Final Project

Ansh Gupta

August 11, 2022

Establishing reliable biomarkers for assessing and validating clinical diagnosis at early stages of Parkinson's disease is crucial for developing therapies to slow or halt disease progression. This data set uses whole blood gene expression profiling from over 500 individuals where we will attempt to find a gene signature. This repository contains the gene expression profiles collected in the GENEPARK consortium. The main study sought a classifier for IPD. These data contain 233 healthy controls, 205 IPD patients, and 48 patients with other neurodegenerative diseases (NDD). Other samples are available in the data and can be used for additional analyses. The largest class of these additional samples are 22 samples from genetic unaffected controls and 41 genetic PD patients.

Note: the original study which uploaded this data to NIH Geo is not yet published.

Data Wrangling

Let's start by loading in our data sets. Download these from the canvas site, and make a new folder for R bootcamp. We'll switch to this directory here.

The tinyTex package will allow you to actually knit .pdf documents from RMarkdown:

Note that we have both a phenotype file, as well as a file which includes the normalized and log transformed expression values. We can use the read.csv function to load in these files.

```
# loading the data
expr = read.csv("simulatedData.csv")
pheno = read.csv("parkPheno.csv")
```

We should start by summarizing both these files. Try the following functions: head(), and View(). Note that while the dimensions on our phenotype file are reasonable, we have 552 columns in our expression file. Just summarize the first 10 columns of this file.

```
# your code here
#commenting these out for ease of viewing in the PDF file
#head(pheno)
#View(pheno)
#head(expr)
#View(expr)
```

Try summarizing the phenotype data:

```
# your code here
summary(pheno[,1:10])
```

```
##  geo_accession      submission_date    last_update_date       type
##  Length:550         Length:550         Length:550         Length:550
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##     tissue            organism          subject_id        disease_label
##  Length:550         Length:550         Length:550         Length:550
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##      sex            mutated_pd_genes
##  Length:550         Length:550
##  Class :character   Class :character
##  Mode  :character   Mode  :character
```

We make the following observations.

1. We have some unnecessary data in this file. We aren't interested in the submission and last update date. We can reduce the dimensions of this file so it handles nicer from now on.
2. We have a LOT of missing data. You'll learn how to handle this in some of your biostats classes! For now, we'll run what analyses we can given the data we have.
3. Some of our scores have been read in as character values (and they should be numbers). If you investigate this further, you'll find that some values have been recorded as "ND", which we'll assume means "no data". We will need to record these as NA values in R.

Our next step is to address item one. We will reduce the dimensions of our pheno data frame to include only that information that we're interested in modelling. We can exclude the dates, type (as it's all RNA), tissue (all whole blood), organism (all homo sapiens), and subject ID (we will be using geo_accession as our unique indicator). As well, we will exclude mutated_pd_genes, as we indend to define our own gene signature later this week.

Subset your pheno data frame to include columns 1,8,9,11:20.

```
# your code here
pheno=pheno[,c(1,8,9,11:20)]
head(pheno)
```

```
##   geo_accession disease_label     sex age_at_exam age_at_symptoms updrs
## 1    GSM2631171   ATYPICAL_PD    Male          NA              53     1
## 2    GSM2631309   ATYPICAL_PD    Male          NA              64     0
## 3    GSM2631219   ATYPICAL_PD    Male          NA              NA     0
## 4    GSM2630775           CBD  Female          NA              60     0
## 5    GSM2631147           CBD  Female          NA              66     0
## 6    GSM2630853       CONTROL    Male          NA              41     0
##   updrs_ii updrs_iii_score_on updrs_iii_score_off updrs_iv hoehn_yahr_on
## 1        4                19                    0        0            2
## 2        0                 0                    0        0            9
## 3        0                 0                    0        0            0
## 4        0                 0                    0        0            9
## 5        0                30                    0        0            9
## 6        0                 1                    0        0            8
##   hoehn_yahr_off moca_score
## 1              0         21
## 2              0          0
```

```
## 3              0          0
## 4              0          0
## 5              0          0
## 6              0         30
```

Next we need to correct the columns which contain "ND". You can use the "which" function to find the index of of the matrices which are "ND", and then set these to NA. Set columns 8,9,11,12,13 to numeric values using the "as.numeric" function inside a "sapply" loop. Run a summary of the data frame again.

```
#your code here
ix=which(pheno==" ND",arr.ind = T) #storing the indices
pheno[ix] = NA #changing "ND" to NA
j=c(8,9,11,12,13) #storing column numbers
pheno[,j] = sapply(unlist(pheno[,j]),as.numeric) #changing those column numbers to have numeric vals
summary(pheno)
```

```
##  geo_accession      disease_label          sex             age_at_exam
##  Length:550         Length:550         Length:550         Min.   :30.00
##  Class :character   Class :character   Class :character   1st Qu.:54.75
##  Mode  :character   Mode  :character   Mode  :character   Median :61.00
##                                                           Mean   :60.56
##                                                           3rd Qu.:68.25
##                                                           Max.   :82.00
##                                                           NA's   :266
##  age_at_symptoms     updrs           updrs_ii      updrs_iii_score_on
##  Min.   :10.00   Min.   : 0.000   Min.   : 0.000   Min.   : 0.0
##  1st Qu.:45.00   1st Qu.: 0.000   1st Qu.: 0.000   1st Qu.: 0.0
##  Median :55.00   Median : 0.000   Median : 0.000   Median : 0.5
##  Mean   :53.61   Mean   : 1.171   Mean   : 4.593   Mean   : 9.0
##  3rd Qu.:64.00   3rd Qu.: 2.000   3rd Qu.: 7.000   3rd Qu.:16.0
##  Max.   :78.00   Max.   :36.000   Max.   :35.000   Max.   :75.0
##  NA's   :325     NA's   :122      NA's   :123      NA's   :154
##  updrs_iii_score_off    updrs_iv      hoehn_yahr_on   hoehn_yahr_off
##  Min.   : 0.000      Min.   : 0.000   Min.   :0.000   Min.   :0.0000
##  1st Qu.: 0.000      1st Qu.: 0.000   1st Qu.:0.000   1st Qu.:0.0000
##  Median : 0.000      Median : 0.000   Median :1.000   Median :0.0000
##  Mean   : 2.523      Mean   : 1.236   Mean   :2.487   Mean   :0.1961
##  3rd Qu.: 0.000      3rd Qu.: 1.000   3rd Qu.:3.000   3rd Qu.:0.0000
##  Max.   :64.000      Max.   :14.000   Max.   :9.000   Max.   :5.0000
##  NA's   :110         NA's   :118      NA's   :158     NA's   :109
##    moca_score
##  Min.   : 0.00
##  1st Qu.: 0.00
##  Median :26.00
##  Mean   :17.62
##  3rd Qu.:29.00
##  Max.   :30.00
##  NA's   :16
```

We have a LOT of missing values present in the data! As mentioned before, imputation of missing values is an entire field unto itself. While we won't be imputing data today, we are going to wrangle the above data to attempt to ameliorate some of these missing values.

To do this we will:

1. Combine our Age variables to be age_at_exam where known, but age_at_symptoms where that is observed without age at exam
2. Combine our updrs scores to be the average updrs
3. Combine our hoehn scores to be the average hoehn
4. Keep our moca score as is
5. Remove the old variables from our pheno dataset.

```
#part 1 - combining Age variables to AgeMaster col
pheno$AgeMaster = pheno$age_at_exam
pheno$AgeMaster[is.na(pheno$AgeMaster)] = pheno$age_at_symptoms[is.na(pheno$AgeMaster)]

#part 2 - combining updrs scores to overallUPDRS col
up = c(6:10)
pheno$overallUPDRS = rowMeans(pheno[,up])

#part 3 - combining Hoehn scores to overallHoehn col
hn = c(11,12)
pheno$overallHoehn = rowMeans(pheno[,hn])

#Part 4 & 5 - keeping moca score and removing old variables from pheno dataset
keep = c(1,2,3,13,14,15,16)
pheno = pheno[,keep]

#View(pheno)
```

As you can see, we have far fewer missing values to contend with!

Let's look at a summary of the first 10 columns of expression data set.

```
# your code here
summary(expr[,1:10])
```

```
##        X           GeneName           GSM2631171          GSM2631309
##  Min.   :    1   Length:20668       Min.   :-5.223788   Min.   :-6.09018
##  1st Qu.: 5168   Class :character   1st Qu.:-0.960423   1st Qu.:-0.92906
##  Median :10334   Mode  :character   Median :-0.004842   Median : 0.01385
##  Mean   :10334                      Mean   :-0.009648   Mean   : 0.01249
##  3rd Qu.:15501                      3rd Qu.: 0.953228   3rd Qu.: 0.95912
##  Max.   :20668                      Max.   : 5.766301   Max.   : 5.66627
##    GSM2631219         GSM2630775          GSM2631147         GSM2630853
##  Min.   :-6.39097   Min.   :-5.206869   Min.   :-5.27578   Min.   :-6.115736
##  1st Qu.:-0.97337   1st Qu.:-0.981831   1st Qu.:-0.96379   1st Qu.:-0.944666
##  Median :-0.01097   Median : 0.001772   Median : 0.01906   Median :-0.007942
##  Mean   :-0.00354   Mean   :-0.000010   Mean   : 0.00298   Mean   :-0.009892
##  3rd Qu.: 0.95324   3rd Qu.: 0.971013   3rd Qu.: 0.98545   3rd Qu.: 0.945826
##  Max.   : 6.56118   Max.   : 5.275719   Max.   : 5.18612   Max.   : 5.570111
##    GSM2630769          GSM2631196
##  Min.   :-5.608142   Min.   :-6.303044
##  1st Qu.:-0.968002   1st Qu.:-0.970730
##  Median :-0.001583   Median :-0.004689
##  Mean   : 0.014813   Mean   :-0.006484
##  3rd Qu.: 0.987677   3rd Qu.: 0.977216
##  Max.   : 5.591597   Max.   : 5.434250
```

We don't need the X1 variable - this is just remaining row labels in the csv file. Let's remove this variable.

```
#your code here
expr$X=NULL
```

We don't see any evidence of missing values in our summary, but we should check all of the columns (excluding the GeneName). You can check this with the "anyNA"" function.

```
# your code here
which(is.na(expr),arr.ind=T)
```

```
##        row col
## [1,] 20668   1
```

Let's identify how big this problem is, and where it occurs.

```
# your code here
expr=expr[-nrow(expr),]
```

So one of our gene names is NA! This isn't useful, so let's remove this row.

```
# your code here
# did this in the previous code chunk!
```

We should see if the unique identifiers in our two data sets match. Check for a perfect match using the "identical" function.

```
# your code here
identical(colnames(expr[,-1]),as.character(pheno[,1]))
```

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

So that we don't lose any work, let's clean up our workspace to include only our cleaned expression and pheno data sets, which we can reload later.

Exploratory Data Analysis

In this section we are going to explore some of the data we have, and maybe develop a diagnostic signature for Parkinson's disease.

First, load in your data from yesterday.

Let's re-examine our pheno data set with the summary function again.

```
summary(pheno)
```

```
##  geo_accession      disease_label          sex              moca_score
##  Length:550         Length:550         Length:550         Min.   : 0.00
##  Class :character   Class :character   Class :character   1st Qu.: 0.00
##  Mode  :character   Mode  :character   Mode  :character   Median :26.00
```

```
##                                                                 Mean   :17.62
##                                                                 3rd Qu.:29.00
##                                                                 Max.   :30.00
##                                                                 NA's   :16
##    AgeMaster      overallUPDRS     overallHoehn
##  Min.   :10.00   Min.   : 0.000   Min.   :0.000
##  1st Qu.:53.00   1st Qu.: 0.000   1st Qu.:0.000
##  Median :60.00   Median : 0.000   Median :0.000
##  Mean   :59.37   Mean   : 1.525   Mean   :1.364
##  3rd Qu.:67.00   3rd Qu.: 0.600   3rd Qu.:4.000
##  Max.   :82.00   Max.   :18.400   Max.   :4.500
##  NA's   :199     NA's   :264      NA's   :258
```

We need to further delve into our disease label in order to simplify some of this analysis. Attach your pheno data frame using the attach function, and then summarize the disease label vector.

```
attach(pheno)
summary(disease_label)
```

```
##    Length     Class      Mode
##       550 character character
```

Here we have the counts of all the diseases in our data set. If you look at the actual excel file (not the csv), I've put in a dictionary for these acronyms if you're curious. Here, our controls and our genetic unaffected are both considered to be healthy controls. Any label which contains PD is some subset of Parkison's Disease, and the other labels represent other neurological disorders. We need to make a variable which records a 1 for our cases, and a 0 for our controls. Here, since we are interested in a signature that distinguishes PD from our other disease, the other diseases are technically part of the control set.

Try to set your case control vector using the grep function to find the indicies which contain "PD". At the end, sum your case vector to check that it worked. Make another variable of the words "case" and "control"

```
# your code here
pdI=grep("PD",disease_label,value = F)
case=rep(0,length(disease_label))
case[pdI]=1
sum(case)
```

```
## [1] 251
```

```
##alternative
case=grepl("PD",disease_label)*1
sum(case)
```

```
## [1] 251
```

```
caseName = ifelse(case == 1, "case","control")
```

We need to find differentially expressed genes. You'll learn more about this later. For now, feel free to use some of my code. Start by downloading the limma package

```
## If using Windows, first go to https://cran.rstudio.com/bin/windows/Rtools/ and install the appropria
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
#BiocManager::install("limma")
library(limma)
```

We will use the following code. **Comment this code with your thoughts below.**

```
#subset our data for a training and test set
set.seed(2) #generating random numbers
prob = runif(ncol(expr)-2) #using a random uniform distribution
k = which(prob>=0.3333333) #selecting indices with probability >= 0.3333333 for training puposes
eset = expr[,2:ncol(expr)] #subsetting all cols except the first col for expr into eset
eset = eset[,k] #keeping indices with prob >= 0.3333333 in the eset subset
rownames(eset) = expr[,1] #setting rownames of the subset same as expr
design = model.matrix(~0+as.factor(case[k])) #creating a design/model matrix
fit = eBayes(lmFit(eset,design)) #using empirical bayesian model for differential expression
topTable(fit, coef=2) #getting a table of the top ranked genes
```

```
##                logFC   AveExpr         t       P.Value     adj.P.Val        B
## EXOC3L4     3.142306  1.471867  29.51102 4.199361e-149 4.759985e-145 329.8083
## FAM132A    -3.159535 -1.461491 -29.50594 4.606362e-149 4.759985e-145 329.7162
## MDM2        3.127701  1.426391  29.45147 1.239815e-148 8.541085e-145 328.7308
## CCR3       -3.145072 -1.547047 -29.41096 2.588618e-148 1.337474e-144 327.9981
## MYO9A      -3.172079 -1.571136 -29.27526 3.042541e-147 1.257604e-143 325.5456
## GADD45GIP1 -3.086818 -1.391784 -29.09131 8.553065e-146 2.946103e-142 322.2251
## ANXA2      -3.106348 -1.517455 -29.04409 2.011963e-145 5.940178e-142 321.3737
## CCNJ       -3.123721 -1.454289 -28.99841 4.602051e-145 1.188882e-141 320.5503
## EMC6        3.070413  1.590585  28.97850 6.599845e-145 1.515544e-141 320.1914
## GEMIN4      3.100428  1.427101  28.89106 3.211859e-144 6.637949e-141 318.6165
```

```
results = topTable(fit, coef=2, number=Inf) #storing the top ranked genes in results
```

Here, we have our gene names, our log fold change for expression, average expression, t statistic, pvalue, adjusted pvalue (for multiple testing!!), and the log odds of differential expression.

Next, we select those genes that have adjusted p-values below 0.001. **Comment the code with your thoughts about what its doing below.**

```
#using the false discovery rate to get the rate of type 1 errors in the null hypothesis by selecting th
selected  = row.names(results)[p.adjust(results$P.Value, method="fdr")<0.001]
direction = sign(results$logFC) #storing the signs as 1,0,-1 for numbers in logFC col
esetSel = eset[selected, ] #subsetting rows with adjusted p-values below 0.001
nrow(esetSel) #checking number of genes in subset
```

```
## [1] 175
```

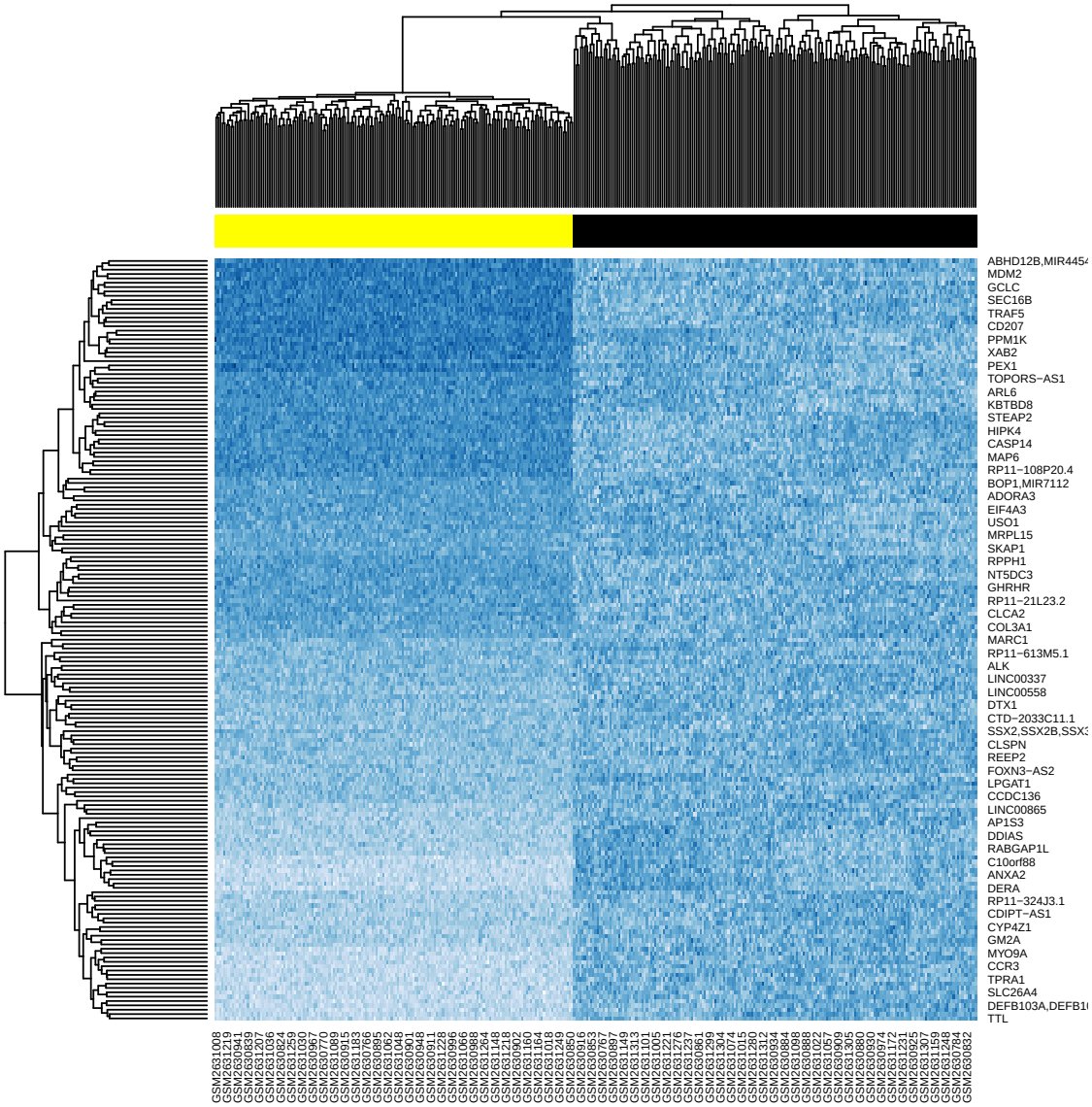
Okay! So we're now looking at just 175 genes!

We are going to make a heat map here. I've provided the code, but **try changing colours, labels, etc. to make it your own.**

```
library("RColorBrewer")
patientcolors <-ifelse(case[k]==1,"yellow","black") #cases are yellow, and controls are black
col <- colorRampPalette(brewer.pal(27, "Blues"))(256)
```

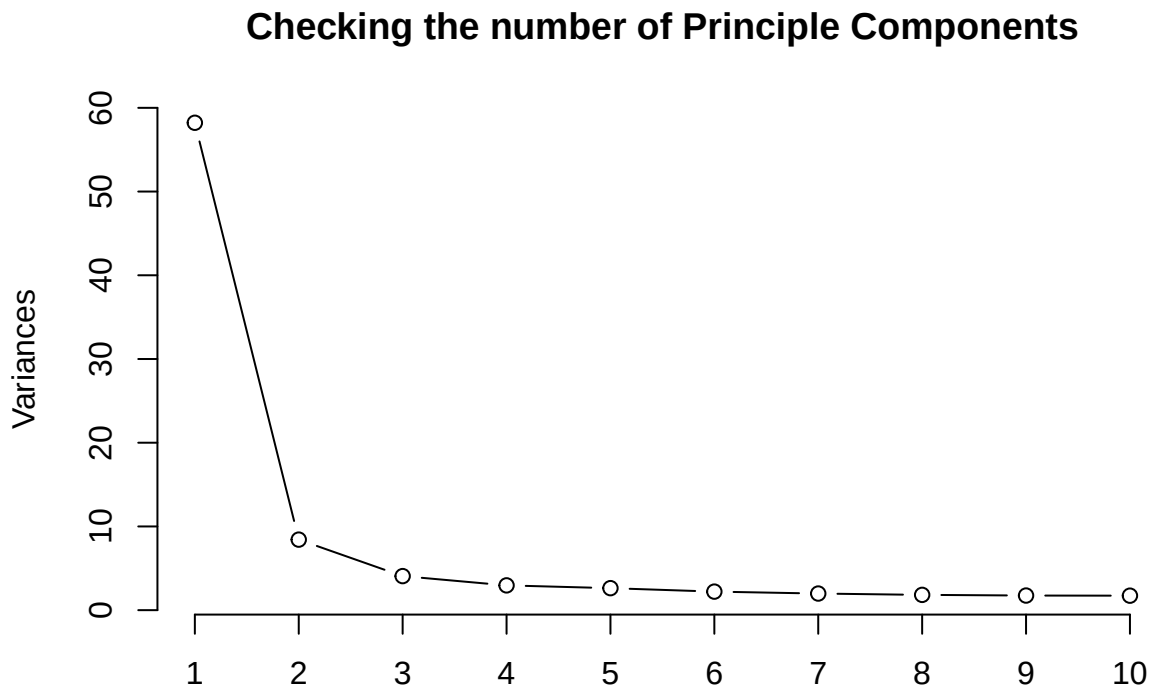
```
## Warning in brewer.pal(27, "Blues"): n too large, allowed maximum for palette Blues is 9
## Returning the palette you asked for with that many colors
```

```
heatmap(as.matrix(esetSel), scale = "none", col =  col, ColSideColors = patientcolors, distfun = functi
```


ABHD12B,MIR445
MDM2
GCLC
SEC16B
TRAF5
CD207
PPM1K
XAB2
PEX1
TOPORS−AS1
ARL6
KBTBD8
STEAP2
HIPK4
CASP14
MAP6
RP11−108P20.4
BOP1,MIR7112
ADORA3
EIF4A3
USO1
MRPL15
SKAP1
RPPH1
NT5DC3
GHRHR
RP11−21L23.2
CLCA2
COL3A1
MARC1
RP11−613M5.1
ALK
LINC00337
LINC00558
DTX1
CTD−2033C11.1
SSX2,SSX2B,SSX
CLSPN
REEP2
FOXN3−AS2
LPGAT1
CCDC136
LINC00865
AP1S3
DDIAS
RABGAP1L
C10orf88
ANXA2
DERA
RP11−324J3.1
CDIPT−AS1
CYP4Z1
GM2A
MYO9A
CCR3
TPRA1
SLC26A4
DEFB103A,DEFB1
TTL

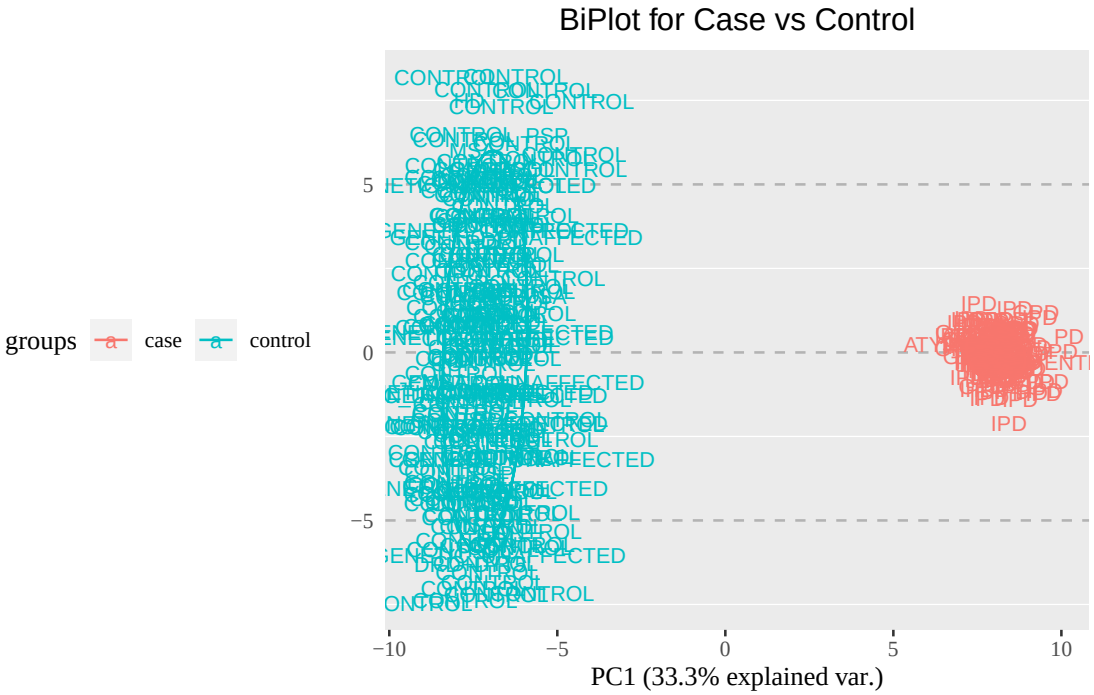
Notice the annotation bar along the top. This indicates PD vs not PD samples. This heat map is an example of a 'non-supervised method' - where we didn't feed the labelled data to the algorithm. Instead, it is just clustering similar samples together. Because all of our PD samples cluster away from the non-PD samples, we are relatively certian we've picked good biomarkers! We should also check a PCA plot.

```
pc<-prcomp(t(esetSel),center=T,scale=T)
plot(pc,type="l",main="Checking the number of Principle Components")
```



Again, I've provided code for you here. **Change it to something you like better!**

```
#install.packages("devtools")
library(devtools)
library(ggpubr)
#install_github("vqv/ggbiplot")
library(ggbiplot)
g = ggbiplot(pc, obs.scale = 1, var.scale = 1,
              groups = as.factor(caseName[k]), ellipse = T,
              circle = T, labels=disease_label[k],var.axes = F)
g = g + scale_color_discrete(type = getOption("ggplot2.discrete.fill"))
g = g + theme(legend.direction = 'horizontal', plot.title = element_text(hjust = .5, family="sans"),
        text = element_text(family = "serif"), legend.position = 'left')
g = g + theme_cleveland()
g = g + ggtitle("BiPlot for Case vs Control")
print(g)
```


BiPlot for Case vs Control
groups
case
control
5
0
−5
−10
−5
0
5
10
PC1 (33.3% explained var.)

We have separation! Notice the obvious differences between cases and controls.

Make a variable which only contains the differential gene names and call it diffGenes AND print out all of these gene names using one line of code.

```
#your code here
(diffGenes = selected)
```

```
##   [1] "EXOC3L4"
##   [2] "FAM132A"
##   [3] "MDM2"
##   [4] "CCR3"
##   [5] "MYO9A"
##   [6] "GADD45GIP1"
##   [7] "ANXA2"
##   [8] "CCNJ"
##   [9] "EMC6"
##  [10] "GEMIN4"
##  [11] "PPM1K"
##  [12] "TTL"
##  [13] "DEFB103A,DEFB103B"
##  [14] "STX3"
##  [15] "DERA"
##  [16] "HIST1H2BO"
##  [17] "RGS4"
##  [18] "MAN1B1-AS1"
##  [19] "PSG5"
##  [20] "OR4C1P"
##  [21] "ABHD12B,MIR4454"
##  [22] "COCH"
##  [23] "RWDD4"
##  [24] "FAR1"
##  [25] "PEX1"
##  [26] "THUMPD1"
##  [27] "CTB-31O20.9"
##  [28] "GCLC"
##  [29] "SEC16B"
##  [30] "CYP2E1"
##  [31] "EGLN1"
##  [32] "PRKAG2"
##  [33] "NPC2"
##  [34] "TPRA1"
##  [35] "SLC26A4"
##  [36] "XAB2"
##  [37] "C10orf88"
##  [38] "MGC45922"
##  [39] "P2RX2"
##  [40] "AARS"
##  [41] "RNF157"
##  [42] "PSMC2"
##  [43] "RBPMS-AS1"
##  [44] "PCDHGB8P"
##  [45] "CD207"
##  [46] "RLTPR"
```

```
##  [47] "TTTY15"
##  [48] "TRAF5"
##  [49] "RP11-245J9.5"
##  [50] "KCNA10"
##  [51] "UQCC2"
##  [52] "RP11-324J3.1"
##  [53] "GABRA1"
##  [54] "CPN2"
##  [55] "MAP6"
##  [56] "C17orf62"
##  [57] "TRAFD1"
##  [58] "HIPK4"
##  [59] "GM2A"
##  [60] "N6AMT1"
##  [61] "RABGAP1L"
##  [62] "ANP32A"
##  [63] "ROBO2"
##  [64] "TOPORS-AS1"
##  [65] "STEAP2"
##  [66] "RNF167"
##  [67] "HDAC2"
##  [68] "ETFB"
##  [69] "RP11-521D12.1"
##  [70] "PHKA1"
##  [71] "TNS4"
##  [72] "EIF4A2"
##  [73] "ZNF689"
##  [74] "BACH1,GRIK1-AS2"
##  [75] "LRIT1"
##  [76] "KBTBD8"
##  [77] "B3GAT2"
##  [78] "DNM2"
##  [79] "DDIAS"
##  [80] "C2CD3"
##  [81] "CNTN2"
##  [82] "AP1S3"
##  [83] "CDIPT-AS1"
##  [84] "HIGD1A"
##  [85] "KIAA0101"
##  [86] "PERM1"
##  [87] "IFNL1"
##  [88] "CYP4Z1"
##  [89] "R3HDM4"
##  [90] "HMGCLL1"
##  [91] "RBM41"
##  [92] "RP11-108P20.4"
##  [93] "ARL6"
##  [94] "LINC00623,LINC00869,LINC01138,LOC103091866"
##  [95] "LINC00865"
##  [96] "ASMTL-AS1"
##  [97] "CASP14"
##  [98] "OR5J2"
##  [99] "DDX60L"
## [100] "ZDHHC24"
```

```
## [101] "MUC20"
## [102] "SYNPO"
## [103] "LAIR2"
## [104] "UCP3"
## [105] "REEP2"
## [106] "HDAC10"
## [107] "CBLN1"
## [108] "AP2M1"
## [109] "FOXN3-AS2"
## [110] "SYP"
## [111] "PPP6R2"
## [112] "CDH26"
## [113] "RPPH1"
## [114] "NT5DC3"
## [115] "ZNF627"
## [116] "STUB1"
## [117] "DTX1"
## [118] "CCDC136"
## [119] "FAM169A"
## [120] "LINC00558"
## [121] "CLCA2"
## [122] "GINM1"
## [123] "GHRHR"
## [124] "PKD2L2"
## [125] "RP11-742B18.1"
## [126] "LPGAT1"
## [127] "EIF4A3"
## [128] "CTD-2033C11.1"
## [129] "LEF1"
## [130] "LMTK2"
## [131] "A1BG"
## [132] "LINC00343"
## [133] "FAM110D"
## [134] "ADORA3"
## [135] "DKC1,MIR664B,SNORA56"
## [136] "BOP1,MIR7112"
## [137] "SCIMP"
## [138] "MAB21L1,MIR548F5"
## [139] "ZNF883"
## [140] "ZC3H14"
## [141] "PADI4"
## [142] "CLSPN"
## [143] "ZNF24"
## [144] "PLIN3"
## [145] "AURKC"
## [146] "RP11-320N7.2"
## [147] "FAM99B"
## [148] "LPCAT4"
## [149] "MPV17L"
## [150] "CD22"
## [151] "NEK11"
## [152] "MARC1"
## [153] "NR3C1"
## [154] "USO1"
```

```
## [155] "GJD4"
## [156] "RP11-21L23.2"
## [157] "LINC01426"
## [158] "STAT1"
## [159] "IGLC1,IGLV3-10,IGLV3-10"
## [160] "MRPL15"
## [161] "INPPL1"
## [162] "C17orf51"
## [163] "DCAF12"
## [164] "LINC00337"
## [165] "CYFIP2"
## [166] "LINC00927"
## [167] "ALK"
## [168] "SSX2,SSX2B,SSX3"
## [169] "ROCK2"
## [170] "MAGEC3"
## [171] "PSKH1"
## [172] "SKAP1"
## [173] "COL3A1"
## [174] "MYLIP"
## [175] "RP11-613M5.1"
```

To use these genes as a classifier, we will need to define a score function. Our score will be the sum of the average expression for the upregulated (positive) genes and the average for the down regulated (negative) genes. Here, I've written you a function which will do this. Please enter it and **make comments to show you understand what its doing.**

```
PDscore<-function(x,g,v,s){
  #x expression values for a sample
  #g all the genes
  #v the diffGenes
  #s is the sign of the logFC

  i = which(g%in%v)
  x = x[i] #updating the expression values for a sample
  s = s[i] #updating the sign of logFC values in the sample
  p = c() #initializing vector p
  n = c() #initializing vector n

  for(i in 1:length(x)){  #checking sign of logFC scores that are > 0 in the sample
    if(s[i]>0){
      p<-append(p,(x[i])) #appending these values to vector p
    }
    else if(s[i]<0){
      n<-append(n,(x[i])) #appending the logFC scores with negative signs to vector n
    }
  }

  #replacing null values with 0 for vectors p and n
  if(is.null(p)){p[1]=0}
  if(is.null(n)){n[1]=0}

  score = mean(p)-mean(n) #taking the difference of the means of the positively and negatively signed l
  return(score) #returning that mean difference
```

```
}
```

Now we can apply our function to our expression set to define a score for each patient. **Comment what this is doing and why each step is necessary!**

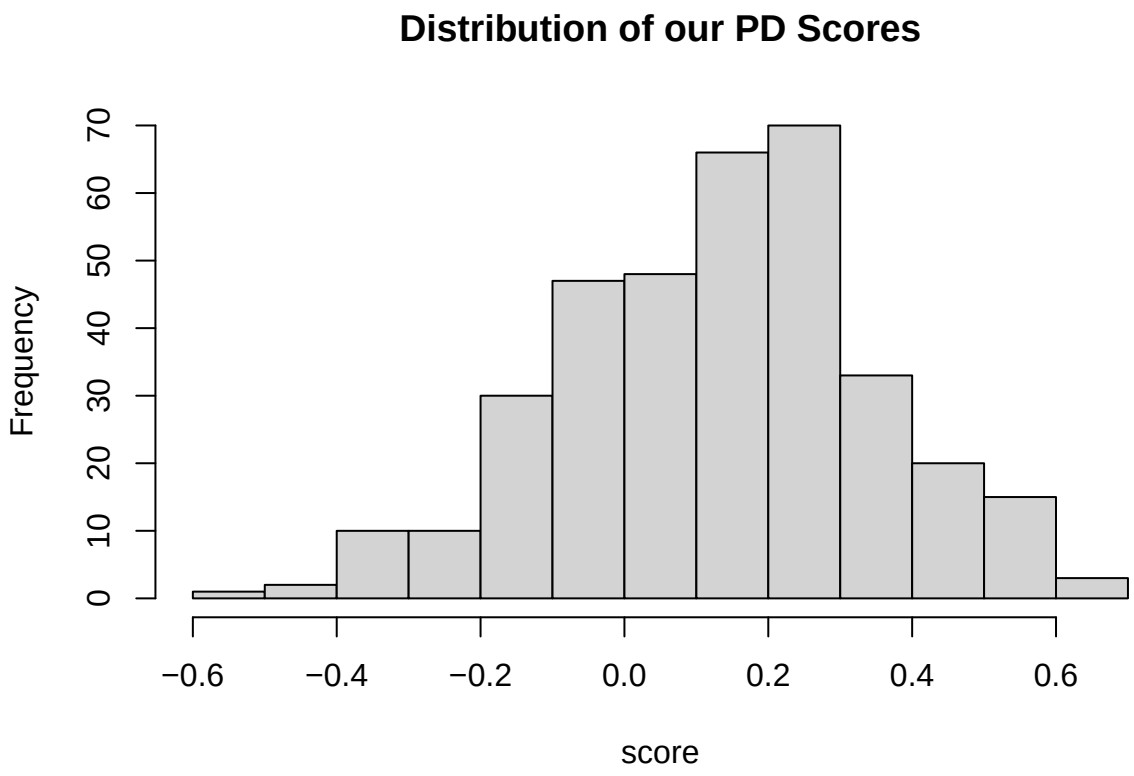
```
score<-c() #initializing vector score

#searching for the Gene Names in results
allGenes<-as.character(expr[as.character(expr$GeneName)%in%rownames(results),1])

#calculating PD Scores for each specific Gene Name

for(i in 1:ncol(eset)){ #updating scores for the entire subset eset
  score[i]<-PDscore(eset[,i],allGenes,diffGenes,direction)
}

hist(score,main="Distribution of our PD Scores") #histogram for our PD Scores
```

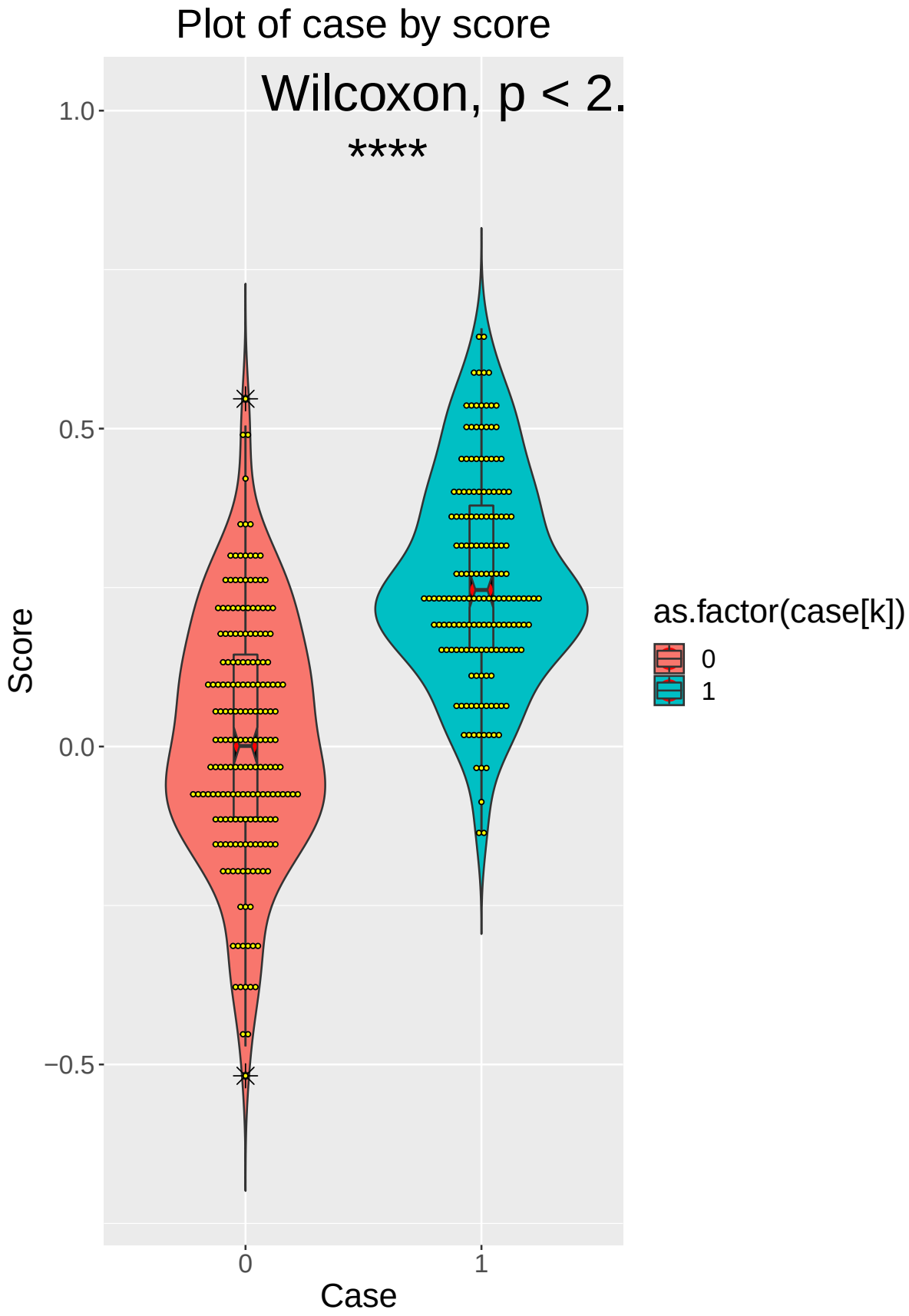


Now we'll use ggplot to make and interpret a violin plot of our score. I've provided some code to do this, but **try to change labels, colours, etc. to make it your own.**

```
#install.packages("ggpubr")
library(ggpubr)

df = data.frame(cbind(case[k],score))
```

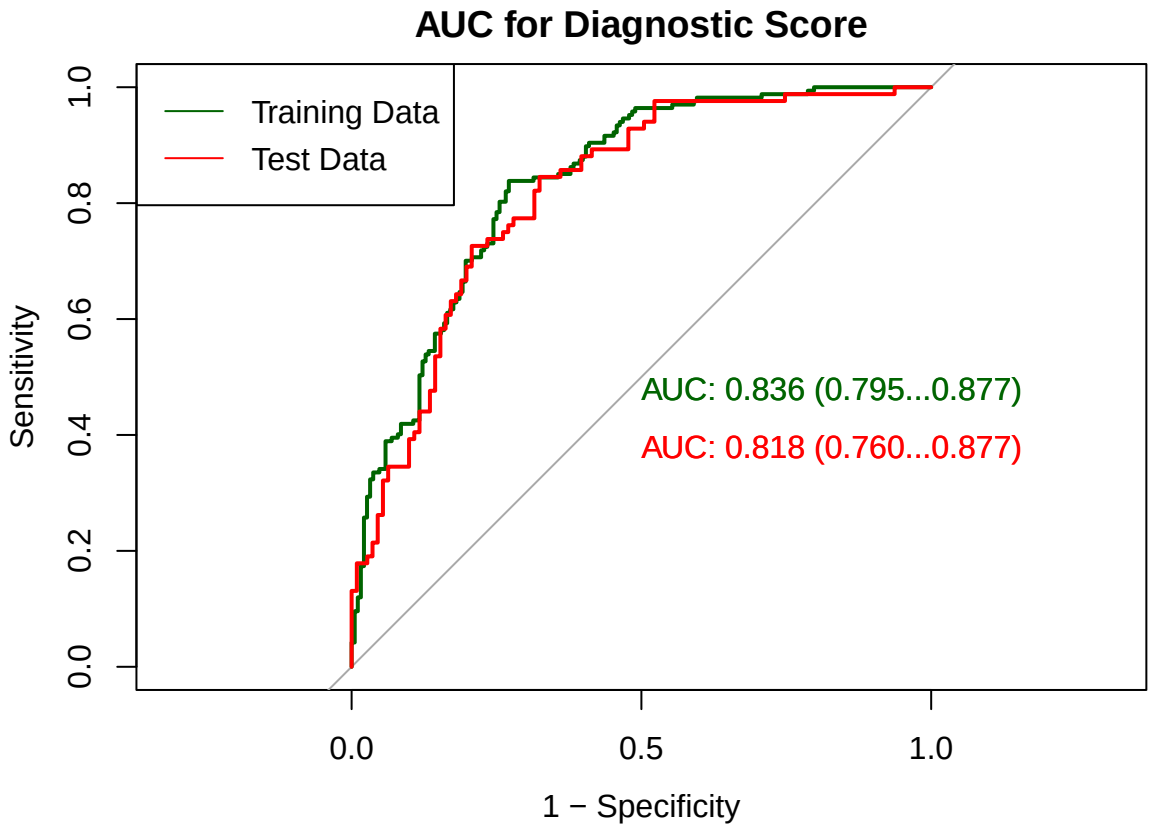
```
dp = ggplot(df, aes(x=as.factor(case[k]), y=score, fill=as.factor(case[k]))) +
  geom_violin(trim=F)+
  scale_color_brewer(palette="Dark2") +
  geom_boxplot(width=0.1, fill="black")+
  labs(title="Plot of case by score",x="Case ", y = "Score") +
  stat_compare_means(label.x = 1.5, label.y = 1, size=10)+
  stat_summary(fun.y=median, geom="point", size=5, color="red") +
  geom_boxplot(width=0.1, outlier.colour="black", outlier.shape=8, outlier.size=4, notch=T) +
  geom_dotplot(binaxis='y', stackdir='center', dotsize=0.2, fill="#ffff00") +
  scale_color_grey() +
  stat_compare_means(aes(label = ..p.signif..), label.x = 1.5, label.y = 0.9, size =10)
dp + theme(plot.title = element_text(hjust = .5, family="sans"), text = element_text(size = 18))
```


Plot of case by score
Wilcoxon, p < 2.
****
1.0
0.5
0.0
−0.5
Score
0
1
Case
as.factor(case[k])
0
1

This shows not only the boxplot of our data, but also the distribution of our data points around the boxplot! As before, we can see that we DO have significant separation for our score, and we can see that the cases are trending to have a higher score. With more time and data cleaning we may be able to find something here!

Let's make a roc plot, first with our training data, and then with our test data. As before, **play with the plot options to make something you like!**

```
#install.packages("verification")
#install.packages("pROC")
library("pROC")
testEset<-expr[,2:ncol(expr)]
testEset<-testEset[,-k]
newScore<-apply(testEset,2,FUN=PDscore,allGenes,diffGenes,direction)
plot.roc(case[k]~score, data=df,legacy.axes=T,print.auc=T, ci=T, col="darkgreen",main="AUC for Diagnosti
plot.roc(case[-k]~newScore,data=data.frame(cbind(case[-k],newScore)),add=T,print.auc=T, ci=T, col="red"
legend("topleft",c("Training Data","Test Data"),lty=c(1,1),col=c("darkgreen","red"))
```



Notice that our score does better with our training data - this is expected! This is why we need to split our data, to avoid problems with over-fitting. These scores are better than random (the grey line), but we'd like to see an AUC as close to 1 as possible. Let's see if we can do better!

Statistics!

We can run a t-test to see if our score is significantly different between cases and controls. Try using the t.test function in R.

```
# your code here
allScore=c(score, newScore)
mergeCase=c(case[k],case[-k])
t.test(allScore[mergeCase==0],allScore[mergeCase==1])
```

```
##
##  Welch Two Sample t-test
##
## data:  allScore[mergeCase == 0] and allScore[mergeCase == 1]
## t = -15.842, df = 548, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.2743704 -0.2138353
## sample estimates:
##   mean of x   mean of y
## 0.005519575 0.249622456
```

The mean scores for our cases and controls are close, but they are significantly different with an extremely small p-value of 2.787e-13. This highlights a classical statistical fallacy - while small p-values are great, they are often meaningless without a large enough effect size. Here, we have achieved significance due to the large sample size of our study, hence our study is adequately powered.

We could also run a simple regression to examine the impact of the score on the log odds of being a case.

```
# your code here
smallModel = glm(case[k]~score,family="binomial")
summary(smallModel)
```

```
##
## Call:
## glm(formula = case[k] ~ score, family = "binomial")
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5109  -0.7565  -0.2132   0.8773   2.1995
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.2373     0.1862  -6.646 3.02e-11 ***
## score         7.9472     0.8874   8.955  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 490.89  on 354  degrees of freedom
## Residual deviance: 350.94  on 353  degrees of freedom
## AIC: 354.94
##
## Number of Fisher Scoring iterations: 5
```

Summarize this output!

Again, we conclude that the score is a statistically significant indicator of the odds of having PD. Let's build a larger model which examines other phenotype variables.

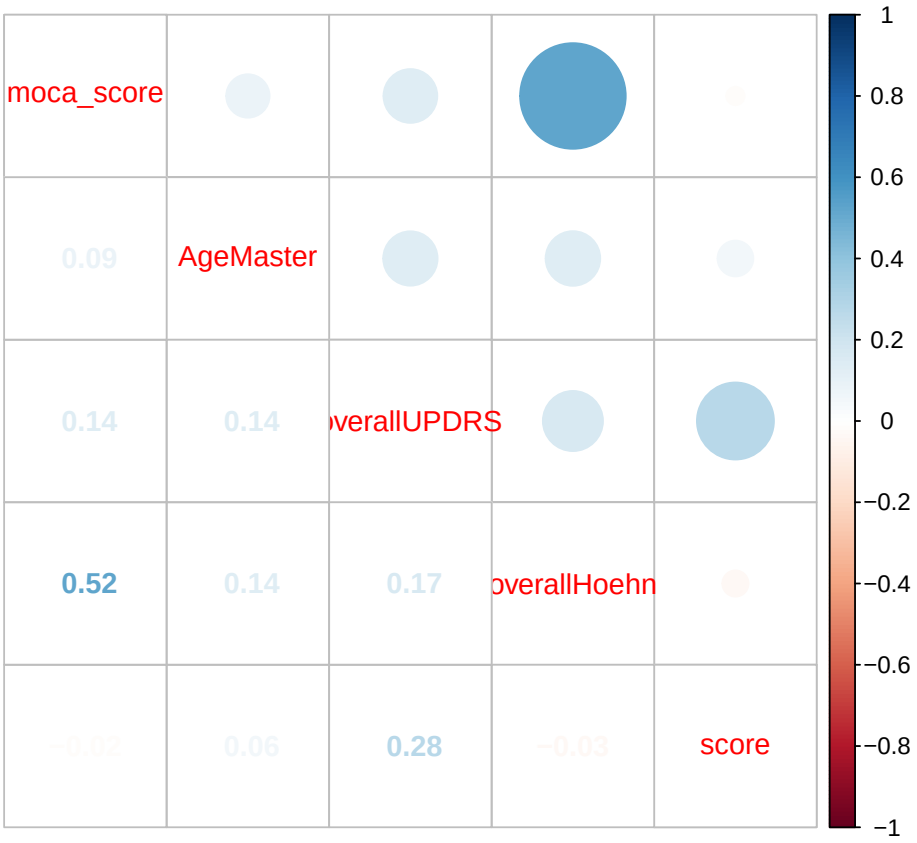
First, build a data frame which includes all the model data we're interested in. Start with the age variables in your pheno set, and then use the cbind() function to add on our scores and the binary case vector. Print a summary of the model data.

```
# your code here
View(pheno)
modelData = cbind(pheno[k,3:7],score)
summary(modelData)
```

```
##      sex                moca_score      AgeMaster      overallUPDRS
##  Length:355         Min.   : 0.00   Min.   :26.00   Min.   : 0.000
##  Class :character   1st Qu.: 0.00   1st Qu.:53.50   1st Qu.: 0.000
##  Mode  :character   Median :25.00   Median :62.00   Median : 0.000
##                     Mean   :17.01   Mean   :60.15   Mean   : 1.489
##                     3rd Qu.:29.00   3rd Qu.:67.00   3rd Qu.: 0.600
##                     Max.   :30.00   Max.   :82.00   Max.   :15.800
##                     NA's   :12      NA's   :132     NA's   :167
##   overallHoehn        score
##  Min.   :0.000   Min.   :-0.51781
##  1st Qu.:0.000   1st Qu.:-0.02063
##  Median :0.000   Median : 0.14507
##  Mean   :1.412   Mean   : 0.12938
##  3rd Qu.:4.000   3rd Qu.: 0.26144
##  Max.   :4.500   Max.   : 0.65770
##  NA's   :161
```

We should examine the correlations in our data set. You can do this quickly by building a correlation plot matrix.

```
#install.packages("corrplot")
library(corrplot)
M<-cor(modelData[,-1],use="pairwise.complete.obs") #for  missing data
corrplot.mixed(M)
```


moca_score
0.09
AgeMaster
0.14
0.14
overallUPDRS
0.52
0.14
0.17
overallHoehn
−0.02
0.06
0.28
−0.03
score
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1

How would you interpret this output? **Answer below!** We can see that there exists a strong positive correlation between the moca_score and overallHoehn. Moreover, there is also a positive correlation between overallUPDRS and score. All the other features are either mildly positively correlated or not correlated. There does not exist a negative correlation between any of the features. For interpretation: blue indicates positive correlation, red indicates negative correlation and white indicates no correlation. Moreover, the size of the circle indicates the positive value of the correlation.

Let's build our first model. Here, we consider the case as our dependent variable, and the others as our explanatory variables.

```
model1<-glm(case[k]~.,family=binomial,data=modelData)
summary(model1)
```

```
##
## Call:
## glm(formula = case[k] ~ ., family = binomial, data = modelData)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7563  -0.2907   0.1996   0.5378   1.8799
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -5.18451    2.65437  -1.953   0.0508 .
## sex Male      0.46642    0.86007   0.542   0.5876
## moca_score    0.11231    0.04814   2.333   0.0196 *
## AgeMaster     0.04904    0.04053   1.210   0.2263
## overallUPDRS  0.23521    0.12351   1.904   0.0569 .
## overallHoehn  0.18013    0.23660   0.761   0.4465
## score         6.01788    2.70221   2.227   0.0259 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 68.752  on 55  degrees of freedom
## Residual deviance: 39.610  on 49  degrees of freedom
##   (299 observations deleted due to missingness)
## AIC: 53.61
##
## Number of Fisher Scoring iterations: 6
```

We will iteratively remove variables with the highest p-values, and then rerun the model until we have our optimal fit!.

Try this on your own first.

```
# your code here
modelData$case = case[k]
modelData2 = na.omit(modelData)
model2 = glm(case~sex+AgeMaster+moca_score+overallUPDRS+overallHoehn+score,family=binomial,data=modelDa
summary(model2)
```

```
##
```

```
## Call:
## glm(formula = case ~ sex + AgeMaster + moca_score + overallUPDRS +
##     overallHoehn + score, family = binomial, data = modelData2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7563  -0.2907   0.1996   0.5378   1.8799
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -5.18451    2.65437  -1.953   0.0508 .
## sex Male      0.46642    0.86007   0.542   0.5876
## AgeMaster     0.04904    0.04053   1.210   0.2263
## moca_score    0.11231    0.04814   2.333   0.0196 *
## overallUPDRS  0.23521    0.12351   1.904   0.0569 .
## overallHoehn  0.18013    0.23660   0.761   0.4465
## score         6.01788    2.70221   2.227   0.0259 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 68.752  on 55  degrees of freedom
## Residual deviance: 39.610  on 49  degrees of freedom
## AIC: 53.61
##
## Number of Fisher Scoring iterations: 6
```

```
modelData2$sex = NULL
modelData2$AgeMaster = NULL

model3 = glm(case~.,family=binomial,data=modelData2)
summary(model3)
```

```
##
## Call:
## glm(formula = case ~ ., family = binomial, data = modelData2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5234  -0.4647   0.2259   0.5873   2.1496
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -2.05966    0.92085  -2.237   0.0253 *
## moca_score    0.09106    0.04207   2.164   0.0304 *
## overallUPDRS  0.26473    0.11272   2.348   0.0188 *
## overallHoehn  0.14260    0.21142   0.674   0.5000
## score         5.85009    2.52620   2.316   0.0206 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 68.752  on 55  degrees of freedom
## Residual deviance: 41.977  on 51  degrees of freedom
## AIC: 51.977
##
## Number of Fisher Scoring iterations: 6
```

This is our final model! Notice that our largest effect size is controlled by our genetic score. At a first glance, we might assume this means that the score has the largest effect on the model. However, if we recall how to interpret our coefficients, the estimated effect size is the change in log odds of being a case for a 1 unit increase in our score. Think about the score distribution: the range of our scores is fairly small. In contrast, the range of the updrs scores varies from 0 to 36. Keep in mind the scale of our data when interpreting these models!

Compare this to your outcome if you use a step function to reduce the model:

```
#your code here
stepModel<-step(model3)
```

```
## Start:  AIC=51.98
## case ~ moca_score + overallUPDRS + overallHoehn + score
##
##                Df Deviance    AIC
## - overallHoehn  1   42.443 50.443
## <none>              41.977 51.977
## - overallUPDRS  1   48.270 56.270
## - moca_score    1   48.569 56.569
## - score         1   48.898 56.898
##
## Step:  AIC=50.44
## case ~ moca_score + overallUPDRS + score
##
##                Df Deviance    AIC
## <none>              42.443 50.443
## - moca_score    1   48.607 54.607
## - score         1   49.481 55.481
## - overallUPDRS  1   49.506 55.506
```

```
summary(stepModel)
```

```
##
## Call:
## glm(formula = case ~ moca_score + overallUPDRS + score, family = binomial,
##     data = modelData2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4180  -0.5387   0.2296   0.5882   2.2804
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -1.7466     0.7470  -2.338   0.0194 *
## moca_score     0.0882     0.0423   2.085   0.0371 *
## overallUPDRS   0.2741     0.1113   2.462   0.0138 *
```

```
## score          5.7640     2.4648   2.339   0.0194 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 68.752  on 55  degrees of freedom
## Residual deviance: 42.443  on 52  degrees of freedom
## AIC: 50.443
##
## Number of Fisher Scoring iterations: 6
```

Notice that our step-wise reduced model chose to keep both age and overallHoehn despite the insignificant p-value. Why? Answer below!

My step-wise model has successfully removed cols with insignificant p-values, including AgeMaster and overallHoehn.

Let's predict the probability of having a case given our manually reduced model. Make a histogram of the score from this model.

```
# your code here
modelScore=predict(stepModel,newdata = modelData)
hist(modelScore, main = "Histogram of Logistic Regression Model")
```

Like before, we'll build a violin plot to compare the output of our regression model. See if you can adapt the violin plot code from before to do this now.

```r
# your code here
library(ggpubr)

df<-data.frame(cbind(case[k],modelScore))

dp <- ggplot(df, aes(x=as.factor(case[k]), y=modelScore, fill=as.factor(case[k]))) +
  geom_violin(trim=FALSE)+
  geom_boxplot(width=0.1, fill="white")+
  scale_color_brewer(palette="Dark2") +
  labs(title="Plot of case by score",x="Case ", y = "Score")+
  stat_compare_means(label.x = 1.5, label.y = 1, size=10)+
  geom_dotplot(binaxis='y', stackdir='center', dotsize=0.1, fill="#ffff00") +
  stat_compare_means(aes(label = ..p.signif..), label.x = 1.5, label.y = 0.9, size =10)
dp + theme(plot.title = element_text(hjust = .5, family="sans"), text = element_text(size = 18))
```

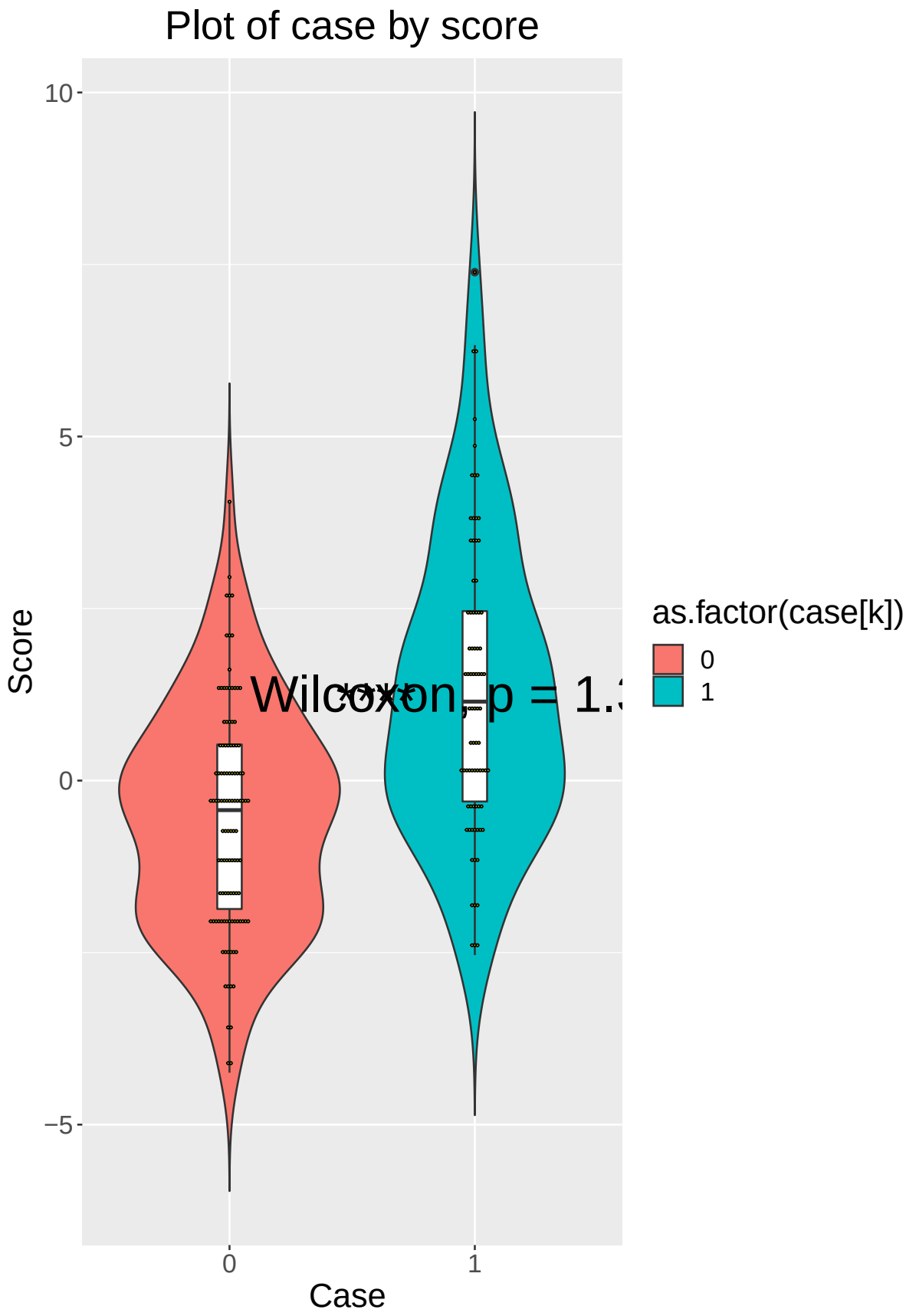
```
## Warning: Removed 167 rows containing non-finite values (stat_ydensity).

## Warning: Removed 167 rows containing non-finite values (stat_boxplot).

## Warning: Removed 167 rows containing non-finite values (stat_compare_means).

## Bin width defaults to 1/30 of the range of the data. Pick better value with `binwidth`.

## Warning: Removed 167 rows containing non-finite values (stat_bindot).
## Removed 167 rows containing non-finite values (stat_compare_means).
```


Plot of case by score
10
5
0
−5
Score
Wilcoxon, p = 1.3
****
0
1
Case
as.factor(case[k])
0
1

Now we're starting to see a clearer separation of scores! It's clear that by including the established tests to pre-screen patients for PD and other neurological diseases we have improved overall performance. While this may be an obvious conclusion, it is worth noting that the context with which our diagnostic signature would be used would be on patients already exhibiting potential PD symptoms. Clearly this needs a little more work, but for a first pass at assessing raw data, it's not bad!

Again, we can examine ROC curves. I've done some of the set up to get the data in the right format. Use the ROC code above to then build your own plot!

```
library("pROC")
nd = cbind(pheno[-k,],newScore)
colnames(nd) = c(colnames(nd[1:ncol(nd)-1]),"score")
newMScore = predict(stepModel,newdata=nd)

plot.roc(case[k]~score, data=df,legacy.axes=F,print.auc=T, ci=T, main="AUC for Diagnostic Score", col="
```

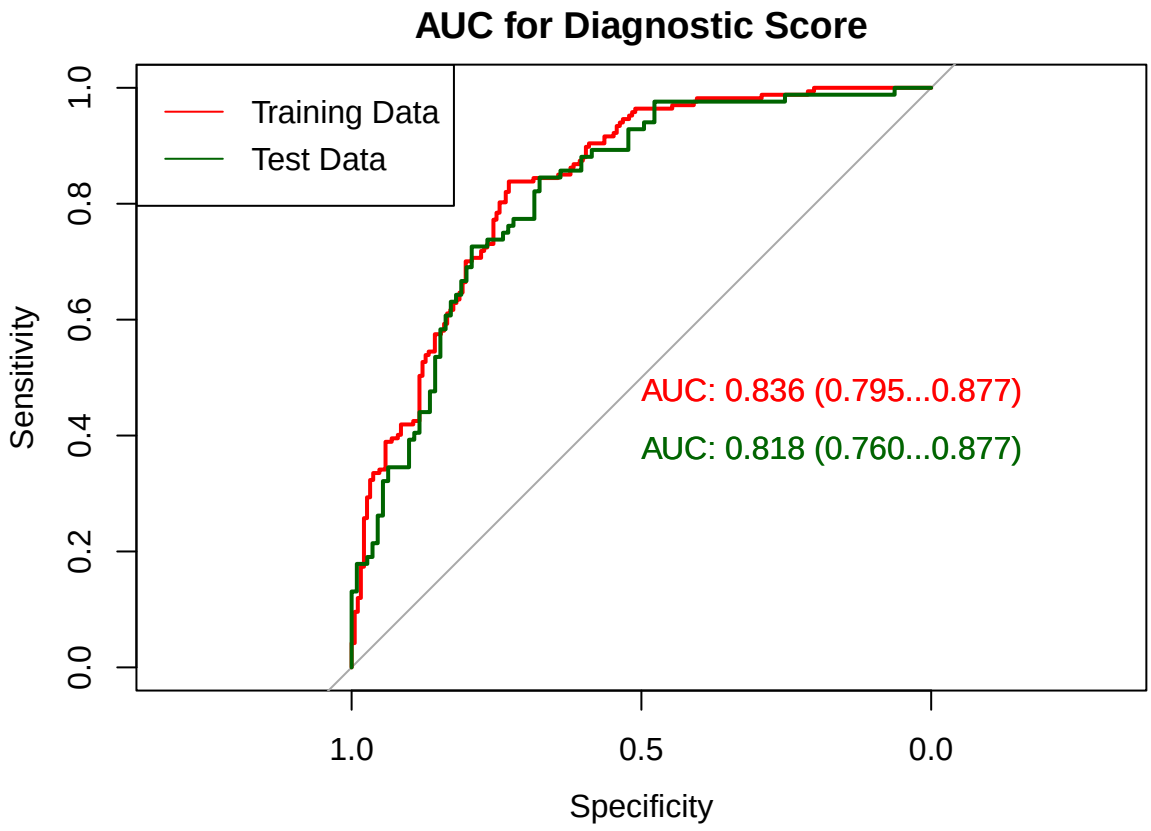
```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```

```
plot.roc(case[-k]~newScore,data=data.frame(cbind(case[-k],newScore)),add=T,print.auc=T, ci=T, col="dark
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
legend("topleft",c("Training Data","Test Data"),lty=c(1,1),col=c("red","darkgreen"))
```



```
#your code for ROC plots here
```

Here, we have a notable increase in AUC, particularly for our training data. Our test data shows an overal improvement as well, although with a large confidence interval. There are clearly some data points in here which are abnormal - and perhaps worth investigating.

##Congratulations, you have finished the R Bootcamp Assignment!