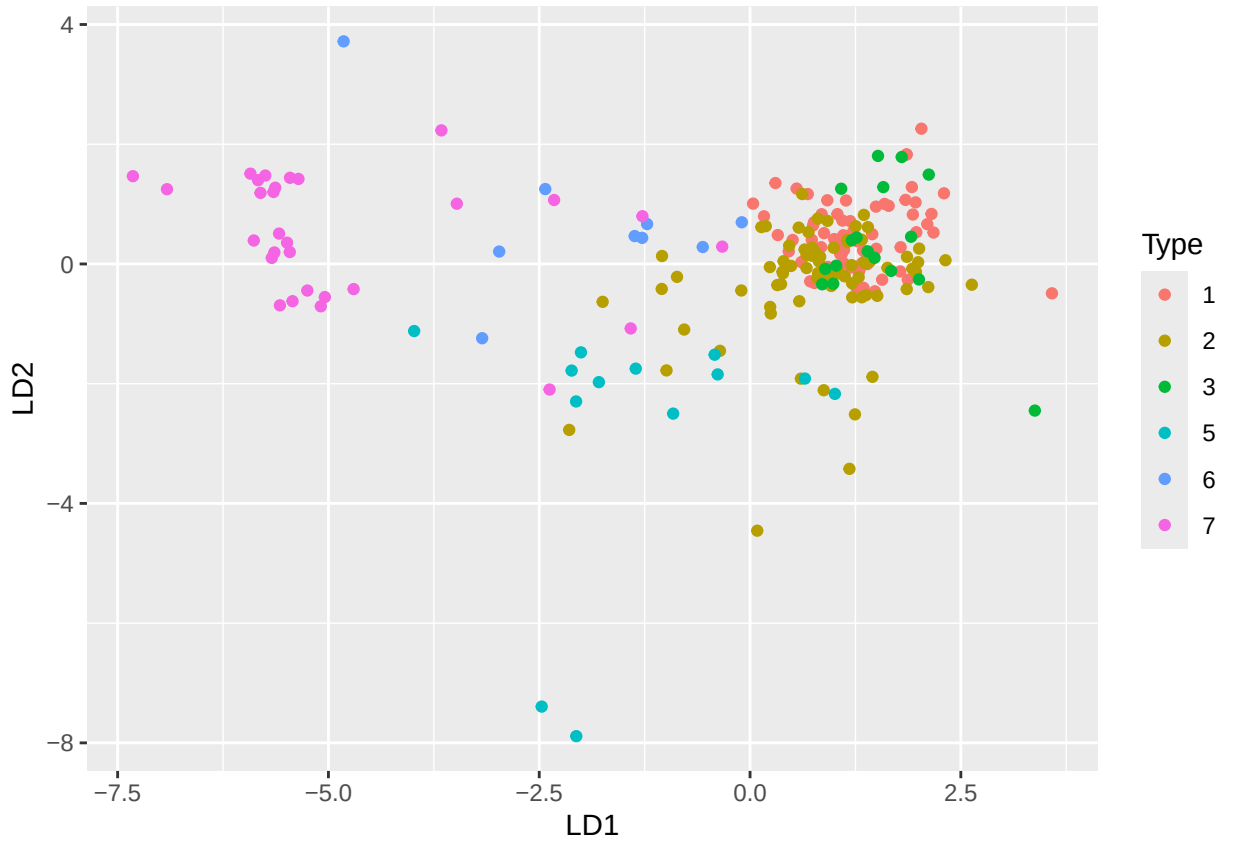
```
mean(values$class == Glass$Type)
```

```
## [1] 0.6728972
```

Correct about two thirds of the time, much better than guessing!

A scatterplot of the LD1 and LD2 with coloring.

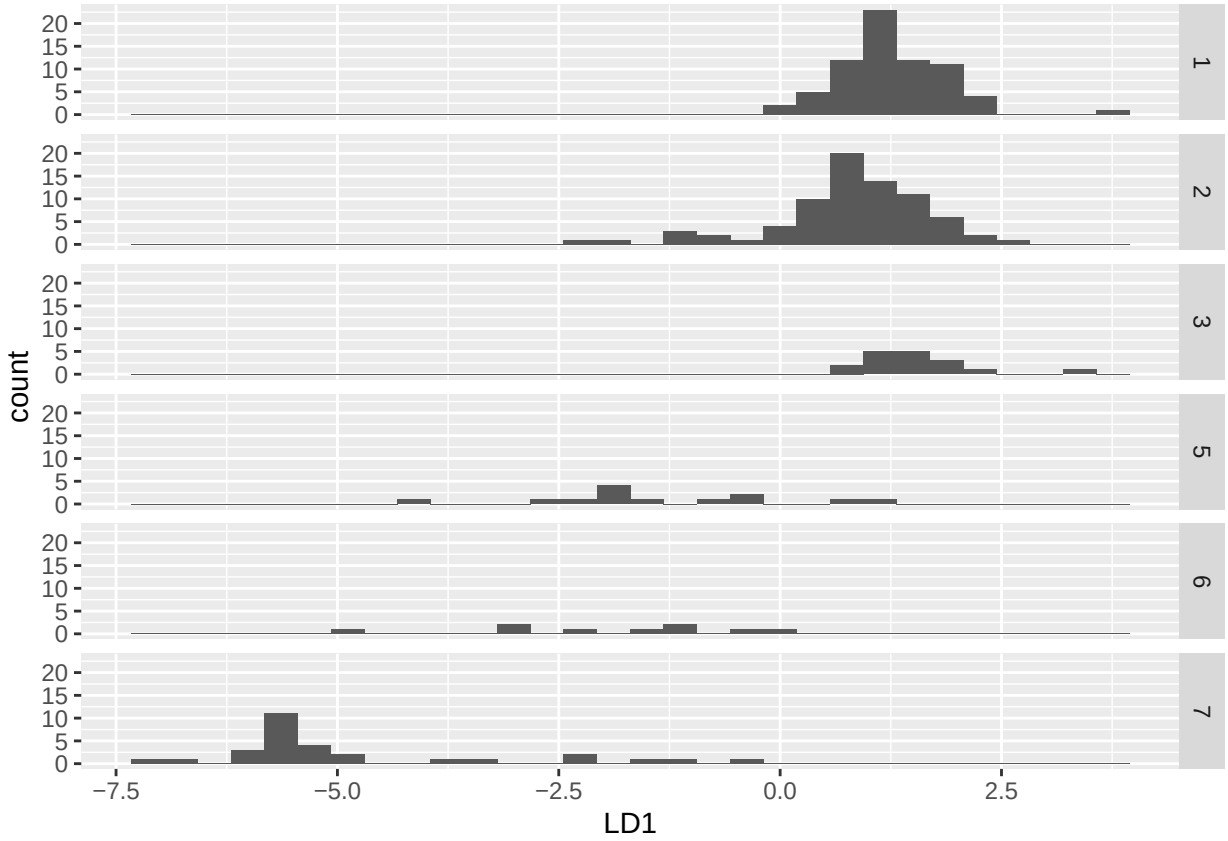
```
lda.data  <- cbind(transformed,predict(ldaData)$x)
ggplot(lda.data, aes(LD1,LD2)) +
  geom_point(aes(color = Type))
```



We do not see perfect separation and understand why our results are not a bit stronger. I could not get the command `ldahist` to compile on my machine. Instead I went ahead and used ggplot to create something equivalent.

```
ggplot(lda.data, aes(LD1)) +
  geom_histogram() +
  facet_grid(rows = "Type")
```

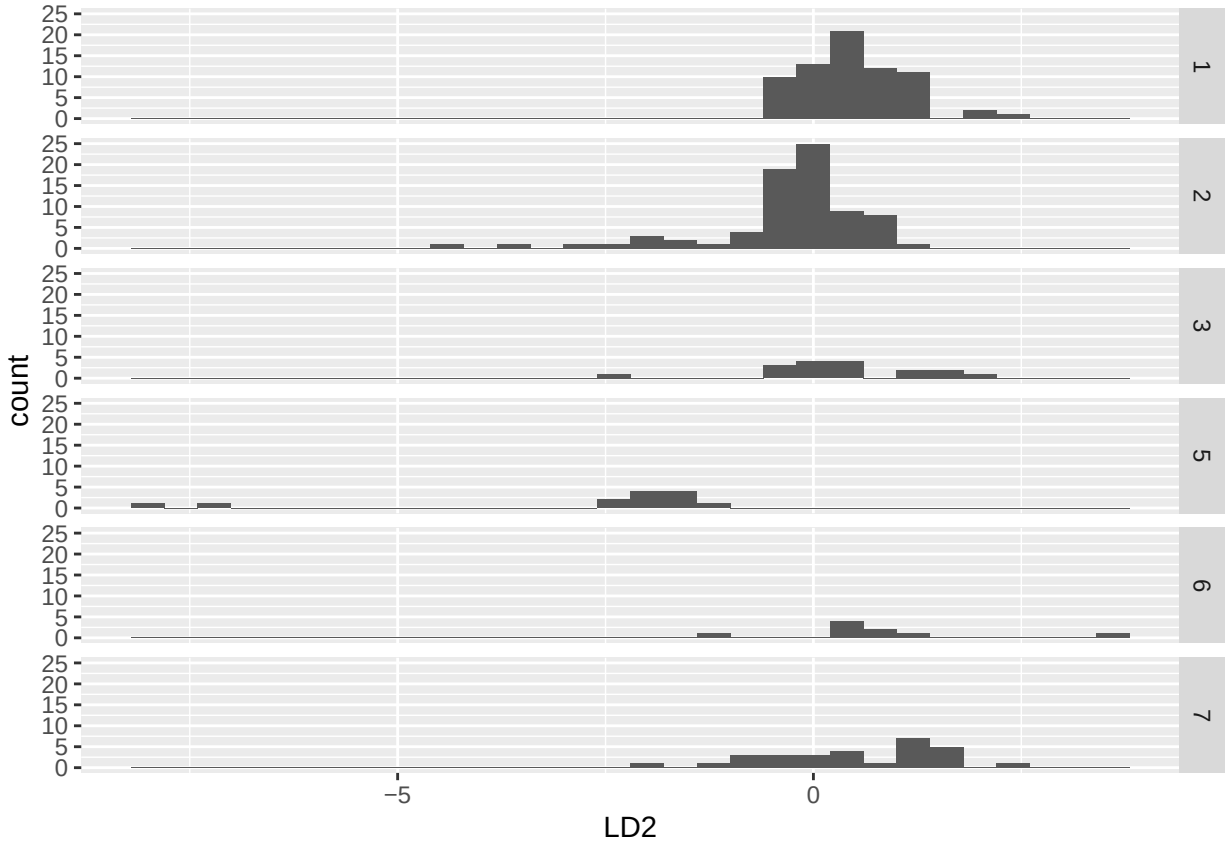
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We see some separation in the first LDA but below in the second, we see almost none.

```
ggplot(lda.data, aes(LD2)) +
  geom_histogram() +
  facet_grid(rows = "Type")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



2a.

```
# Load the data
data("heptathlon", package = "HSAUR2")

# Apply Grubbs' test to each event
outlier_tests <- apply(heptathlon[, 1:7], 2, function(x) grubbs.test(x))

# Print out the results
outlier_tests
```

```
## $hurdles
##
##  Grubbs test for one outlier
##
## data:  x
## G.Launa (PNG) = 3.5024, U = 0.4676, p-value = 0.000436
## alternative hypothesis: highest value 16.42 is an outlier
##
##
## $highjump
##
##  Grubbs test for one outlier
##
## data:  x
## G.Launa (PNG) = 3.61806, U = 0.43184, p-value = 0.0001698
## alternative hypothesis: lowest value 1.5 is an outlier
```

```
##
##
## $shot
##
##  Grubbs test for one outlier
##
## data:  x
## G.Hui-Ing (TAI) = 2.08971, U = 0.81047, p-value = 0.3702
## alternative hypothesis: lowest value 10 is an outlier
##
##
## $run200m
##
##  Grubbs test for one outlier
##
## data:  x
## G.Joyner-Kersee (USA) = 2.15480, U = 0.79847, p-value = 0.3048
## alternative hypothesis: lowest value 22.56 is an outlier
##
##
## $longjump
##
##  Grubbs test for one outlier
##
## data:  x
## G.Launa (PNG) = 2.68319, U = 0.68752, p-value = 0.04594
## alternative hypothesis: lowest value 4.88 is an outlier
##
##
## $javelin
##
##  Grubbs test for one outlier
##
## data:  x
## G.Scheider (SWI) = 1.69718, U = 0.87498, p-value = 1
## alternative hypothesis: highest value 47.5 is an outlier
##
##
## $run800m
##
##  Grubbs test for one outlier
##
## data:  x
## G.Launa (PNG) = 3.30186, U = 0.52681, p-value = 0.001808
## alternative hypothesis: highest value 163.43 is an outlier
```

According to this test, the competitor Launa (PNG) is identified as an outlier in multiple events:

1. Hurdles: Highest value 16.42 is an outlier, with a p-value of 0.000436 (very strong evidence).
2. High jump: Lowest value 1.5 is an outlier, with a p-value of 0.0001698 (very strong evidence).
3. Long jump: Lowest value 4.88 is an outlier, with a p-value of 0.04594 (moderate evidence).
4. 800m run: Highest value 163.43 is an outlier, with a p-value of 0.001808 (strong evidence).

Remove Launa (PNG) from the dataset as she is consistently identified as an outlier across multiple events.

```r
heptathlon_cleaned <- heptathlon[-which(rownames(heptathlon) == "Launa (PNG)"), ]
```

2b.

```r
# Identify the columns for the running events
running_events <- c("hurdles", "run200m", "run800m")

# Apply the transformation: subtract each value from the maximum value in its column
heptathlon_cleaned[running_events] <- apply(
  heptathlon_cleaned[running_events],2,
  function(x) max(x) - x)
```

By transforming all the running events by subtracting each athlete's time from the maximum recorded time in the event, we can more easily compare performance across events where higher values consistently represent better outcomes.

```r
# Check the transformed data for running events
head(heptathlon_cleaned[running_events])
```

```
##                     hurdles run200m run800m
## Joyner-Kersee (USA)    2.16    4.05   18.16
## John (GDR)             2.00    2.96   20.55
## Behmer (GDR)           1.65    3.51   22.47
## Sablovskaite (URS)     1.24    2.69   14.43
## Choubenkova (URS)      1.34    2.68   18.77
## Schulz (GDR)           1.10    1.96   20.88
```

2c.

```r
# Perform PCA on the 7 event results
Hpca <- prcomp(heptathlon_cleaned[, 1:7], scale. = TRUE)

# View a summary of the PCA results
summary(Hpca)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6     PC7
## Standard deviation     2.0793 0.9482 0.9109 0.68320 0.54619 0.33745 0.26204
## Proportion of Variance 0.6177 0.1284 0.1185 0.06668 0.04262 0.01627 0.00981
## Cumulative Proportion  0.6177 0.7461 0.8646 0.93131 0.97392 0.99019 1.00000
```

```r
# Check the rotation matrix (loadings)
Hpca$rotation
```

```
##                 PC1         PC2        PC3         PC4         PC5         PC6
## hurdles  -0.4503876  0.05772161  0.1739345 -0.04840598 -0.19889364  0.84665086
## highjump -0.3145115 -0.65133162  0.2088272  0.55694554  0.07076358 -0.09007544
## shot     -0.4024884 -0.02202088  0.1534709 -0.54826705  0.67166466 -0.09886359
## run200m  -0.4270860  0.18502783 -0.1301287 -0.23095946 -0.61781764 -0.33279359
## longjump -0.4509639 -0.02492486  0.2697589  0.01468275 -0.12151793 -0.38294411
## javelin  -0.2423079 -0.32572229 -0.8806995 -0.06024757  0.07874396  0.07193437
## run800m  -0.3029068  0.65650503 -0.1930020  0.57418128  0.31880178 -0.05217664
##                  PC7
## hurdles  -0.06961672
## highjump  0.33155910
## shot      0.22904298
## run200m   0.46971934
```

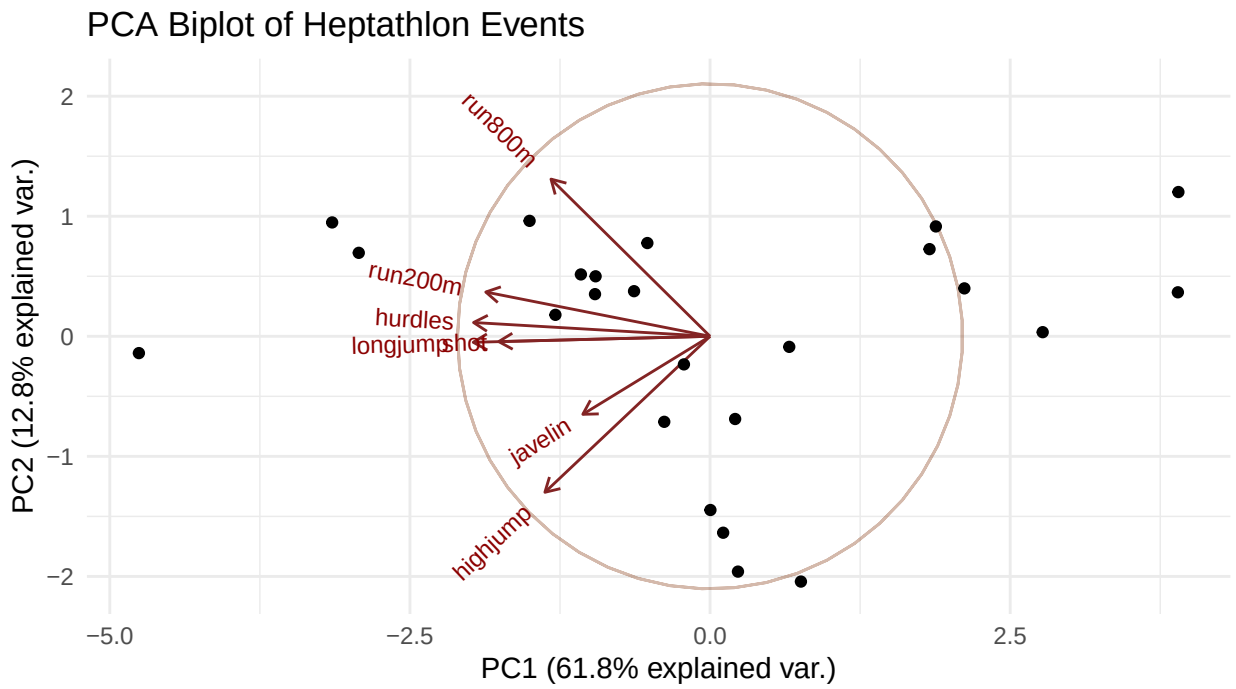
```
## longjump -0.74940781
## javelin  -0.21108138
## run800m   0.07718616
```

```
# Check the PCA scores (principal components)
head(Hpca$x)
```

```
##                            PC1        PC2          PC3        PC4        PC5
## Joyner-Kersee (USA) -4.7575302 -0.1398614  0.006040526 -0.2934163 -0.3618331
## John (GDR)          -3.1479434  0.9485903  0.243919842 -0.5491714  0.7536446
## Behmer (GDR)        -2.9261848  0.6953424 -0.622293440  0.5547449 -0.1903504
## Sablovskaite (URS)  -1.2881355  0.1790071 -0.250632380 -0.6371742  0.6036215
## Choubenkova (URS)   -1.5034510  0.9617733 -1.780588549 -0.7840353  0.5896995
## Schulz (GDR)        -0.9584671  0.3512164 -0.413086366  1.1135469  0.7148389
##                            PC6         PC7
## Joyner-Kersee (USA) -0.27050283 -0.47587527
## John (GDR)           0.37770017 -0.05172711
## Behmer (GDR)        -0.25780287  0.11054960
## Sablovskaite (URS)  -0.21575716  0.53075152
## Choubenkova (URS)    0.08014332 -0.30081842
## Schulz (GDR)        -0.25436956  0.03838796
```

2d.

```
# Visualize the first two principal components using ggbiplot
ggbiplot(Hpca, obs.scale = 1, var.scale = 1,
         circle = TRUE) +
  ggtitle("PCA Biplot of Heptathlon Events") +
  theme_minimal() +
  theme(legend.position = "bottom")
```



PC1 (explains 61.8% of the variance): Separates competitors based on running events (800m, 200m, hurdles). Athletes with higher PC1 scores performed better in running events, while those with lower PC1 scores did better in field events (high jump, javelin).

PC2 (explains 12.8% of the variance): Mainly reflects performance in the high jump. Athletes with higher PC2 scores excelled in high jump.

Event Correlation:

1. Running events (800m, 200m, hurdles) are positively related, as their arrows point in similar directions.
2. Field events (high jump, javelin) are negatively related to running events, meaning athletes who performed well in running events tended to perform worse in field events.

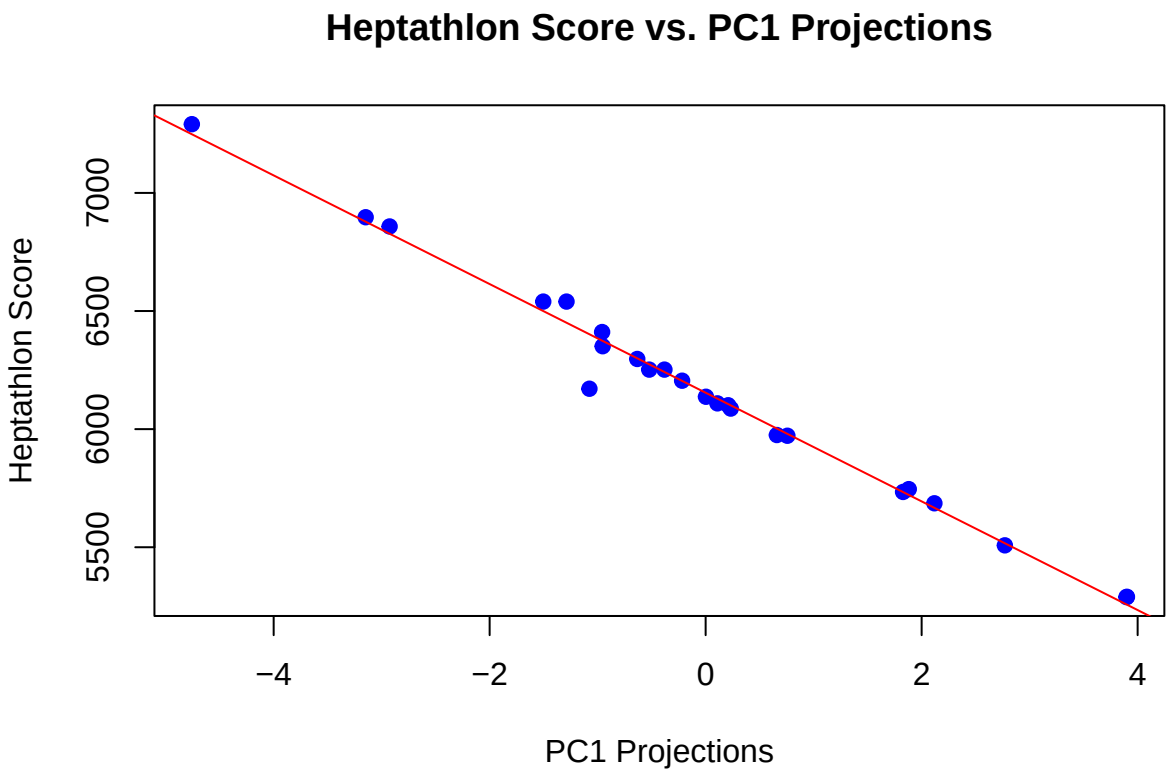
Competitors: Athletes farther from the center of the plot showed more extreme performances (either very good or bad in certain events), while those closer to the center had more balanced performances across all events.

2e.

```
# Assuming the heptathlon score is stored in a column called 'score' in the dataset
# Replace 'score' with the actual column name if different
heptathlon_scores <- heptathlon_cleaned$score

# Plot heptathlon score vs. PC1 projections
plot(Hpca$x[,1], heptathlon_scores,
     xlab = "PC1 Projections",
     ylab = "Heptathlon Score",
     main = "Heptathlon Score vs. PC1 Projections",
     pch = 19, col = "blue")
```

```r
# Add a linear regression line to see the relationship
abline(lm(heptathlon_scores ~ Hpca$x[,1]), col = "red")
```



The plot shows a negative linear relationship between the heptathlon score and PC1 projections. Since PC1 primarily reflects performance in running events, higher PC1 values indicate stronger performance in these events. The negative slope of the regression line suggests that competitors with lower PC1 projections tend to have higher overall heptathlon scores. This indicates that excelling in running events plays a significant role in achieving a higher overall score. Conversely, competitors with higher PC1 values tend to have lower total scores, meaning weaker performance in running events is associated with a lower heptathlon score.

In summary, the plot highlights that performance in running events is a key factor in determining a competitor's overall heptathlon success.

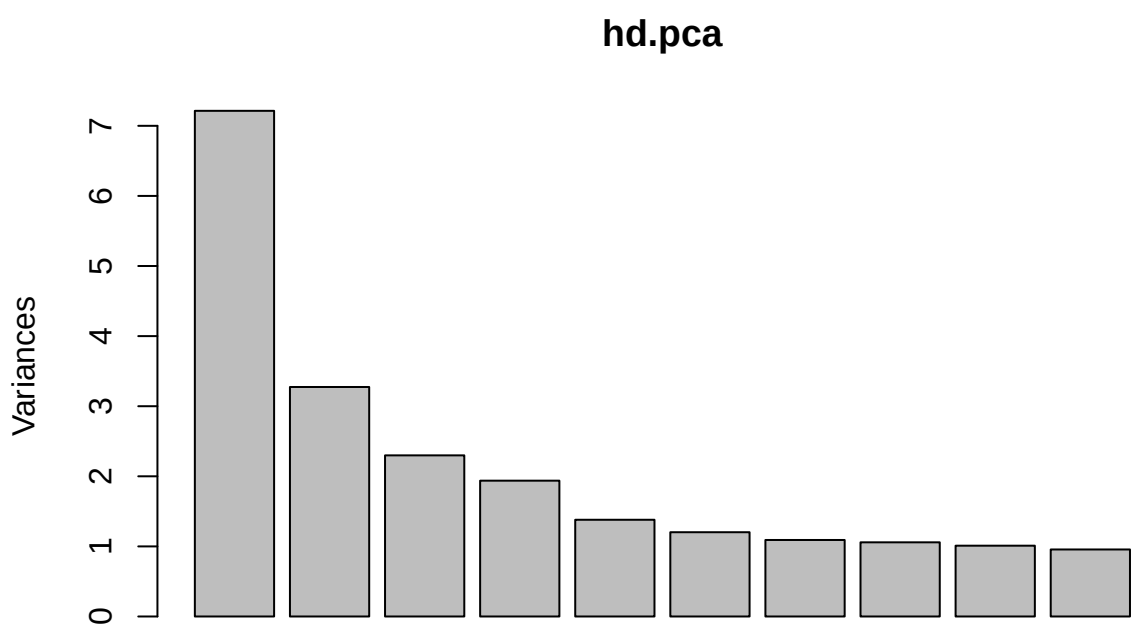
3.

Load up the data and get it going...

```r
housingData = read.csv("housingData-1.csv")
hd <- housingData %>%
  select_if(is.numeric) %>%
  dplyr::mutate(age = YrSold - YearBuilt,
                ageSinceRemodel = YrSold - YearRemodAdd,
                ageofGarage = ifelse(is.na(GarageYrBlt), age, YrSold - GarageYrBlt)) %>%
  dplyr::select(!c(Id,MSSubClass, LotFrontage, GarageYrBlt,
                   MiscVal, YrSold , MoSold, YearBuilt,
                   YearRemodAdd, MasVnrArea))
```

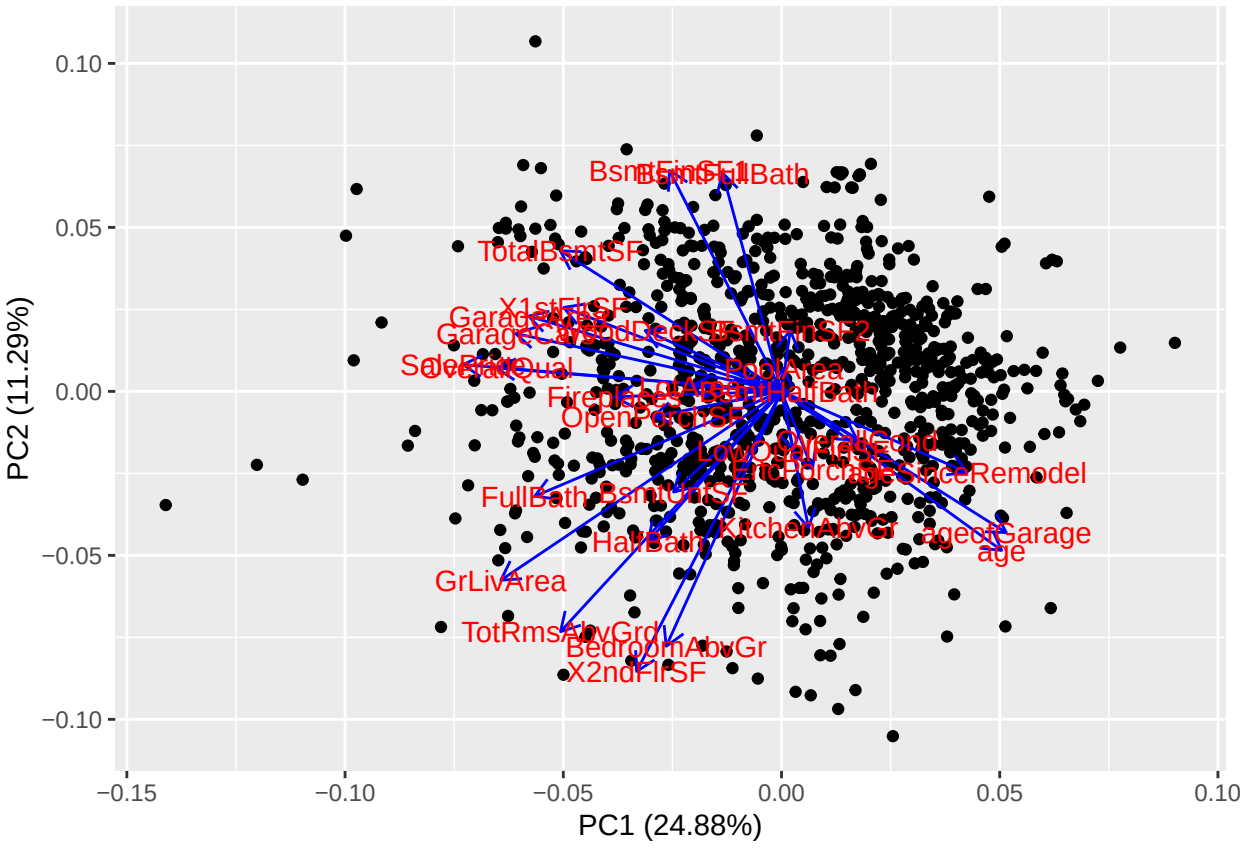
Next we will preform the PCA of the scaled data.

```
hd.pca <- prcomp(hd,scale = TRUE)
plot(hd.pca)
```



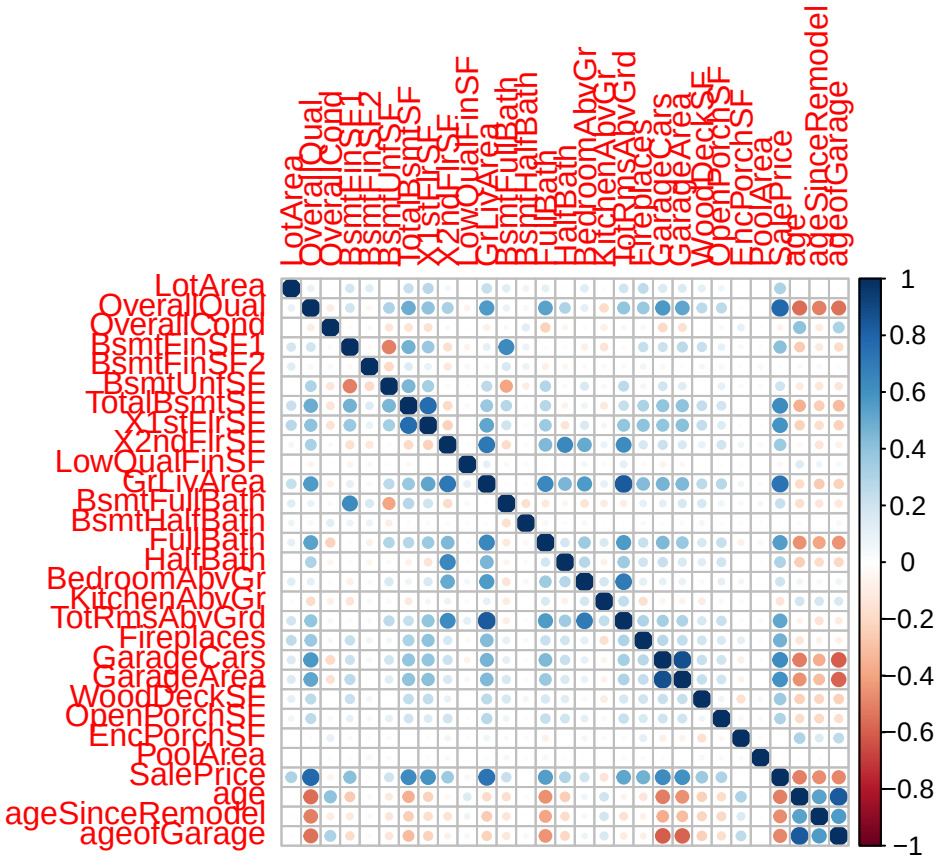
The scree plot shows that there is a lot of information stored in that first principle component. Let's look at the vectors and see what contributes the most to the principle components.

```
autoplot(hd.pca, data = hd,
         loadings = TRUE,
         loadings.colour = 'blue',
         loadings.label = TRUE)
```



That is way too busy of a graph to glean much info. Let's look to the correlations and see what we can get from that.

```
cor(hd) %>% corrplot()
```


LotArea
OverallQual
OverallCond
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
TotalBsmtSF
X1stFlrSF
X2ndFlrSF
LowQualFinSF
GrLivArea
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageCars
GarageArea
WoodDeckSF
OpenPorchSF
EncPorchSF
PoolArea
SalePrice
age
ageSinceRemodel
ageofGarage
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1