Yousra Belgaid, Prisha Bhamre

Final Exam: Stat 467 Spring 2023

Instructions: For each piece or group of software listed describe what it does for you, apply it to the data set I suggest if I do, and provide the results of the analysis and the conclusion.

1. Checking conditions of data

a. Suite 1: gui.mvntest, gui.orth, gui.explore (Class 3)

i. What does each do?

Gui.mvntest: a function that tests if a multivariate distribution is normal. The function takes two arguments, m1, which is a matrix representing the data to be tested, and ntest, which specifies the number of tests to be performed. Visual exploration of the dataset that helps determine whether the data deviates from normality. Deviations from normality may indicate non-linear relationships between the variables.

1) When looking at non-normality data, this function will provide non-interesting rejected Q-Q plots as well as 'interesting vectors' or ones that suggest non-normality or deviation from normality.
2) In other words, the smaller the p-value, the stronger the evidence against the null hypothesis. Therefore, a p-value less than the alpha value of 0.05 (or the alpha value specified) indicates that the observed effect or difference is very unlikely to be due to chance alone, and it is more likely that there is a real, interesting effect or difference present in the data.
3) The function uses the fdr (*false discovery rate*) function to identify the interesting (significant) directions in the data set- **the directions in which the data significantly deviate from normality**.

Gui.orth: takes the input (set of interesting directions) and returns the orthogonalized vectors.

gui.explore: explores data by projecting it onto an orthonormal matrix of interesting directions and visualizing it using different types of plots, such as pairs plots, animations, and 3D scatterplots. This approach can be helpful for exploring and understanding relationships among variables in the dataset from a different perspective, revealing patterns or trends in the dataset.

What characterization of the multivariate normal is gui.mvntest based on?
the **gui.mvntest** is based on one characterization of the multivariate normal: that every linear combination is normal. A normal distribution fit test is performed to ensure that the direction obtained is normally distributed.

b. Suite 2: gui.asym (Class 3)

i. What does it do?

The **gui.asym** function allows us to examine asymmetry of data, specifically for detecting potential outliers or "contamination" in the data. It calculates the trimmed mean and standard deviation of the data, and then plots the data points along one direction, with the trimmed mean indicated as a vertical line on the plot. In other words, calculates how far each point is from the center of the data and is used to identify the potential outliers in the dataset.

c. **Suite 3: Perm.cov.test (Class 10)**
    i. **What does it do:**

Perm.cov.test: assess the statistical significance of the differences in covariance structure between two groups of data Inputs include dat0: the data set, idcol: column index that correspond to the grouping in the dataset, numvec: a vector of column indices that correspond to the numeric variables in the dataset that will be used to calculate the linear discriminant function. The output will have a different color for each group.

d. For each data set in the rest of the test apply suite 1 and 2 to check normality of each subgroup of data. With mvn test, use at least 15 directions.
e. Where multiple subgroups exist use Perm.cov.test to test for different covariances. Then apply the specific procedures in the problem to the data set

2. One sample inference:

a. **Suite 4: gui.bootstrap.simconf, gui.bootstrap.1sample (Class 2)**
    i. What does each do

gui.bootstrap.simconf: This function performs a bootstrap simulation to obtain confidence intervals for the population mean vector of a multivariate dataset, based on the Hotelling's T-squared statistic. It performs the bootstrap simulation by resampling the data with replacement nboot times, calculating the Hotelling's T-squared statistic and subtracting the original mean vector, and storing the result as a difference vector (mmumat). Z1 is the Hotelling's T-squared statistic for each difference vector in mmumat using the inverse covariance matrix. The confidence intervals for the population mean vector of each variable are found by subtracting the maximum and minimum difference (within the quantiles of z1) from the original mean vector.

Gui.bootstrap.1sample: performs a bootstrap analysis for a one-sample Hotelling's T-squared test. bootstrap samples nboot times with replacement from the original data, calculates the Hotelling's T-squared statistic for each sample, and stores stat each in Tvec.

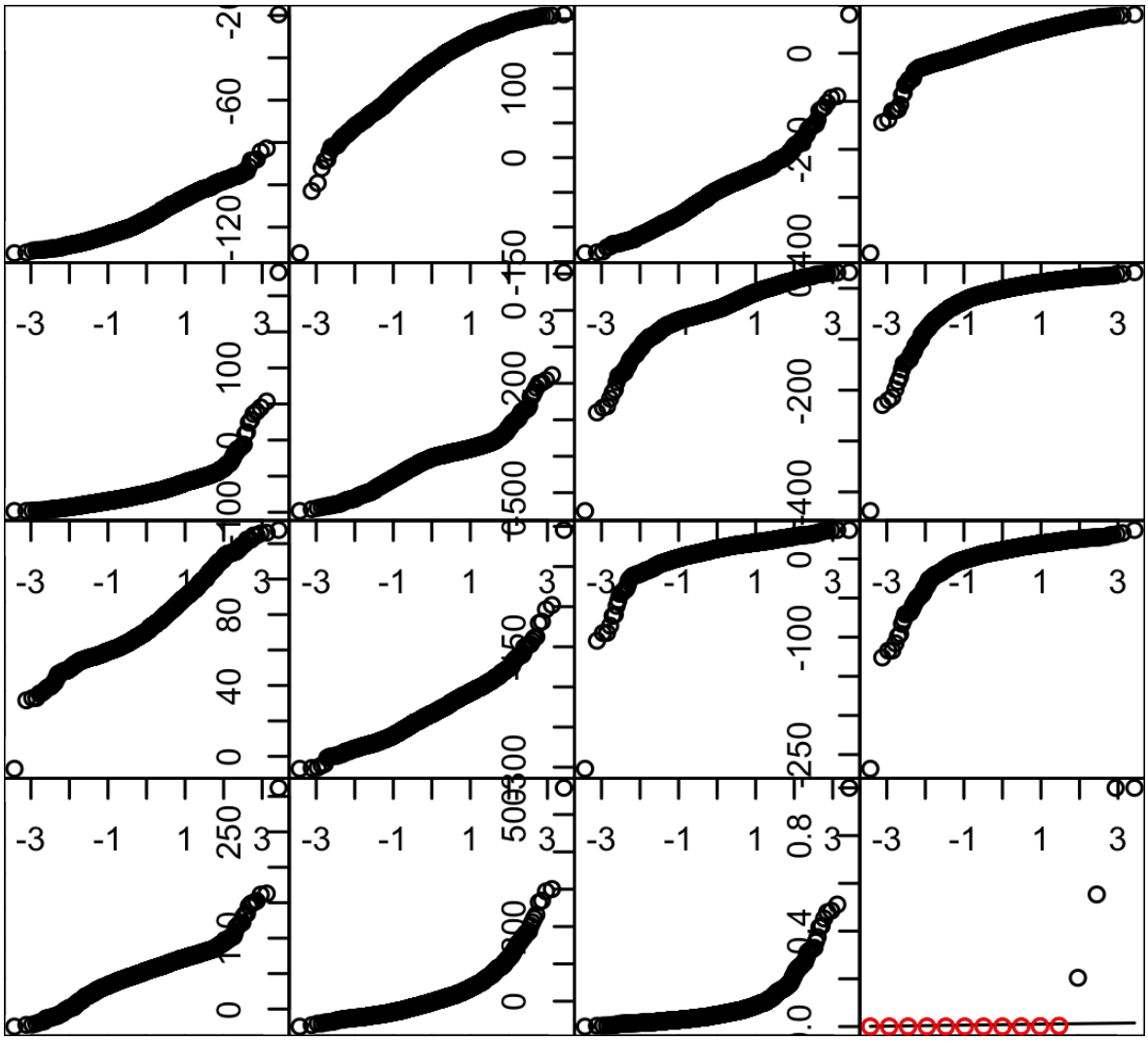
APPLICATION:

Apply suite 1 and 2 To the Pulsar measurements (col9=1) in the pulsar data set

```
> pulsar = pulsar[2:10]
> pulsar1 = subset(pulsar, pulsar[, 9] == 1)
> pulsar1 = pulsar1[1:8]
> pulsar1 = data.matrix(pulsar1)
```

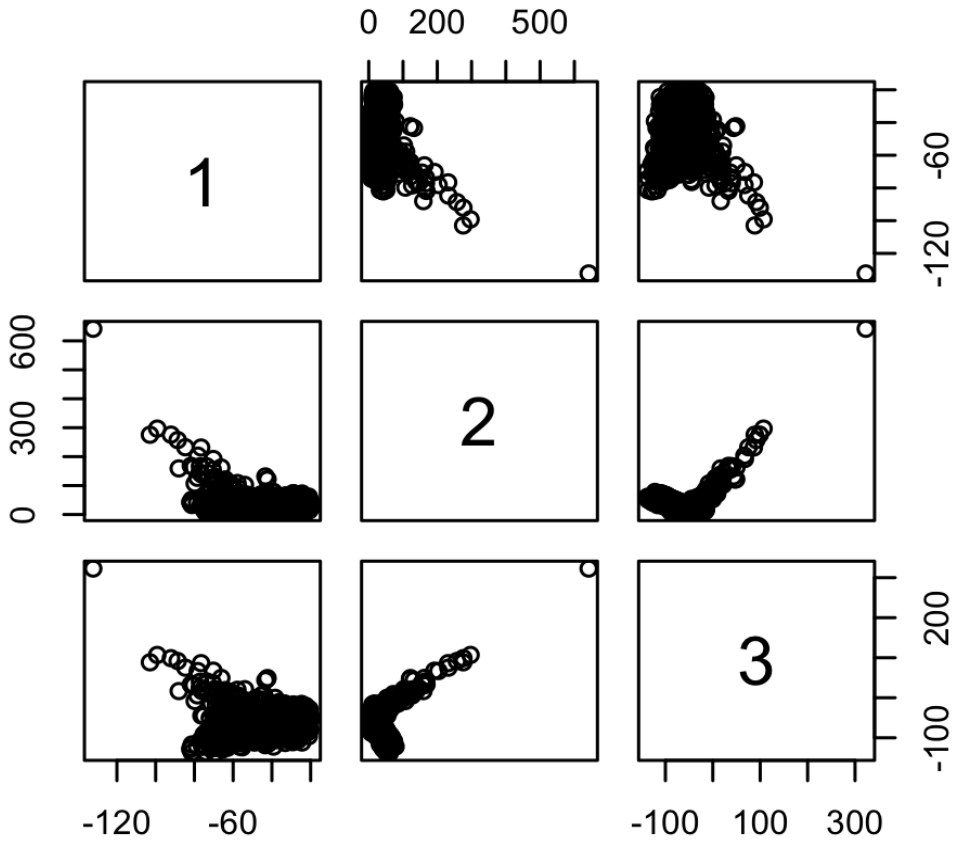
Suite1:

```
> mvntest<-Multivariate.normal.test(pulsar1, ntest = 15, Q = 0.05)
```

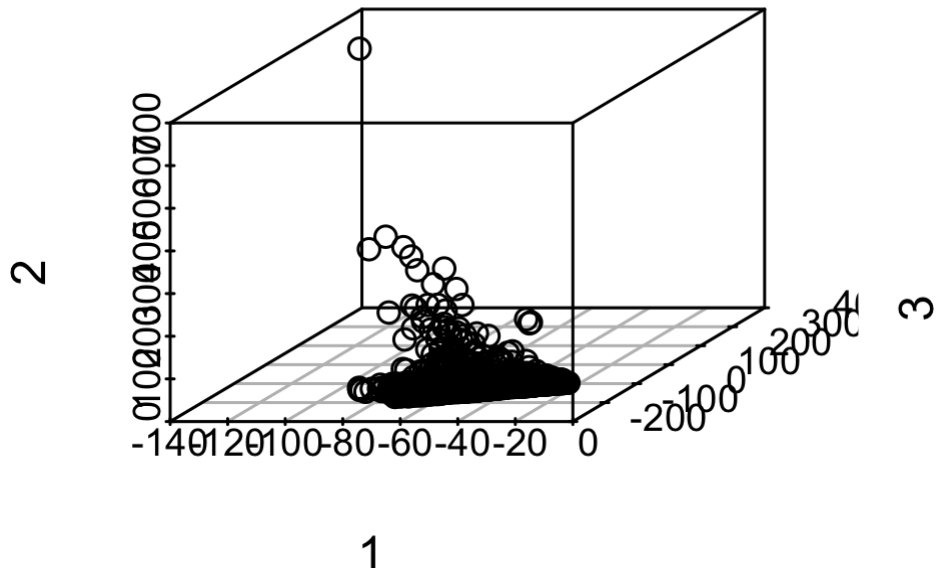


```
$Interesting.directions
         [,1]        [,2]        [,3]       [,4]
u1 -0.42901969 -0.01188772 -0.829953764  0.1390709
u1 -0.29359919 -0.05048929 -0.502842303  0.2009018
u1 -0.30362022  0.11382894  0.588836832 -0.4648573
u1  0.10102681  0.37149358 -0.005733853 -0.4086507
u1  0.02593634  0.06421690  0.455220700  0.4537073
u1 -0.10842022  0.10325536  0.265185452 -0.4388719
u1 -0.26362731 -0.42276426 -0.120576830  0.7105890
u1  0.01075925 -0.38823659 -0.605840617 -0.1456447
u1  0.45500572  0.57785120 -0.336252125 -0.2272923
u1  0.08868625 -0.67142292  0.213761718 -0.0506926
u1  0.28301852  0.43616791  0.322083192 -0.1020788
         [,5]        [,6]       [,7]       [,8]
```

```
u1 -0.04573772 -0.26194120 -0.1668086 -0.09542854
u1  0.25124713 -0.18111586 -0.5850274  0.42409114
u1 -0.47570422  0.14212458  0.1729249  0.23588066
u1  0.09113666  0.04722494  0.8174817  0.07710626
u1 -0.02453998 -0.64707314  0.2606849 -0.30800132
u1 -0.17009242  0.49121042  0.6037474 -0.28270383
u1 -0.28325941 -0.17763962  0.2442027  0.24671611
u1 -0.10011421 -0.25814168 -0.1869927  0.59099502
u1 -0.23156967 -0.02436757  0.1451203 -0.46803022
u1 -0.43892592 -0.22202341  0.4501153  0.22024545
u1  0.60569927  0.24622224 -0.4180669 -0.11499346
> orthmat<-orthogonalize(mvntest$Int[1:3,])
          [,1]       [,2]       [,3]      [,4]
[1,] -0.4290197 -0.01188772 -0.8299538 0.1390709
          [,5]       [,6]       [,7]       [,8]
[1,] -0.04573772 -0.2619412 -0.1668086 -0.09542854
> expl<-explore(orthmat,pulsar1,T,F,F)
```

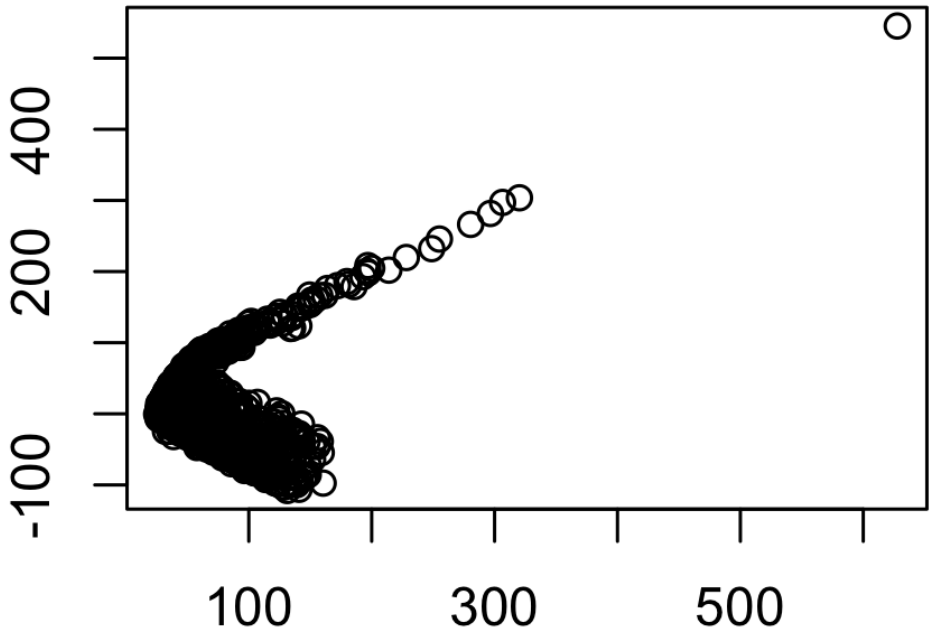



```
> expl<-explore(orthmat,pulsar1,F,T,F,c(1,3,2))
```



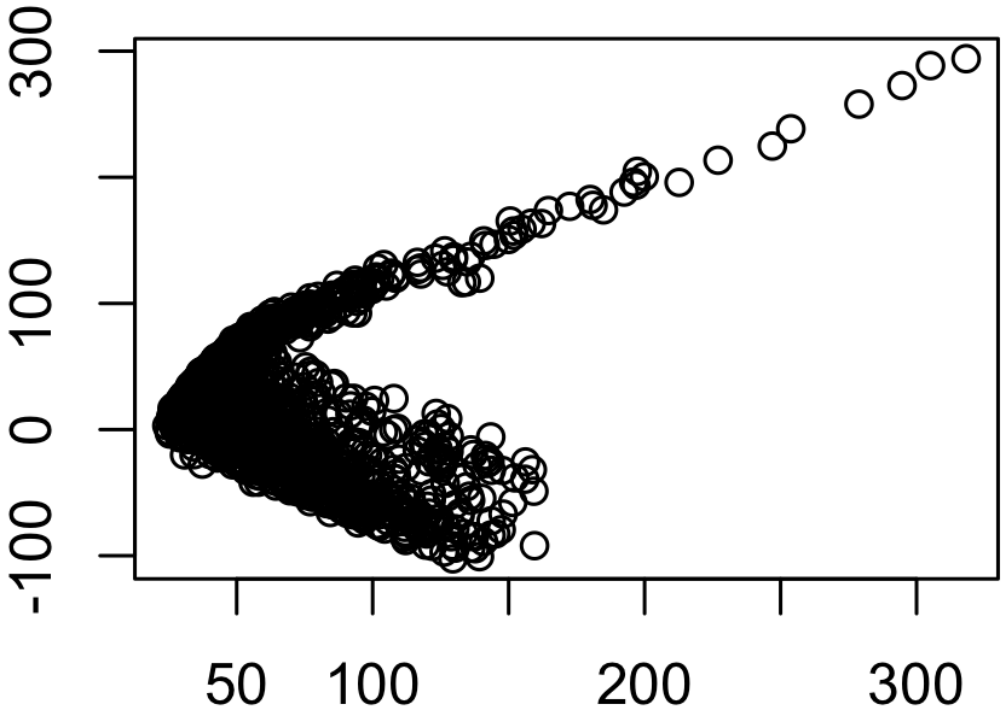
Suite 2:

> asym.contamination.plot(pulsar1,0,c(.2,.7),0.95,T,F,F)



[1] "Would you like to identify the points to remove, then reanalyze? (Y/N)"

1: Y
[1] "How many points?"
1: 1
[1] 1481
[1] 0



Calculate simultaneous confidence intervals of all 8 variables in the pulsar data set.

> gui.bootstrapsimconf()
[1] "pulsar1" "diag(8)" "10000" "0.05"

**** for brevity omitted 100, 200,... 10000 listing the times the bootstrap loop had run so far *****

$bootconf
[,1] [,2] [,3]
[1,] 53.971743 56.690608 59.202953
[2,] 38.056394 38.710598 39.413514
[3,] 2.970686 3.130655 3.288713
[4,] 14.361628 15.553576 16.622969
[5,] 45.868908 49.825995 53.562006
[6,] 54.772726 56.468963 58.049164
[7,] 2.498349 2.757069 3.012412
[8,] 13.537023 17.931728 21.962453

$Hotelling

```
         [,1]      [,2]      [,3]
[1,] 53.761238 56.690608 59.619978
[2,] 37.926352 38.710598 39.494844
[3,]  2.947826  3.130655  3.313485
[4,] 14.187161 15.553576 16.919991
[5,] 45.404960 49.825995 54.247030
[6,] 54.542803 56.468963 58.395123
[7,]  2.453864  2.757069  3.060273
[8,] 12.963205 17.931728 22.900252
```

************EXPLANATION OF INPUT AND RESULTS*********

The input includes the data matrix pulsar, the contrast matrix is simply diag(8) since there are 8 variables, and nboot=10000 bootstrap iterations or resampling with replacement, and the alpha value of 0.05. The output includes the bootstrapped Hotelling confidence intervals and the Hotelling CI. with three columns representing the lower bound, point estimate, and upper bound of the confidence intervals for the population mean vector of each variable. The bootstrap CI tends to be more conservative (smaller CI) than the normal Hotelling CI.

3. Two sample inference

a. Suite 5: gui.twosample.perm, gui.bootstrap.twosample.simconf

i. What does each do

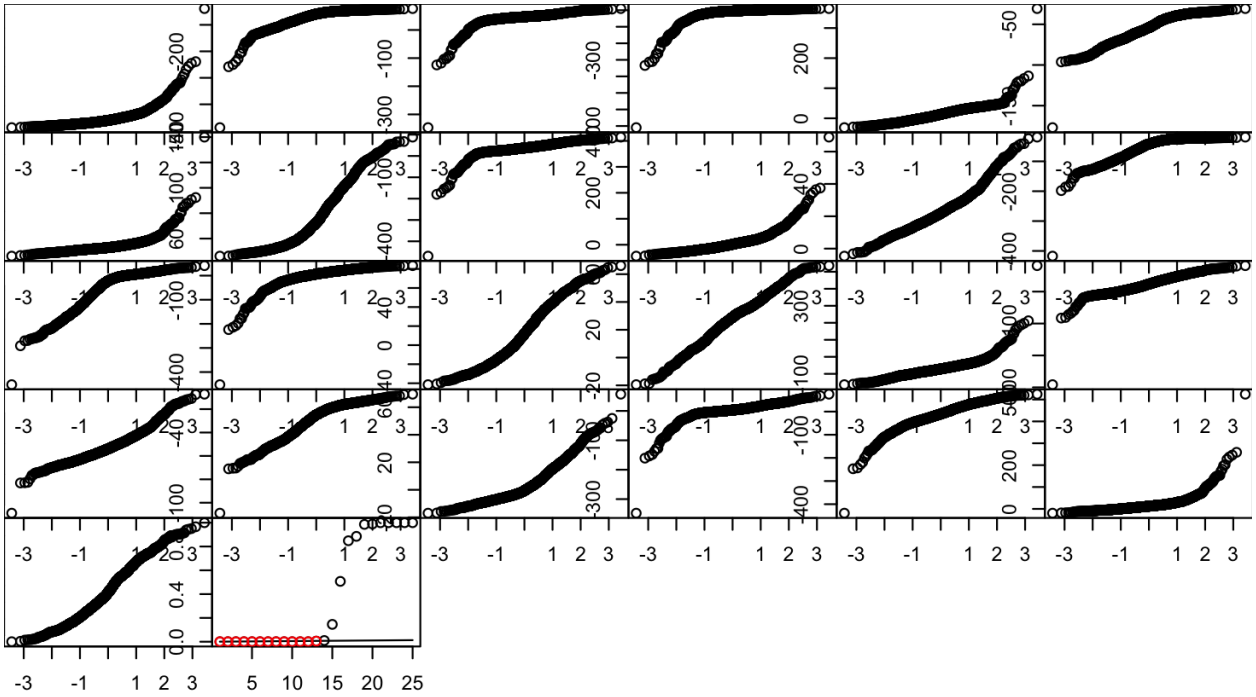
Gui.twosample.perm: perform the two-sample permutation test using Hotelling T^2 statistic.

gui.bootstrap.twosample.simconf: generates bootstrap confidence intervals for the difference between the means of two independent groups using Hotelling's T^2 test. The function first calculates a Hotelling's T^2 confidence interval for the difference between the means of the two groups. Then, it generates nboot bootstrap samples, calculates the difference in means for each bootstrap sample, and computes the confidence intervals for the differences. The function returns a list with two components: bootconf, a matrix with the confidence intervals for the differences, and Hotelling, the Hotelling's T^2 confidence interval.

APPLICATION:

Apply Suites 1 2, and 3 to groups in the pulsar data set

Suite 1 when PulsarData$v9 == ‘1’



Code: Suite 1 when PulsarData$v9 == ‘1’

```
>pulsar <- read.csv("~/Desktop/Stat 467/pulsar.csv", header=TRUE)
>pulsarOnly <- pulsar[pulsar$v9 == '1',]
>pulsar1<-pulsarOnly[,2:9]
```

#Subsetted Data into Matricies

```
>pulsarMat<-as.matrix(pulsar1)
```

#Multivariate Normal Test where Pulsar$v9 == '1'

```
>mvntest<-Multivariate.normal.test(pulsarMat, ntest = 25, Q = 0.05)
```

#Interesting Directions Found

```
 [,1]       [,2]       [,3]       [,4]       [,5]       [,6]       [,7]
u1  0.55297322  0.10003582  0.21025440 -0.158098937  0.31253780 -0.43511409
-0.12548086
u1 -0.28428096 -0.26549815 -0.12620648 -0.553204582 -0.56963342  0.05760581
-0.03328879
u1  0.06466824 -0.10843156 -0.60738423  0.123886831  0.07881779 -0.07202796
-0.34732657
u1 -0.03322593 -0.31331967  0.14880361 -0.814869576 -0.06925669 -0.40059909
0.16013446
u1  0.20124601  0.43172588  0.11716701  0.619216260  0.37292815  0.33634949  0.34997394
u1  0.27087052  0.70948649 -0.23215539 -0.232249537 -0.01229791  0.13051220
0.17063497
```

u1 0.43668331 -0.43814297 0.16882707 -0.105200867 0.19479969 0.13927311
-0.72025969
u1 0.04670099 -0.05924366 0.50846918 0.001702308 -0.67043707 -0.24432912
-0.25620115
u1 -0.03787990 -0.27523961 -0.07100622 -0.306973901 0.16344019 0.33770940
-0.71906385
u1 0.19034577 -0.42747556 0.39623531 0.101516756 -0.22353445 -0.19813755
-0.72039849
u1 -0.06484559 0.52690138 0.27392598 -0.147650557 0.19958415 -0.26538250
-0.62654663
u1 0.06600420 0.36890186 -0.01474850 -0.407349972 0.22624191 -0.22724328
-0.57172933
u1 0.21442289 0.17241021 0.41622616 0.353403982 -0.58198117 0.07442345
-0.50795720
         [,8]
u1 0.55880378
u1 -0.44477284
u1 -0.68393058
u1 -0.15379442
u1 0.03560596
u1 -0.51876888
u1 0.04062854
u1 -0.40117800
u1 -0.40708752
u1 -0.07432986
u1 -0.34425495
u1 0.51352118
u1 -0.15458820

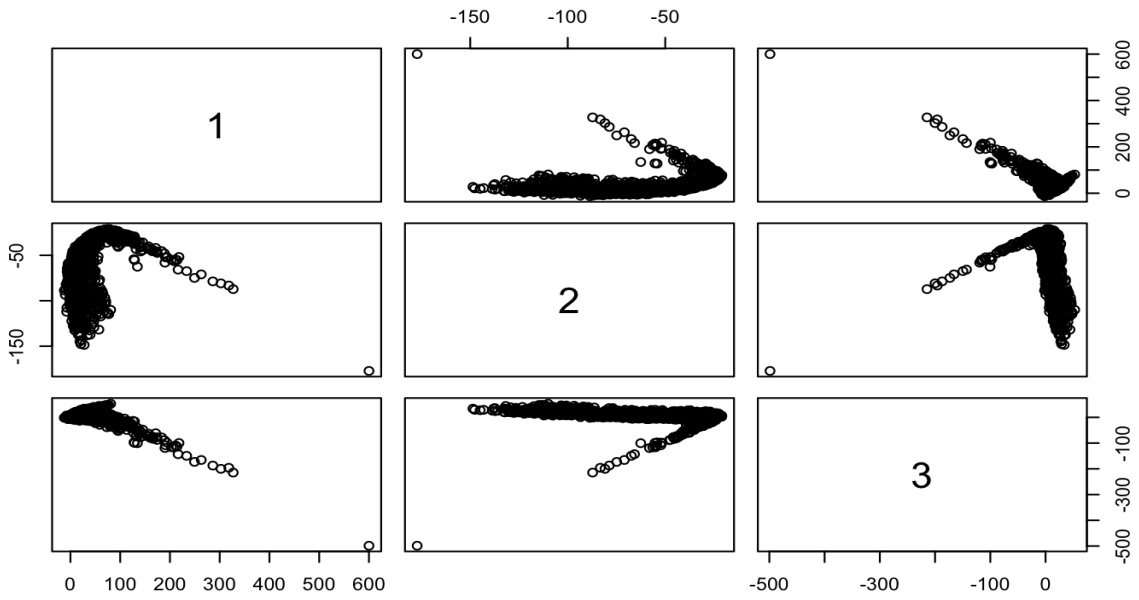
#orthogonalize Interesting Vectors

```
>orthmat<-orthogonalize(mvntest$Int[1:3,])
```

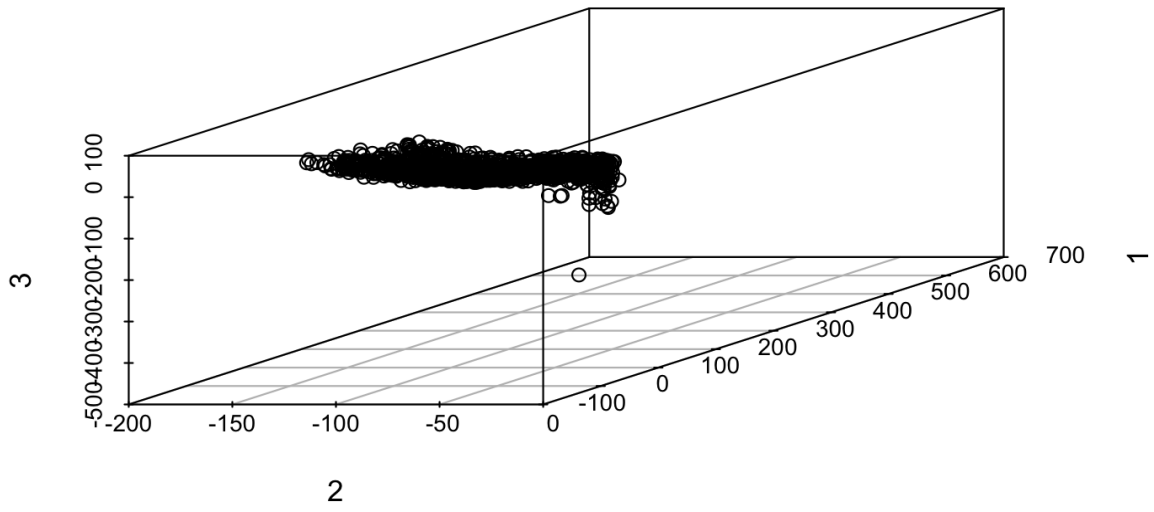
```
 [,1]      [,2]      [,3]       [,4]      [,5]       [,6]       [,7]      [,8]
[1,] 0.5529732 0.1000358 0.2102544 -0.1580989 0.3125378 -0.4351141 -0.1254809 0.5588038
```

#Explore Interesting Directions

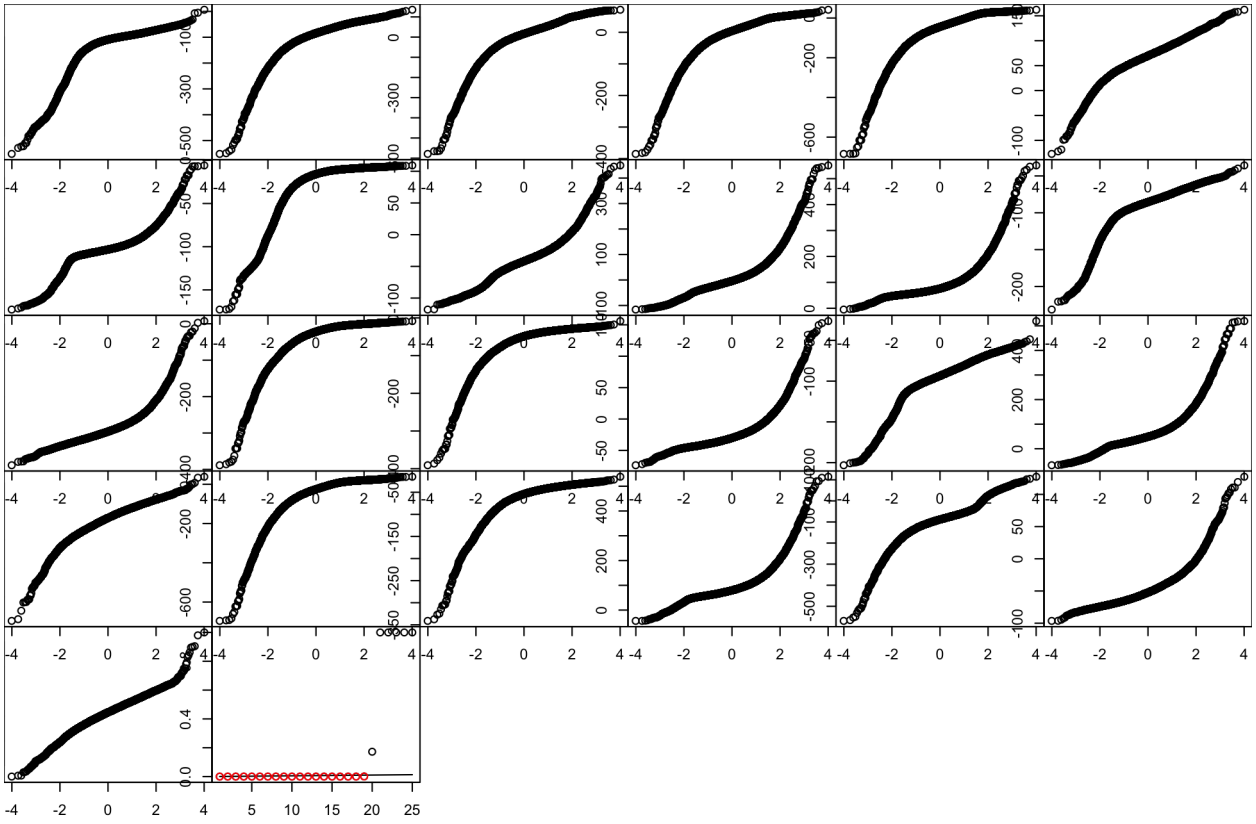
```
>expl<-explore(orthmat,pulsarMat,T,F,F)
```



>expl<-explore(orthmat,pulsarMat,F,T,F,c(2,1,3))



Suite 1 when PulsarData$v9 == '0'



#creating non pulsar data set

```
>nonPulsar <- pulsar[pulsar$v9 == '0',]
>pulsar0<-nonPulsar[2:9]
```

#Subsetted Data into Matricies

```
>nonPulsarMat<-as.matrix(nonPulsar)
```

#Multivariate Normal Test where Pulsar$v9 == '0'

```
>mvntest<-Multivariate.normal.test(nonPulsarMat, ntest = 25, Q = 0.05)
```

#Interesting Directions Found

```
      [,1]        [,2]        [,3]        [,4]        [,5]        [,6]
u1 -0.0172319372 -0.14189522 -0.12651861  0.81386333 -0.26696501 -0.15829365
u1 -0.4117427968 -0.05654860  0.40465991  0.19959930 -0.29858184  0.41385029
u1  0.3618354753  0.45486973  0.50334354 -0.04214046  0.15442994  0.16465447
u1  0.0986938348  0.53102149 -0.15053102  0.50358291 -0.07713939  0.19178010
u1 -0.0107050951  0.09992345  0.45486003  0.49537444 -0.08784278  0.36018923
u1 -0.0004703839  0.11535590  0.39294780  0.62141819 -0.62259146  0.12690318
u1  0.0141707509  0.13922503  0.42865446  0.39766388 -0.62247904 -0.27494735
u1 -0.2824457566 -0.16598429 -0.21156901 -0.76421682 -0.29925782  0.07107561
u1  0.3842580281 -0.38836917 -0.02585083 -0.35546843 -0.17075962  0.59075439
```

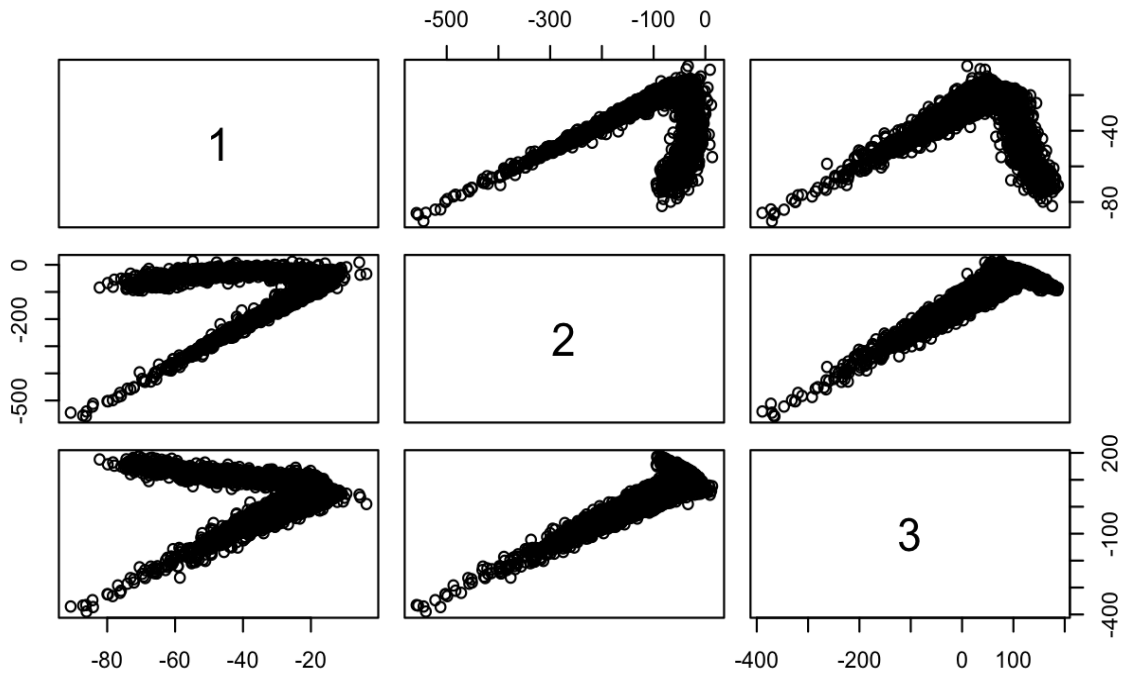
```
u1 -0.4452996696 -0.47448006  0.38834071  0.02246581 -0.61631591  0.15520191
u1  0.7071881881  0.07133782 -0.30267463  0.02603632  0.38723475 -0.42309616
u1  0.1617532460 -0.17506247  0.12371982  0.19768730 -0.65607669 -0.09280443
u1 -0.3031977129  0.41368426 -0.15929799 -0.09940190 -0.40517419 -0.28357039
u1 -0.6175312416 -0.12317901  0.51495100  0.33946254 -0.29583232 -0.36334384
u1  0.0990135962 -0.14177097 -0.37489468 -0.60620923 -0.36131004  0.25637289
u1 -0.3302438225  0.23323493 -0.28256621 -0.31358143 -0.76653681  0.05884190
u1  0.4132584687 -0.22046098  0.34878160  0.08435475 -0.41680379  0.44126568
u1  0.0911126885  0.48560220 -0.43660626  0.33553525 -0.04168804 -0.22996201
u1 -0.4893867344  0.06724892 -0.21777041  0.25933933  0.58679040  0.22254616
         [,7]        [,8]
u1 -0.44970438 -0.05125526
u1 -0.44852601 -0.40259389
u1 -0.28044163 -0.52671708
u1 -0.52415665 -0.33845247
u1 -0.27076349 -0.57170072
u1 -0.08314569  0.18837861
u1 -0.39527450 -0.13862266
u1 -0.04987571  0.40839961
u1  0.12407745  0.42537545
u1 -0.11199070 -0.09374862
u1  0.08557485  0.25733595
u1  0.57436079 -0.34621171
u1  0.59355223 -0.32372418
u1  0.01629326 -0.05719936
u1  0.29938425  0.41971266
u1  0.02987538 -0.25774004
u1 -0.32851595  0.41891114
u1  0.14555986 -0.61389898
u1 -0.27080055 -0.41727050
```

#orthogonalize Interesting Vectors

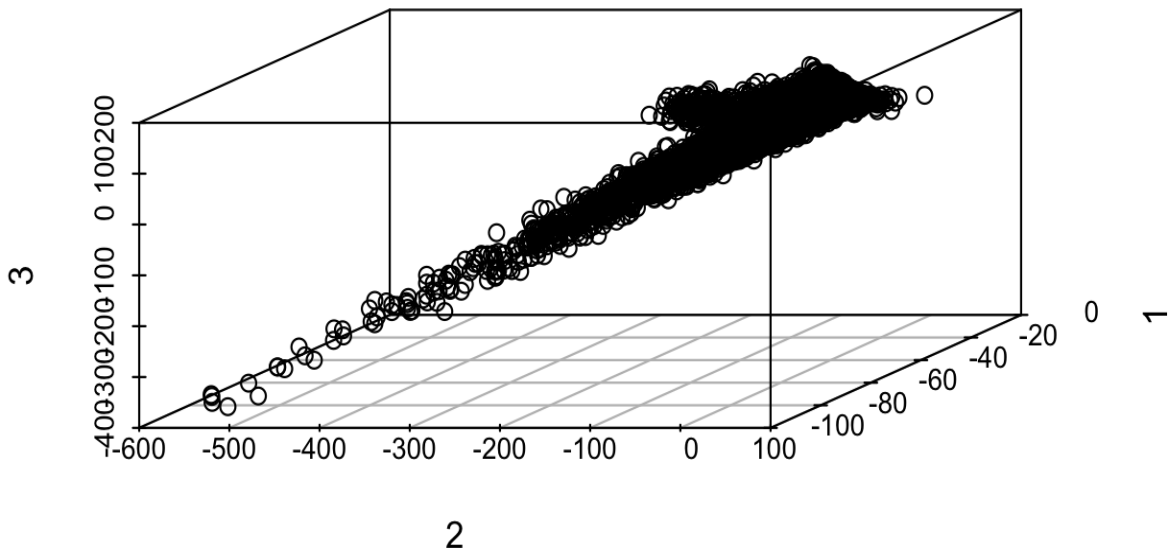
```
>orthmat<-orthogonalize(mvntest$Int[1:3,])

[,1]       [,2]       [,3]      [,4]      [,5]       [,6]       [,7]
[1,] -0.01723194 -0.1418952 -0.1265186 0.8138633 -0.266965 -0.1582937 -0.4497044
           [,8]
[1,] -0.05125526
```

#Exploring Interesting Directions

```
>expl<-explore(orthmat,nonPulsarMat,T,F,F)
```

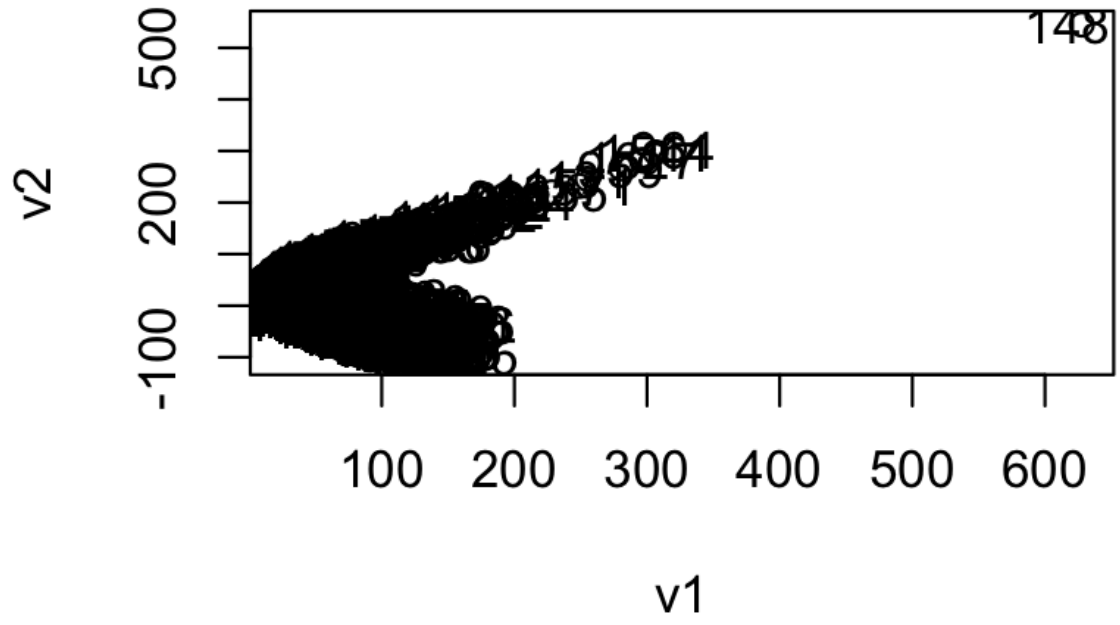


>expl<-explore(orthmat,nonPulsarMat,F,T,F,c(2,1,3))

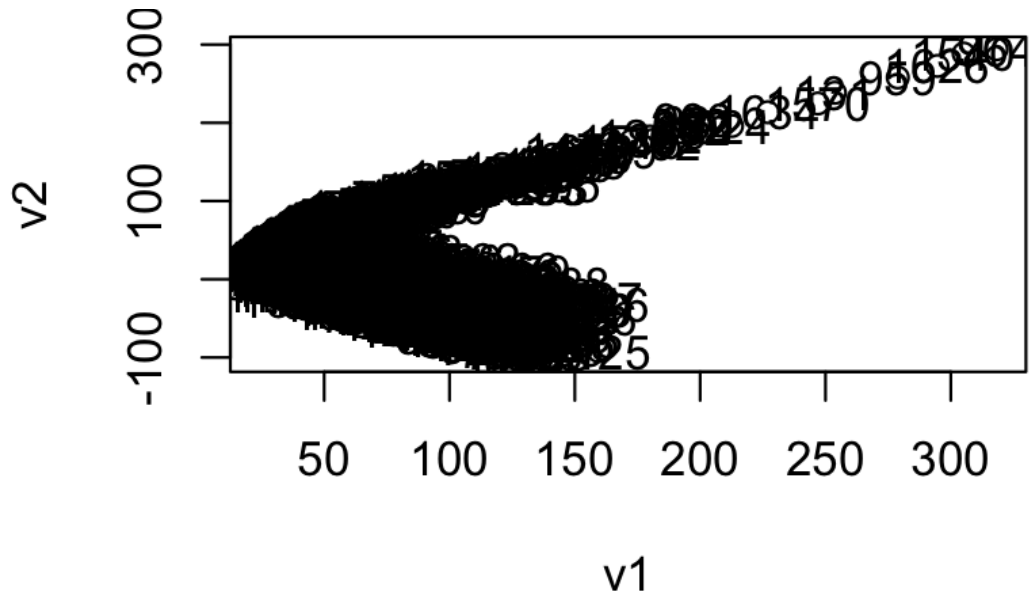


Suite 2: pulsar$v9 == '1'

>asym.contamination.plot(pulsarMat,0,c(.2,.7),0.95,T,F,F)

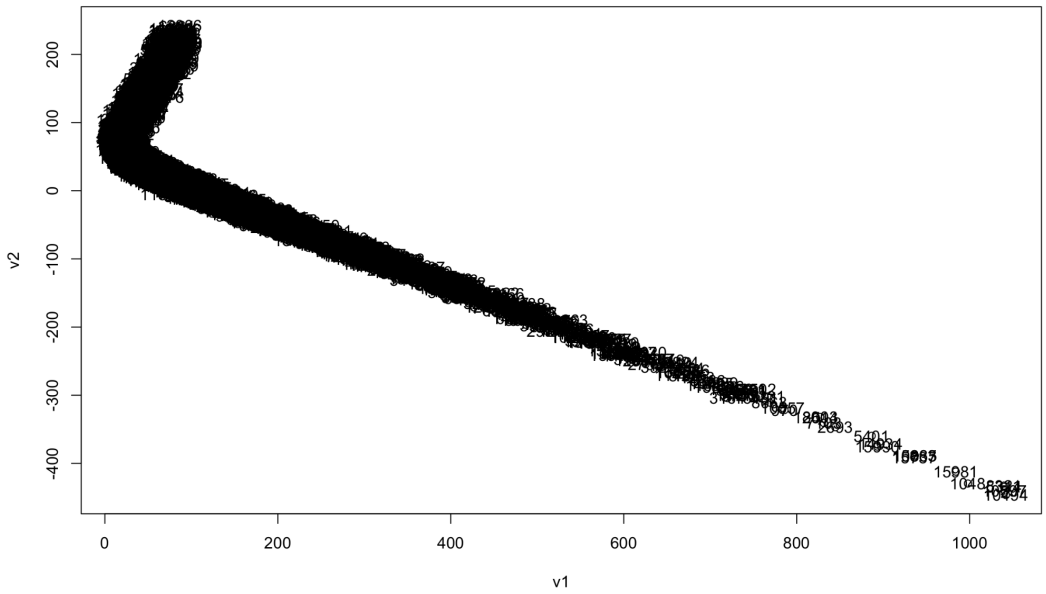


[1] "Would you like to identify the points to remove, then reanalyze? (Y/N)"
1: Y
2:
Read 1 item
[1] "How many points?"
1: 1
2:
Read 1 item
[1] 1481
[1] 0



Suite 2: for pulsar$v9 == '0'

>asym.contamination.plot(nonPulsarMat,0,c(.2,.7),0.95,T,F,F)

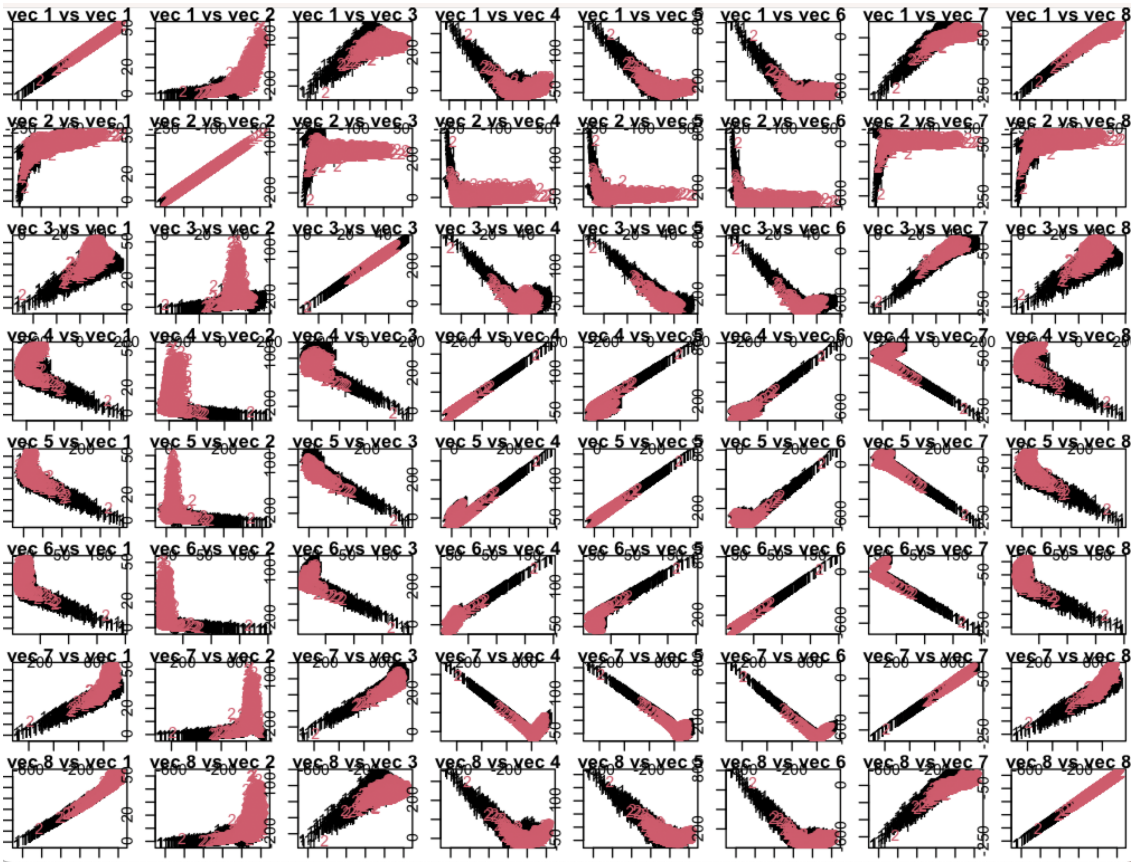


Suite 3: Perm.cov.test

```
>pulsartrain<- pulsar[,2:10]
>quartz(width = 8, height = 6)
>par(mar=c(0.75,0.75,0.75,0.75))
```

#9 is the id col or grouping variable and numvec=8 is the number of column indices starting at col1 that correspond to the numeric variables in the dataset that will have cov computed.

```
>Perm.cov.test(pulsartrain,9,8)
```



- **Test for a difference in means between the pulsar data and non-pulsar data**

```
>gui.twosample.perm()
[1] "pulsar" "10"     "0"      "10000"
***for breviety omitted list of permutations completed****
$H
         [,1]
[1,] 12369.96

$H1
        [,1]
[1,] 1373.91

$P
     [,1]
[1,]    0

$mu
[1] 2577.218211   59.872118    8.629143   -2.920215  -15.172732  -40.962737  -33.180979
[8]    6.105605   95.688615

$df
[1]     9.00 20751.41

$permP
[1] 0
```

*************EXPLANATION OF INPUT AND RESULTS*********

Gui.twosample.perm has 3 inputs: the data (in this case the pulsar dataset), the id column /grouping column (10) and the hypothesized difference of means of 0. In this case, the null hypothesis for this test is that the 2 groups have equal multivariate means. In other words, the difference of means between the mean vector of the 2 groups is 0. Lastly, we decided to run a permutation test of 10,000 by default.

H0: μX = μY. *(The multivariate vector of means for two groups (pulsar and non-pulsar) are equal.)*
HA: The multivariate vector of means for the two groups are not equal.

The output of the difference of means test shows the value of the Hotelling T^2 Statistics (has a value of 12369.96), and the value of H,1 (value of 1373.91) which is the sum of squared differences between the means of the two groups. With a p-value of <0.0001, we can reject the null hypothesis that the mean vectors for pulsar and non-pulsar are equal based on the evidence that (T^2: 12369.96; d.f: 9.00 20751.41; p-value <0.0001).

---

4. **Manova: (Class 6)**
    a. **Suite 6: gui.1waymanovatest, gui.multiway.manova.test.portmanteau, gui.multiway.manova.test.int**
        i. What does each do

gui.1waymanovatest: The function calculates the MANOVA statistic lambda and clambda for the original data and then generates nperm permutations of the ONE grouping variable, calculates the MANOVA statistic for each permutation, and computes the residuals, sum of squares, mean matrices and estimates the p-value as the proportion of permuted statistics that are equal to or less than than the original statistic.

gui.multiway.manova.test.portmanteau: calculates lambda and clambda for the original data and then generates nperm permutations of the multiple grouping variables with only one interaction term, calculates the MANOVA statistic (lambda and clambda) for each permutation, and estimates the p-value from the permuted lambda that are equal to or less than the original lambda while also computing the residuals, sum of squares, mean matrices.

gui.multiway.manova.test.int: calculates MANOVA for multiway designs with multiple interaction terms, includes an unique input parameter int.str which is list of vectors representing the interaction terms, calculates lambda and clambda for the original data and then generates nperm permutations of the multiple grouping variables with only one interaction term, calculates the MANOVA statistic lambda and clambda for each permutation, and estimates the p-value from the permuted lambda that are equal to or less than the original lambda while also computing the residuals, sum of squares, mean matrices.

APPLICATION:

ii. **Apply suites 1,2 and 3 to testmat.csv (4 groups, 100 each, look at data, you'll need to use read.csv to bring it in and assign a name to it in R remember to separate the groups when you apply it, remember to remove the group indicators when you apply it.)**

#create group testmats and remove col x1 and x2 and grouping col for normality testing

```
> testmat1 = as.matrix(subset(testmat[,1:8], testmat[,11] == 1))
> testmat2 = as.matrix(subset(testmat[,1:8], testmat[,11] == 2))
> testmat3 = as.matrix(subset(testmat[,1:8], testmat[,11] == 3))
> testmat4 = as.matrix(subset(testmat[,1:8], testmat[,11] == 4))
```

Suite1 Group1:

```
> mvntest<-Multivariate.normal.test(testmat1, ntest = 15, Q = 0.05)
$estimate.of.prob.not.normal$q.5
[1] 0.04239672

$estimate.of.prob.not.normal$q.95
[1] 0.1707497
```

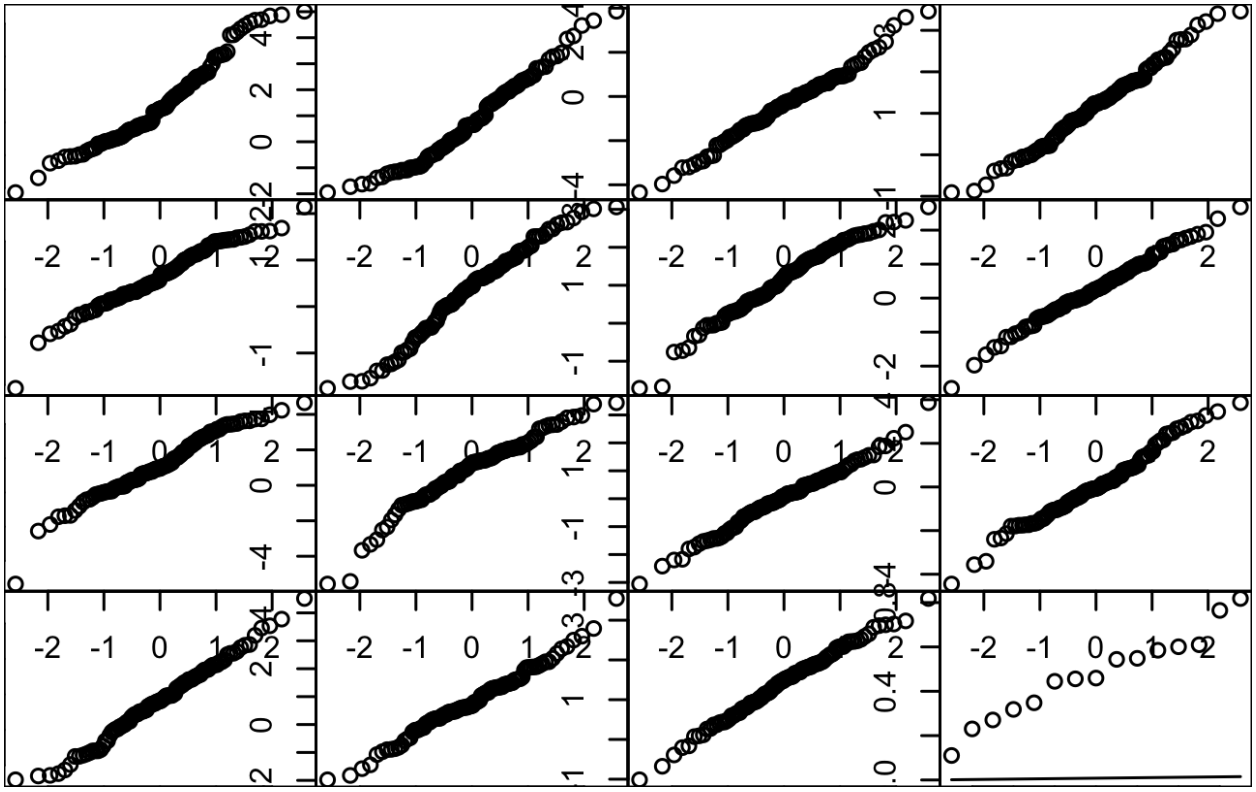
$estimate.of.prob.not.normal$q.99
[1] 0.2501058

$estimate.of.prob.not.normal$q.999
[1] 0.3506184

$pvals
 [1] 0.6705 0.3524 0.4210 0.8463 0.7172 0.3829 0.5112 0.4700
 [9] 0.6811 0.7608 0.8815 0.2738 0.1701 0.7826 0.2951

$Interesting.directions
[1] 0

$rejectingplots
[1] "none"



No interesting directions therefore cannot use gui.ortho and by extension gui.explore

Suite1 Group 2:

> mvntest<-Multivariate.normal.test(testmat2, ntest = 15, Q = 0.05)
$estimate.of.prob.not.normal$q.5
[1] 0.04239672

$estimate.of.prob.not.normal$q.95
[1] 0.1707497

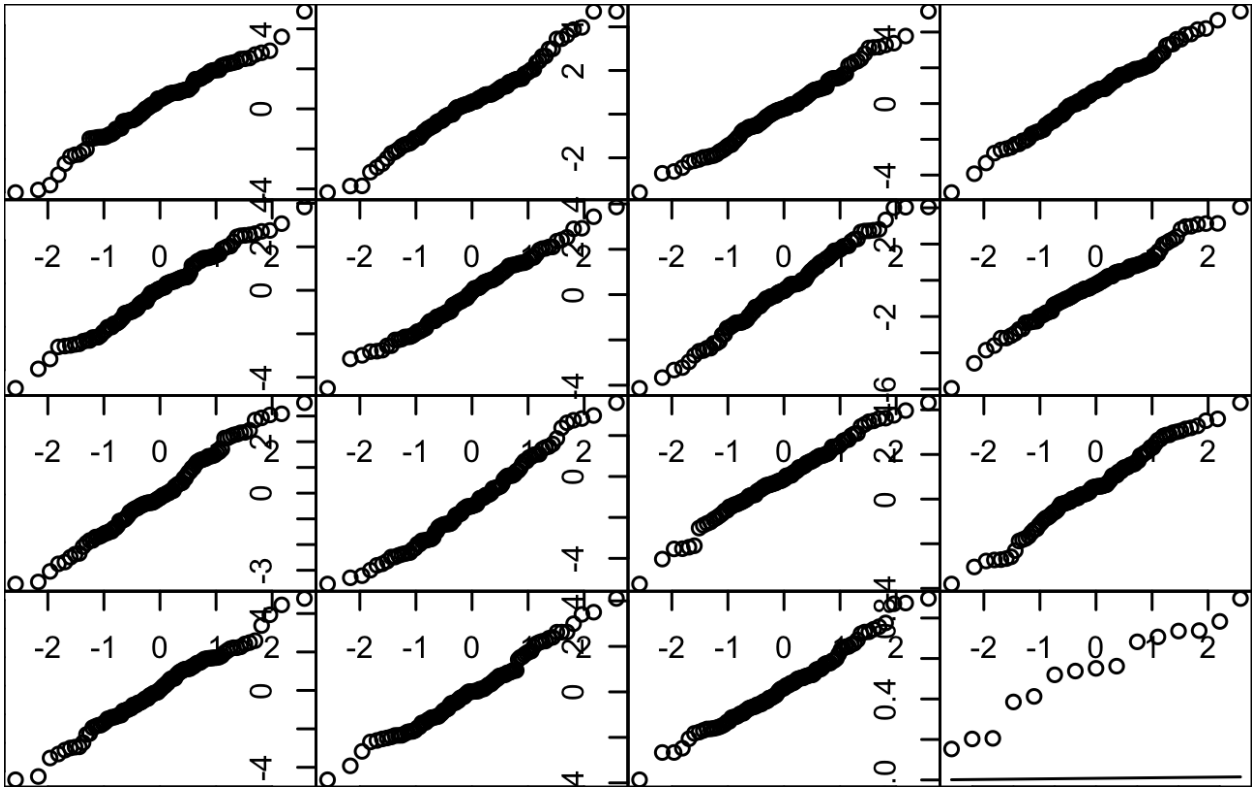
$estimate.of.prob.not.normal$q.99
[1] 0.2501058

$estimate.of.prob.not.normal$q.999
[1] 0.3506184

$pvals
 [1] 0.4128 0.8965 0.6825 0.7357 0.7378 0.7067 0.5622 0.5381
 [9] 0.1530 0.2055 0.5194 0.7833 0.5514 0.2025 0.3854

$Interesting.directions
[1] 0

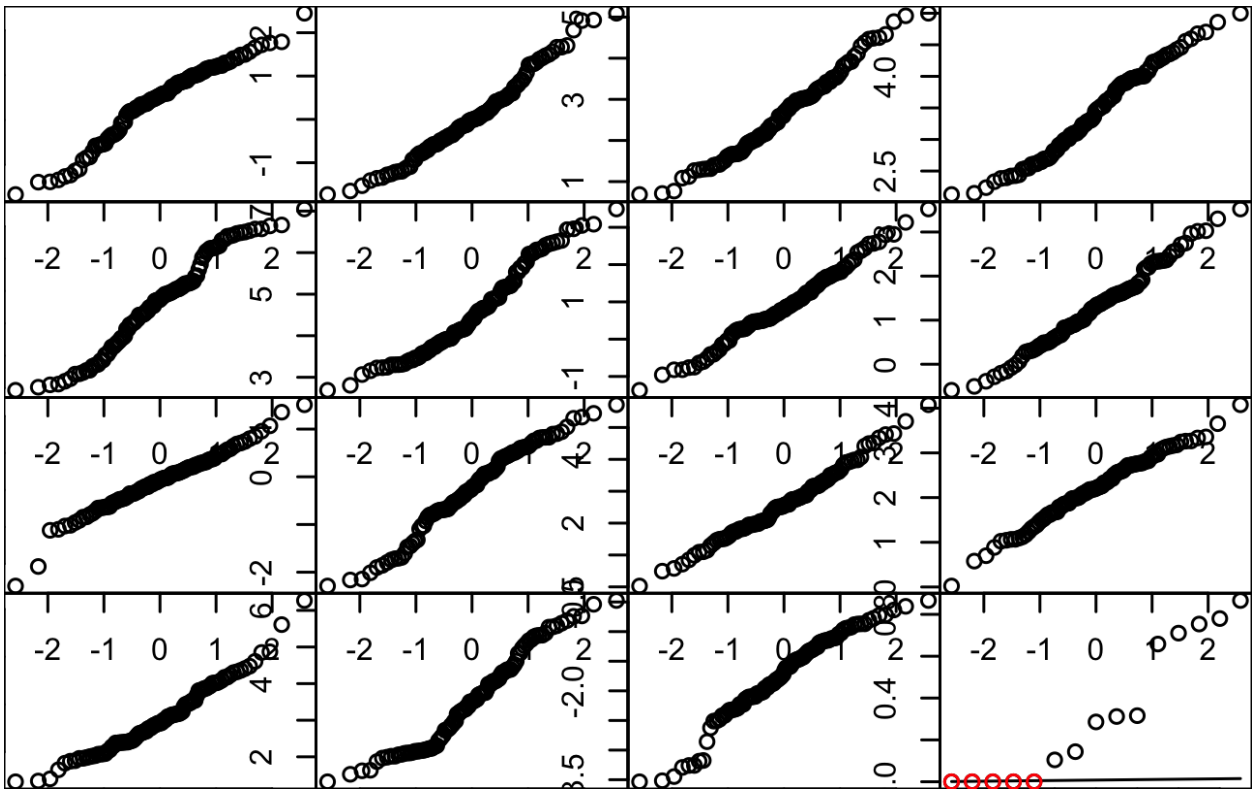
$rejectingplots
[1] "none"



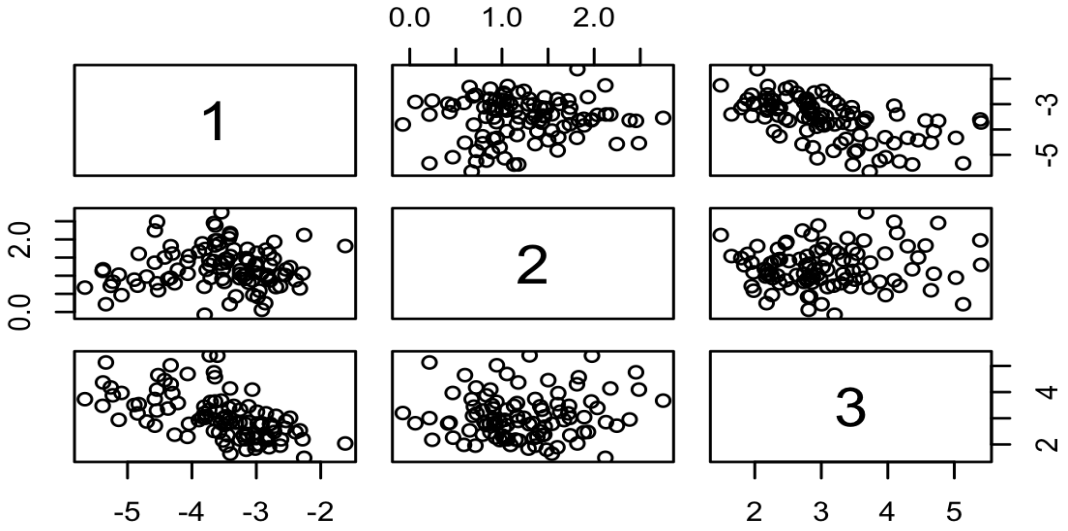
No interesting directions therefore cannot use gui.ortho and by extension gui.explore

Suite1 Group 3:

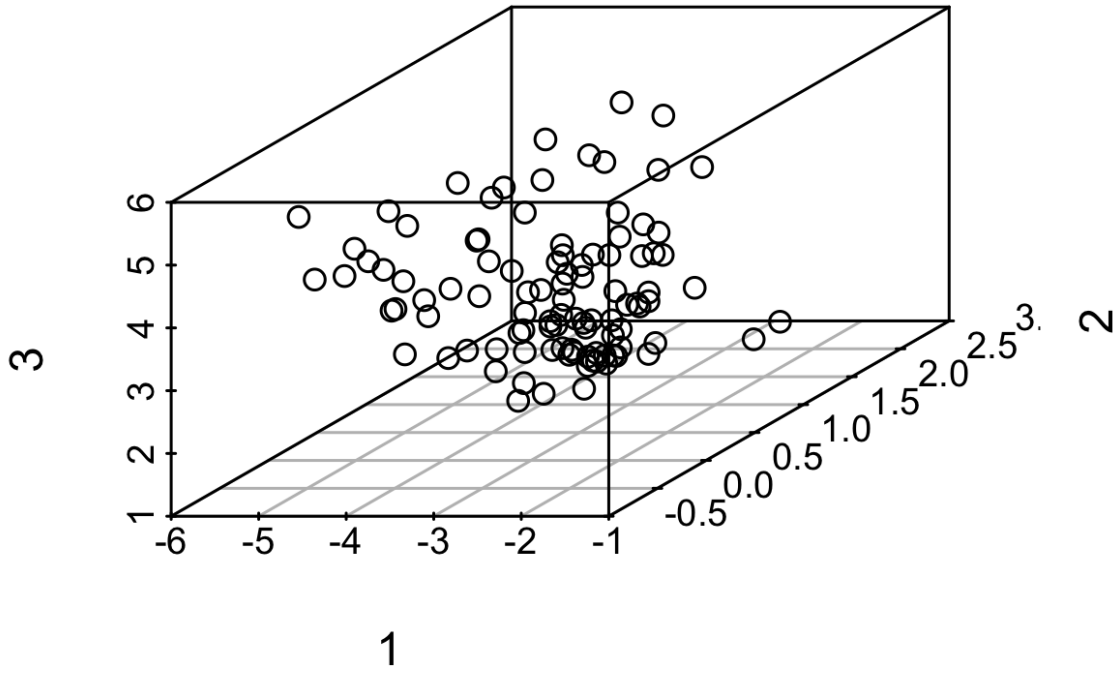
```
> mvntest<-Multivariate.normal.test(testmat3, ntest = 15, Q = 0.05)
$Interesting.directions
        [,1]       [,2]       [,3]       [,4]        [,5]
u1  0.2291419 -0.4733499 -0.23612522  0.3464874 -0.11981937
u1 -0.4995580  0.2594091  0.09979288  0.1255213  0.44920523
u1  0.1044430  0.2227213 -0.09820005 -0.1060298  0.05190422
u1 -0.3132834  0.4355368  0.09087080  0.4140339 -0.32993141
u1 -0.3437754  0.4520845 -0.29147183 -0.3087195  0.13809828
         [,6]       [,7]        [,8]
u1 -0.02596647 -0.66546913 -0.299574847
u1  0.23364579  0.48764812  0.404052101
u1  0.77760355  0.55757852 -0.018605954
u1  0.49396550  0.42381019  0.002810148
u1 -0.61205087  0.02689782  0.320583260

$rejectingplots
[1]  1  4  6 14 15

$pvals
 [1] 0.0000 0.1438 0.1045 0.0000 0.8663 0.0000 0.7538 0.7103
 [9] 0.3159 0.7800 0.3119 0.2859 0.6582 0.0000 0.0000
```



```
> orthmat<-orthogonalize(mvntest$Int[1:3,])
          [,1]       [,2]       [,3]      [,4]       [,5]
[1,] 0.2291419 -0.4733499 -0.2361252 0.3464874 -0.1198194
           [,6]       [,7]       [,8]
[1,] -0.02596647 -0.6654691 -0.2995748
> expl<-explore(orthmat,testmat3,T,F,F)
```

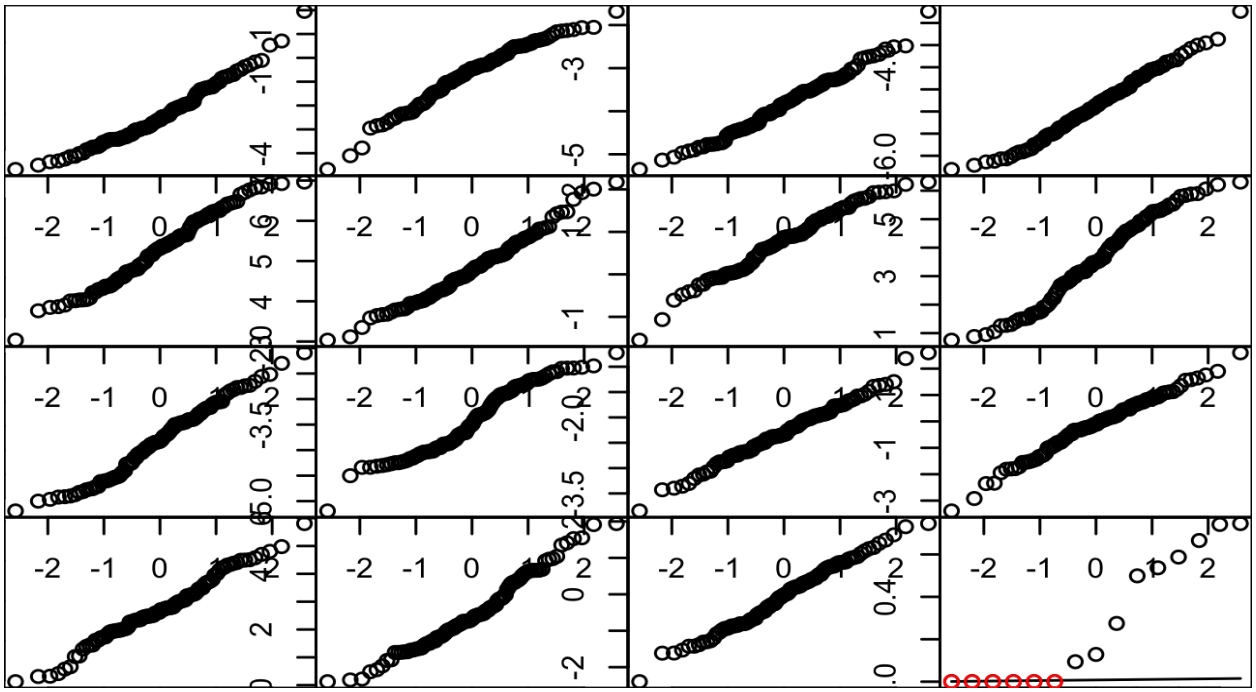



```
> expl<-explore(orthmat,testmat3,F,T,F,c(1,2,3))
```



Suite1 Group 4:

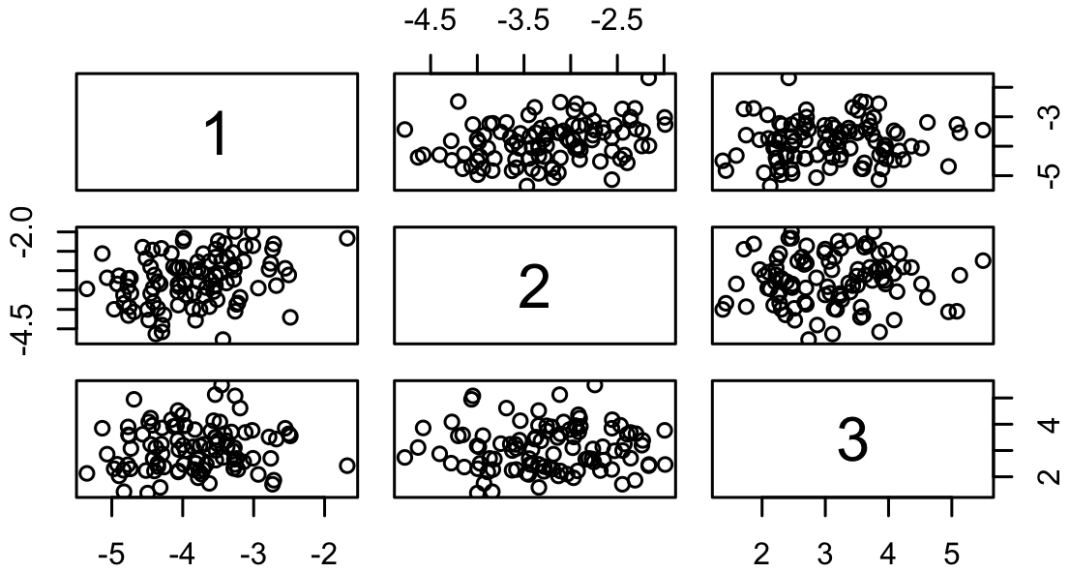
```
> mvntest<-Multivariate.normal.test(testmat4, ntest = 15, Q = 0.05)
$Interesting.directions
         [,1]       [,2]       [,3]        [,4]       [,5]
u1 -0.11277399  0.2139449 -0.64028112 -0.199587001 -0.25466792
u1  0.05500256 -0.4197648 -0.63265144  0.009740219  0.05066097
u1  0.15124795  0.2148695 -0.21814316 -0.264519733  0.15380178
u1 -0.45750840 -0.5526772  0.42988193 -0.346986471 -0.38731369
u1  0.07629654  0.3894471  0.05073182 -0.572924644 -0.63579385
u1  0.52197228 -0.1614776 -0.03224238  0.389662125  0.19662727
         [,6]       [,7]       [,8]
u1 -0.5746191  0.07036482 -0.30285417
u1 -0.2183462 -0.23131099 -0.56274607
u1  0.5140900  0.58235329  0.43164700
u1 -0.1559367  0.05242089 -0.05440398
u1 -0.1479739 -0.28379090 -0.07089456
u1  0.5424113  0.24485348 -0.39467485

$rejectingplots
[1]  3  4  6 10 11 13

$pvals
 [1] 0.7455 0.6656 0.0000 0.0000 0.4990 0.0000 0.5368 0.2747
 [9] 0.0941 0.0000 0.0000 0.7415 0.0000 0.1284 0.5876
```



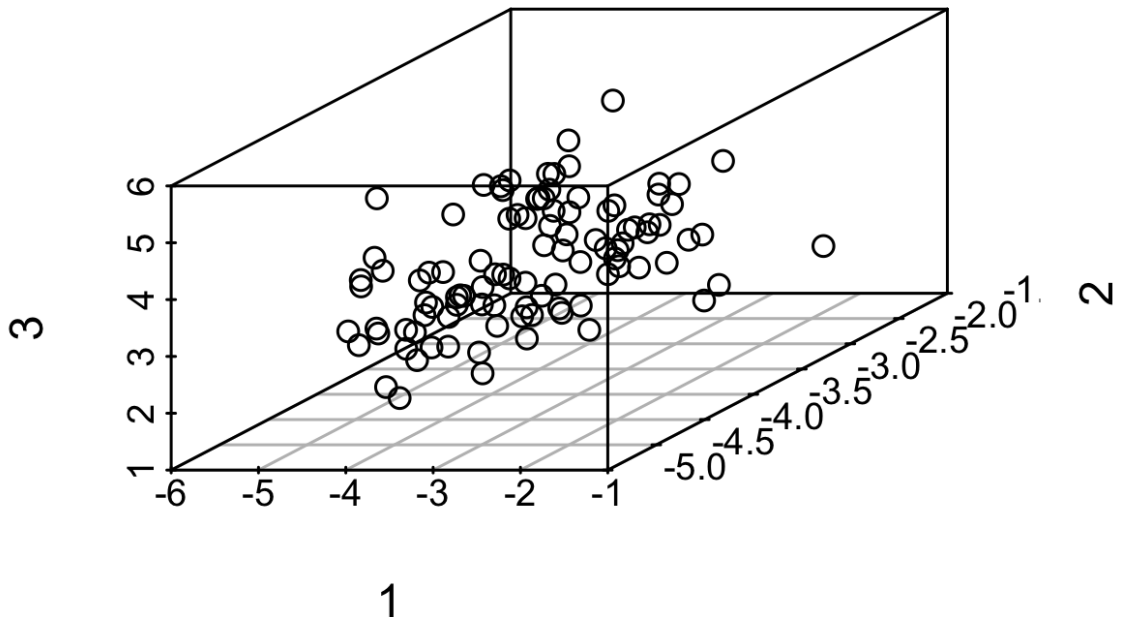
```
> orthmat<-orthogonalize(mvntest$Int[1:3,])
```

```
          [,1]      [,2]       [,3]      [,4]       [,5]
[1,] -0.112774 0.2139449 -0.6402811 -0.199587 -0.2546679
          [,6]       [,7]       [,8]
[1,] -0.5746191 0.07036482 -0.3028542
```

```
> expl<-explore(orthmat,testmat4,T,F,F)
```

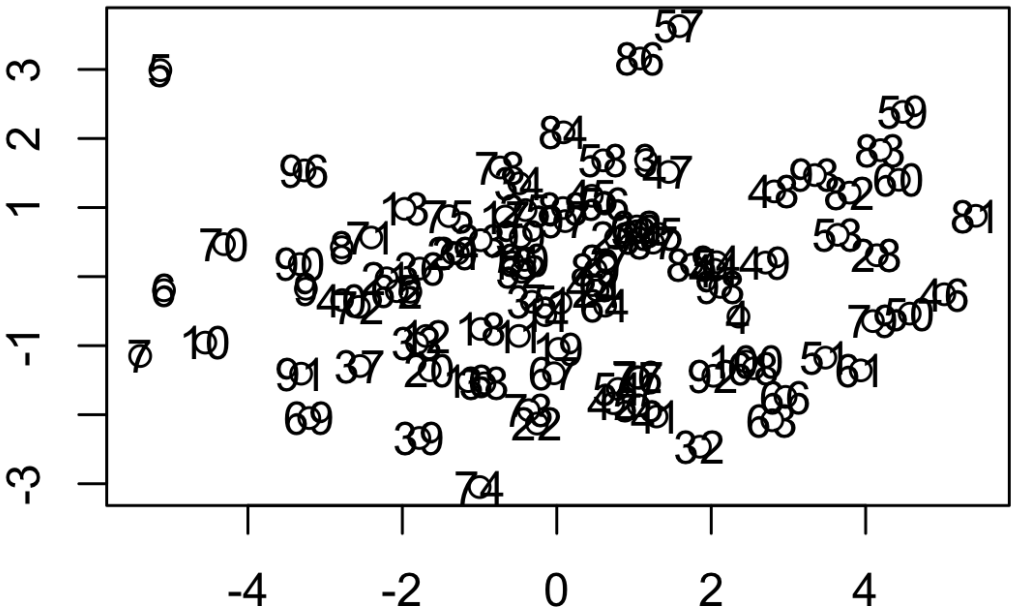



```
> expl<-explore(orthmat,testmat4,F,T,F,c(1,2,3))
```



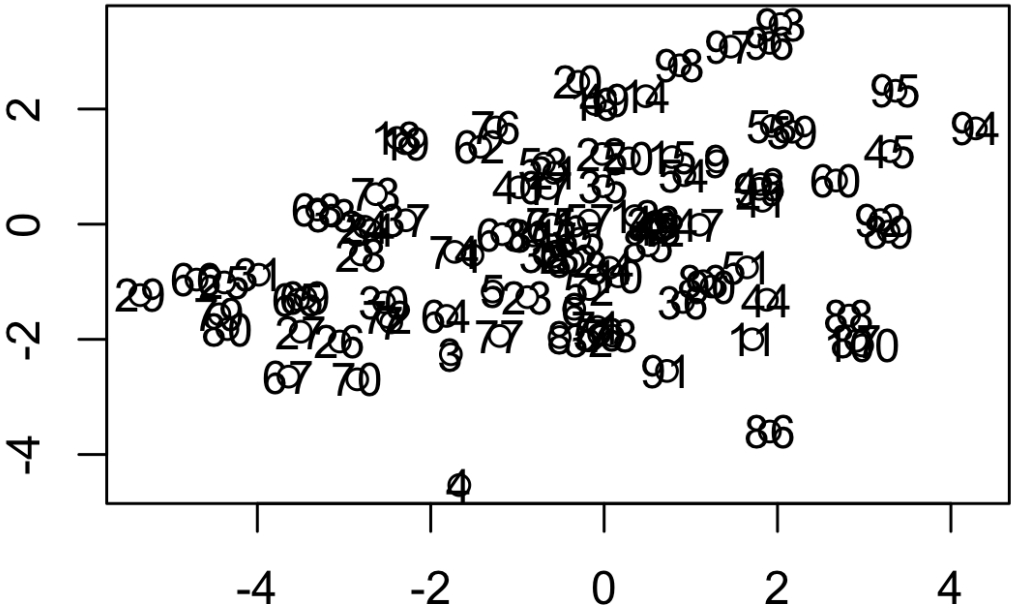
Suite 2 Group 1:

```
> asym.contamination.plot(testmat1,0,c(.2,.7),0.95,T,F,F)
```



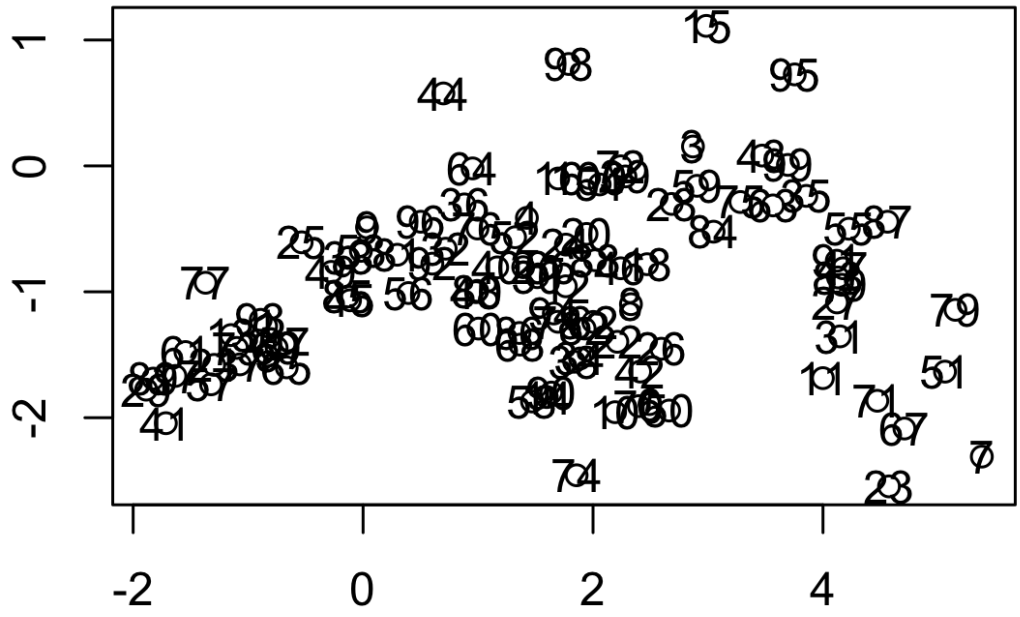
Suite 2 Group 2:

> asym.contamination.plot(testmat2,0,c(.2,.7),0.95,T,F,F)



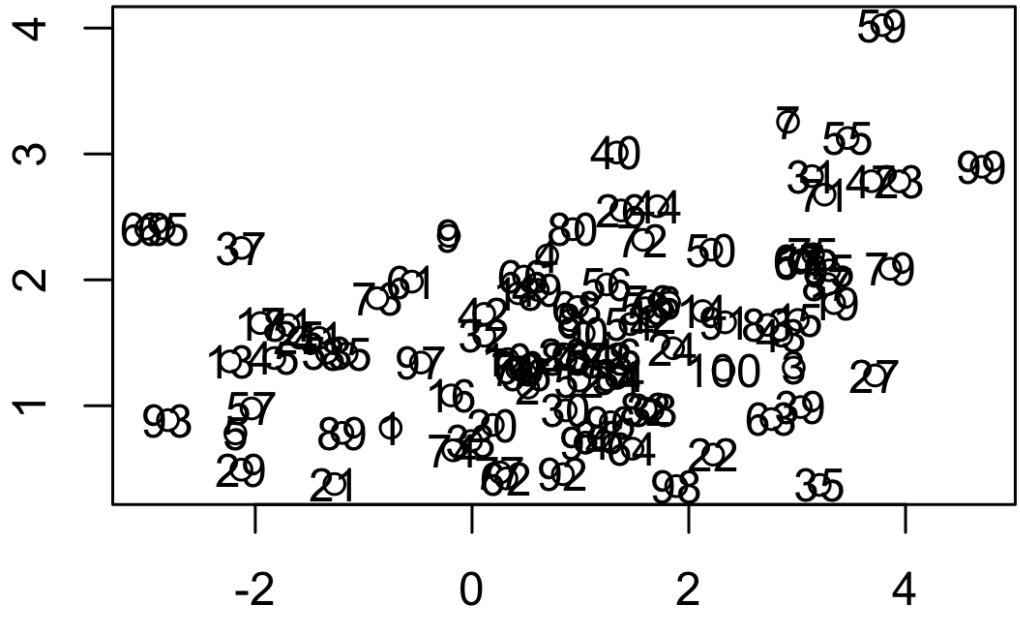
Suite 2 Group 3:

> asym.contamination.plot(testmat3,0,c(.2,.7),0.95,T,F,F)



Suite 2 Group 4:

> asym.contamination.plot(testmat4,0,c(.2,.7),0.95,T,F,F)

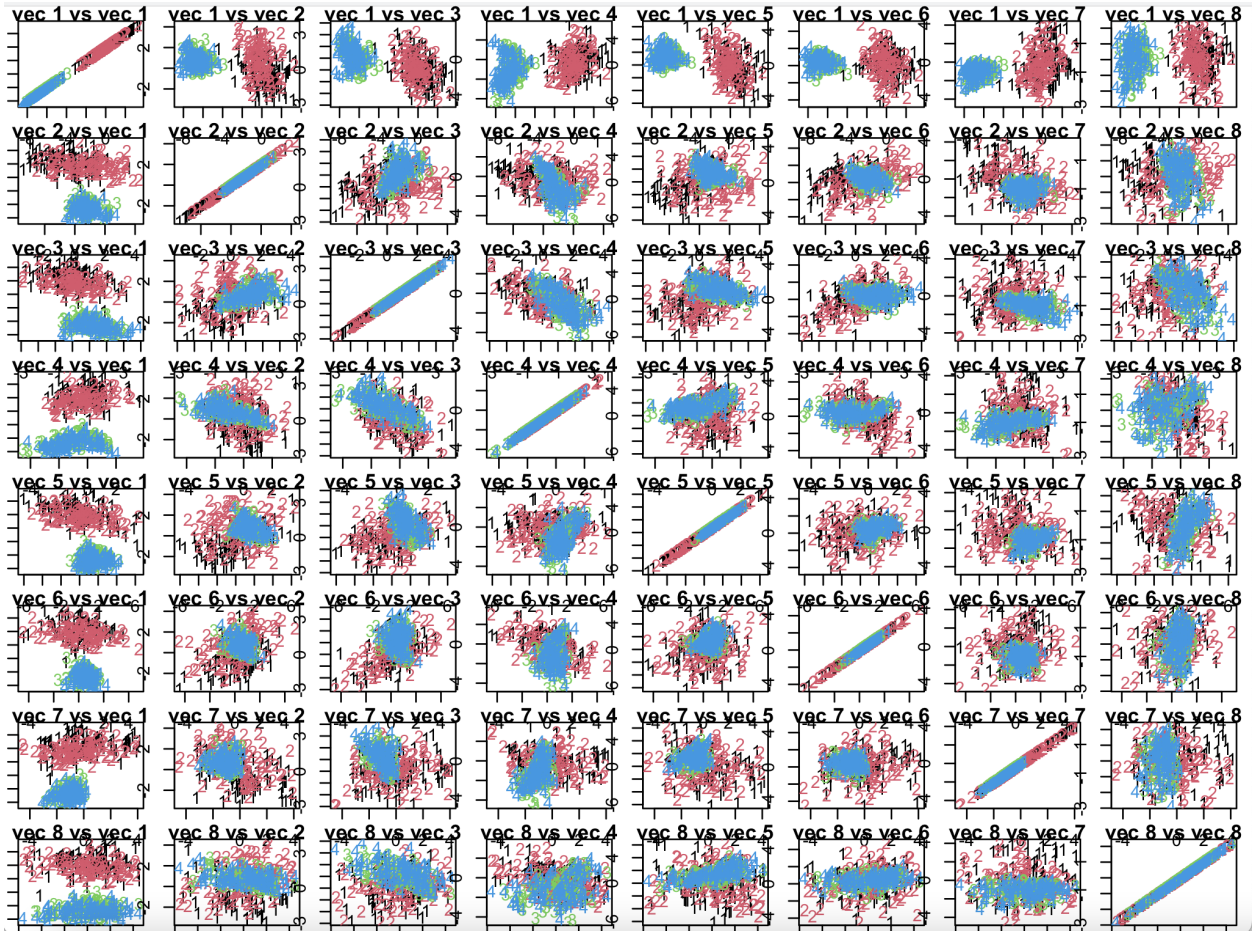


Suite 3 (Run on entire data set except factor columns have been removed):
#remove factor x1 and x2 columns including grouping col as id col

```
> Ptestmat = testmat[,-c(9:10)]
> quartz(width = 8, height = 6)
> par(mar=c(0.75,0.75,0.75,0.75))
```

#9 is the id col or grouping variable and numvec=8 is number of column indices starting at col1 that correspond to the numeric variables in the dataset that will have cov computed.

> Perm.cov.test(Ptestmat,9,8)



iii. Test for main effects and interactions in the data(x1 smth about if its in the climate model and x2 smth about time or whatever are factors rest are variables)

#remove the grouping col for manova

```
> Mtestmat = testmat[1:10]
> Mtestmat = data.matrix(Mtestmat)
> manova = gui.multiway.manova.test.portmanteau()
[1] "Mtestmat" "c(9:10)"  "10000"
[1] 0.92537679 0.08300489 0.89375097
```

---

[1] 0.92537679 0.08300489 0.89375097

****omitted code list of number of iterations and residuals to use in resid plots if necessary****

$p
[1] 1e-04 0e+00 0e+00
$cp
[1] 1e-04 0e+00 0e+00

***** EXPLANATION OF INPUT AND OUTPUT *****
The inputs were the data matrix Mtestmat which is testmat with the last column (splits the data into groups 1-4) removed, the idcol vector c(9:10) indicating which of the columns are the factor variables and nperm = 10000 indicating the number of permutations.
The output included a bunch of residuals that I decided to leave out because you would only really use that to run residplot2 like we did in quiz 2!
The output of the manova portmanteau test p and cp values are necessary to reach a conclusion about the main effects and interactions in the data. The p values are all less than 0.05 so we can conclude that factor X1 has an statistically significant effect on the 8 dependent variables, factor X2 has an statistically significant effect on the 8 dependent variables and factor X1 and factor X2 have a significant interaction effect.

5. **Mancova:(Class 6)**
    a. **Suite 7: Residplot2**
        i. **Residplot2**: This function creates a scatter plot matrix (using the "pairs" function) to visualize the residuals of a MANOVA test. The resulting matrix is then plotted using the "pairs" function, with the residuals as the dependent variable and the other variables as independent variables.

6. **Principle components: Factor analysis: Canonical correlation**
    a. **Suite 8: gui.princomp, princomp, factanal, eigen, cancor, p.perm, p.asym**
        i. What does each do

Gui.princomp: a graphical interface implementation for performing principal component analysis (PCA) with bootstrapped confidence intervals on the eigenvalues of the correlation or covariance matrix of the input data matrix.

a. Inputs arguments include: Dataset (input data matrix) , Alpha value (significance level) , F (whether to scale covariance matrix) , Nboot (number of bootstrap iterations).
b. This graphical interface first computes the eigenvalues and eigevectors of the covariance matrix, and then performs the bootstrapped confidence interval calculations on the eigenvalues.
c. Returns list containing the eigenvalues and eigenvectors, bootstrapped and normal confidence intervals for the eigenvalues, proportion of variance explained by each principal component, and covariance matrices for each principal component.

Princomp: Principal component analysis (PCA) used for dimensionality reduction of a dataset; used in exploratory data analysis and for making decisions in predictive models. PCA commonly used for dimensionality reduction by using each data point onto only the first few principal components (mostly first and second dimensions) to obtain lower-dimensional data while keeping as much of the data's variation as possible.

a. First principle component can be defined as the direction that maximizes the variance of the projected data.
b. Principal components are usually analyzed by eigendecomposition of the data covariance matrix or the singular value decomposition of data matrix.

Factanal: Performs maximum likelihood factor analysis on a covariance matrix or a data matrix. It takes a data set as input and returns the factor loadings (a correlation matrix of variables and extracted factors. Values of matrix indicate the strength and direction of the relationship between factor and variable) of each variable on each factor, as well as eigenvalues and communalities.

Eigen: Computes eigenvalues and eigenvectors of numeric (double, integer, logical) or complex matrices.

Cancor: Used to perform canonical correlation analysis in r. Canaoncial Correlation Analysis measures the linear relationship between two sets of variables.The canonical correlation analysis seeks linear combinations of the y variables which are well explained by linear combinations of the x variables. The relationship is symmetric as 'well explained' is measured by correlations. It also finds the correlation between two sets of variables while accounting for variance separately.

a. Input Arguments: In R, the cancor function takes in: (X (numeric matrix; n×p1) and Y(numeric matrix; (n x p2))
b. The output of the the function includes the canonical correlations (a vector of correlation coefficients between the canonical variates), the canonical coefficients (the weights applied to each variable in each set to form the canonical variates), and the canonical variates (the linear combinations of the original variables that maximize the canonical correlation).

P.perm: Runs a permutation test to assign the statistical significance of canonical correlation coefficients. To test the hypothesis of no correlation between two sets (X, Y) of variables, the values of one variable (Y) are randomly reassigned.

a. Input Arguments: *p.perm(X, Y, nboot = 999, rhostart = 1, type = "Wilks")*
    i. Where X (array of independent variables), Y (array of dependent variables), nboot (number of permutations calculated),
    ii. nrhostart (index of the largest canonical correlation coefficient included in the calculation of the test statistic). Determines how many correlation coefficients are included in the calculation of the test statistic
    iii. type (test statistic to be used. One of "Wilks" (default), "Hotelling", "Pillai", or "Roy").

P.asym: This function runs asymptotic tests to assign the statistical significance of canonical correlation coefficients. the function p.asym calculates p-values for each test statistic: the first p-value is calculated including all canonical correlation coefficients, the second p-value is calculated by excluding the third p-value is calculated by excluding rho[1], rho[2] etc., allowing assessment of the statistical significance of each individual correlation coefficient.

a. Input Arguments: p.asym(rho, N, p, q, tstat = "Wilks")
    i. Where rho is the vector containing the canonical correlation coefficients
    ii. N is the number of observations for each variable
    iii. P is the number of independent variables
    iv. Q is the number of dependent variables
    v. Tstat is test statistic to be used. One of "Wilks" (default), "Hotelling", "Pillai", or "Roy"

APPLICATION:

ii. **How can canonical correlation analysis be used to support Factor analysis conclusions?**

Canonical correlation analysis can be used to determine the degree of association between two sets of factors; one set obtained from one group of variables and the other set obtained from a different group. CCA can be used to validate the findings of Factor analysis by assessing the degree of association between the identified factors and a different set of variables. If the canonical correlations between the sets of factors are statistically significant, this suggests that there is a strong association between the two sets of factors and supports the conclusions of the factor analysis. Conversely, if the canonical correlations are not statistically significant, this suggests that there is little or no association between the two sets of factors and may raise questions about the validity of the factor analysis findings.

7. Discriminant analysis (Class 10!)

a. Suite 9: LDA,QDA, tree, rpart, randomForest, svm

i. What does each do

LDA (Linear Discriminant Analysis): model for classification and dimensionality reduction. Used when the covariance matrix does not depend on the population. Also used to find a linear transformation that classifies different classes. It fits a linear discriminant model to the data using maximum likelihood estimation and returns a list of output that includes the estimated coefficients, the prior probabilities of each class, the means of the predictor variables for each class, and the linear discriminants.

***lda(formula, data, ..., subset, na.action)*

a. Input Arguments: A formula of the form groups ~ x1 + x2 + ...The response is the grouping factor and the right-hand side specifies the (non-factor) discriminators. Subset: An index vector specifying the cases to be used in the training sample

$$\delta_k(\mathbf{x}) = \mathbf{x}^T \mathbf{\Sigma}^{-1} \mu_k - \frac{1}{2} \mu_k^T \mathbf{\Sigma}^{-1} \mu_k + \log \pi_k \,,$$

** where K is classes: LDA assumes equal covariances among k classes, classifier becomes linear*

QDA (Quadratic Discriminant Analysis): similar to Linear discriminant analysis except relax the assumption that the mean and covariance of all the classes were equal. Required calculating it separately. QDA assumes that the data from each class comes from a multivariate normal distribution, just like LDA. However, it does not assume equal covariance matrices between classes. Itallows each class to have its own covariance matrix, which can lead to more complex decision boundaries.
***qda(formula, data, ..., subset, na.action)*

a. Output contains information about the model fit, including the estimated class probabilities and covariance matrices.

$$\delta_k(\mathbf{x}) = -\frac{1}{2} \log |\mathbf{\Sigma}_k| - \frac{1}{2} (\mathbf{x} - \mu_k)^T \mathbf{\Sigma}_k^{-1} (\mathbf{x} - \mu_k) + \log \pi_k \,.$$

tree: produces decision trees, tends to usually overfit.

rpart: an alternative approach to fitting trees. Usually produces much nicer trees in R since it performs cross-validation by default. Usually produces much smaller trees than the tree function. Works by recursively splitting the data based on the values of predictor variables.

a. Input arguments: a formula object that specifies the response variable and predictor variables, and a data frame containing the data to be used for building the tree.

randomForest: creates a forest of decision trees and combines their predictions to make a final prediction. One of the advantages of randomForest is that it reduces overfitting because each decision tree is trained on a different subset of features and data points.

SVM(Support Vector Machines): Used for classification (mainly used for classification) and regression analysis. Main characteristics of SVMs is that they use a subset of training data points, known as, support vectors, to construct decision boundary or hyperplane. SVMs are also robust to other changes in data, such as median and outliers, since the decision boundary is determined only by the support vectors. Important to note (as per the slides): Logistic regression is similar to SVM, but instead of using only some of the data points, logistic regression assigns a weight to each data point to determine its influence on the decision boundary.

APPLICATION:

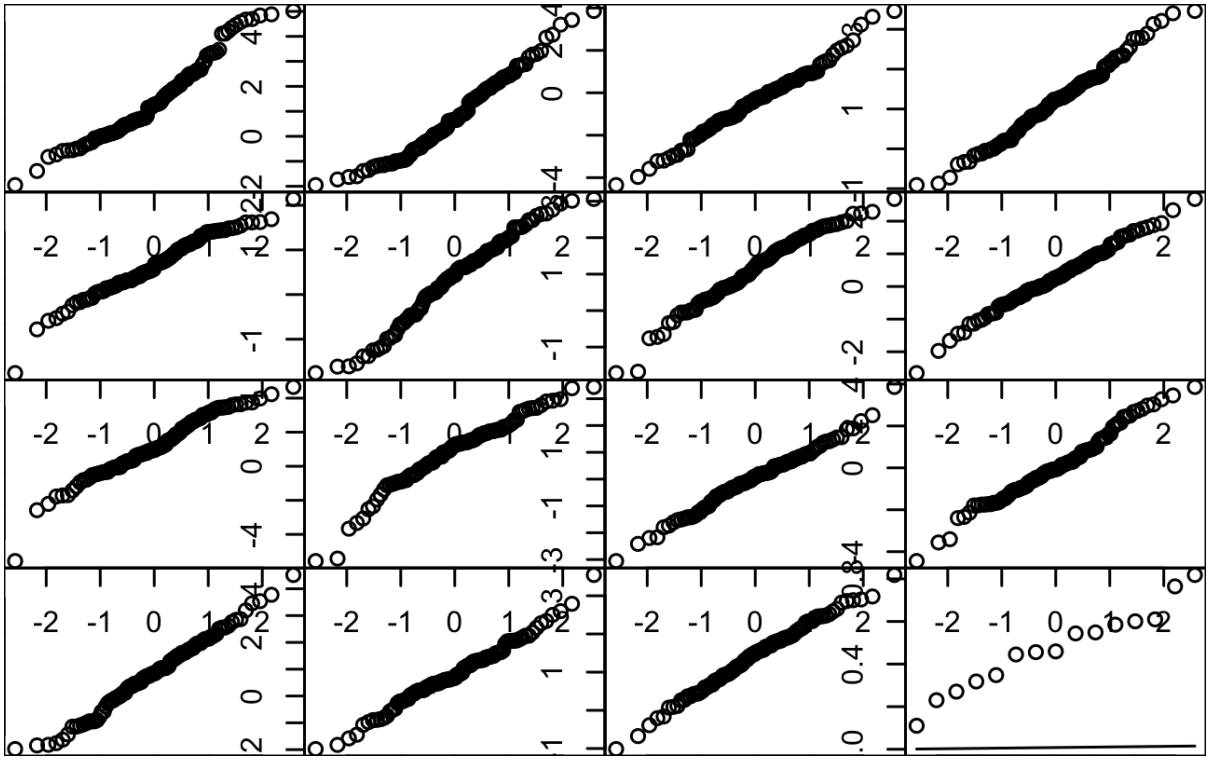
ii. Apply suites 1,2 and 3 to testmat.csv

#create group testmats and remove col x1 and x2 and grouping col for normality testing

```
> testmat1 = as.matrix(subset(testmat[,1:8], testmat[,11] == 1))
> testmat2 = as.matrix(subset(testmat[,1:8], testmat[,11] == 2))
> testmat3 = as.matrix(subset(testmat[,1:8], testmat[,11] == 3))
> testmat4 = as.matrix(subset(testmat[,1:8], testmat[,11] == 4))
```

Suite1 Group1:

```
> mvntest<-Multivariate.normal.test(testmat1, ntest = 15, Q = 0.05)
$estimate.of.prob.not.normal$q.5
[1] 0.04239672

$estimate.of.prob.not.normal$q.95
[1] 0.1707497

$estimate.of.prob.not.normal$q.99
[1] 0.2501058

$estimate.of.prob.not.normal$q.999
[1] 0.3506184

$pvals
 [1] 0.6705 0.3524 0.4210 0.8463 0.7172 0.3829 0.5112 0.4700
 [9] 0.6811 0.7608 0.8815 0.2738 0.1701 0.7826 0.2951

$Interesting.directions
[1] 0

$rejectingplots
[1] "none"
```

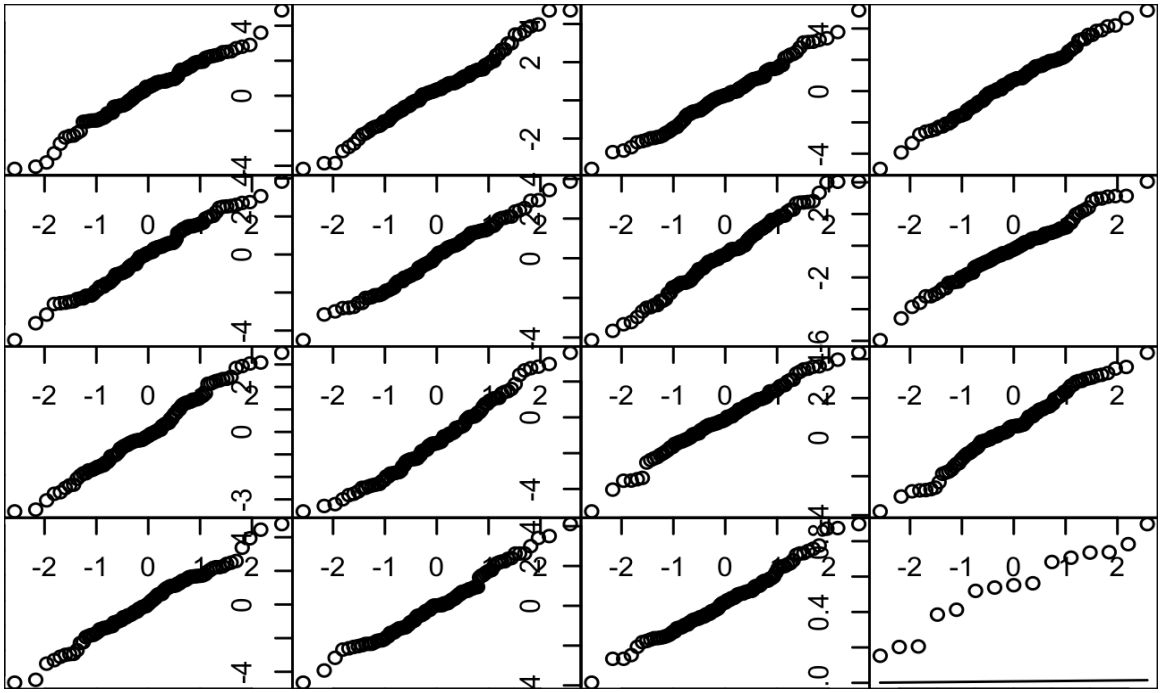


No interesting directions therefore cannot use gui.ortho and by extension gui.explore

Suite1 Group 2:

```
> mvntest<-Multivariate.normal.test(testmat2, ntest = 15, Q = 0.05)
$estimate.of.prob.not.normal$q.5
[1] 0.04239672

$estimate.of.prob.not.normal$q.95
[1] 0.1707497

$estimate.of.prob.not.normal$q.99
[1] 0.2501058

$estimate.of.prob.not.normal$q.999
[1] 0.3506184

$pvals
 [1] 0.4128 0.8965 0.6825 0.7357 0.7378 0.7067 0.5622 0.5381
 [9] 0.1530 0.2055 0.5194 0.7833 0.5514 0.2025 0.3854

$Interesting.directions
[1] 0

$rejectingplots
```

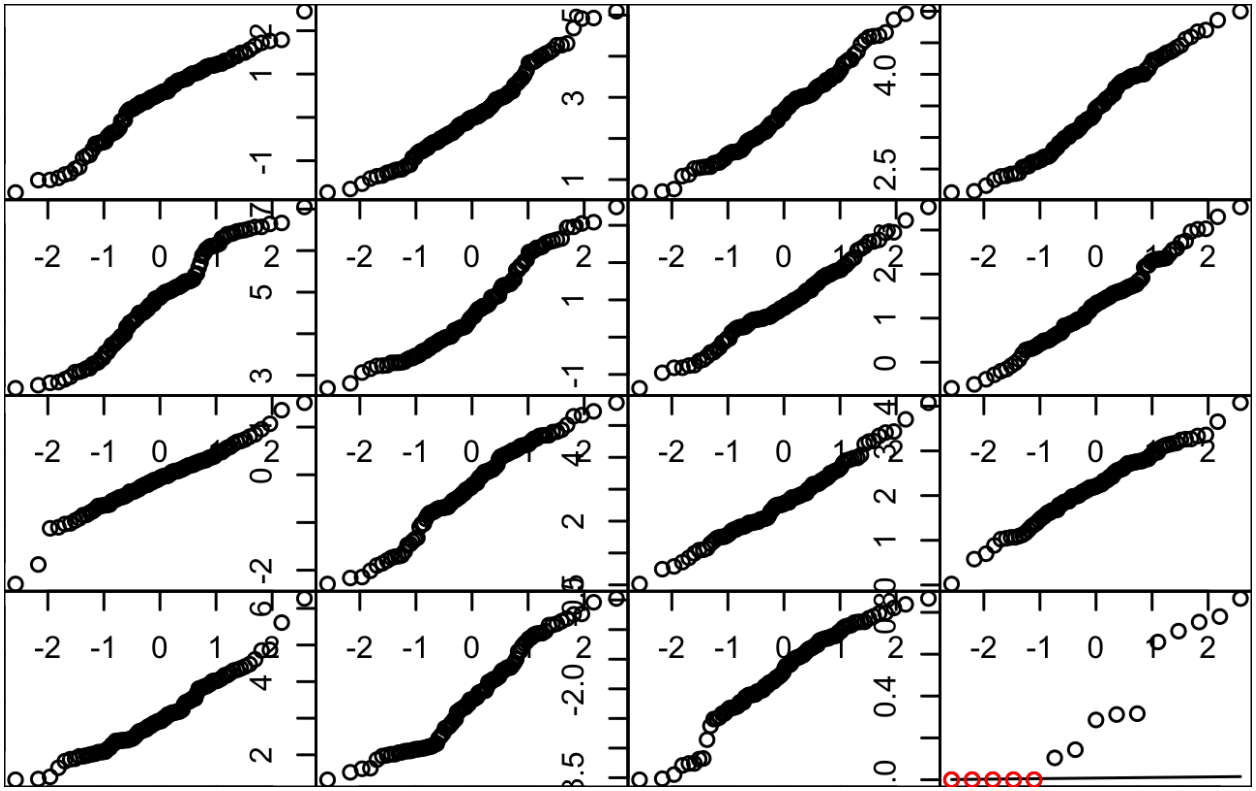
[1] "none"



No interesting directions therefore cannot use gui.ortho and by extension gui.explore

Suite1 Group 3:

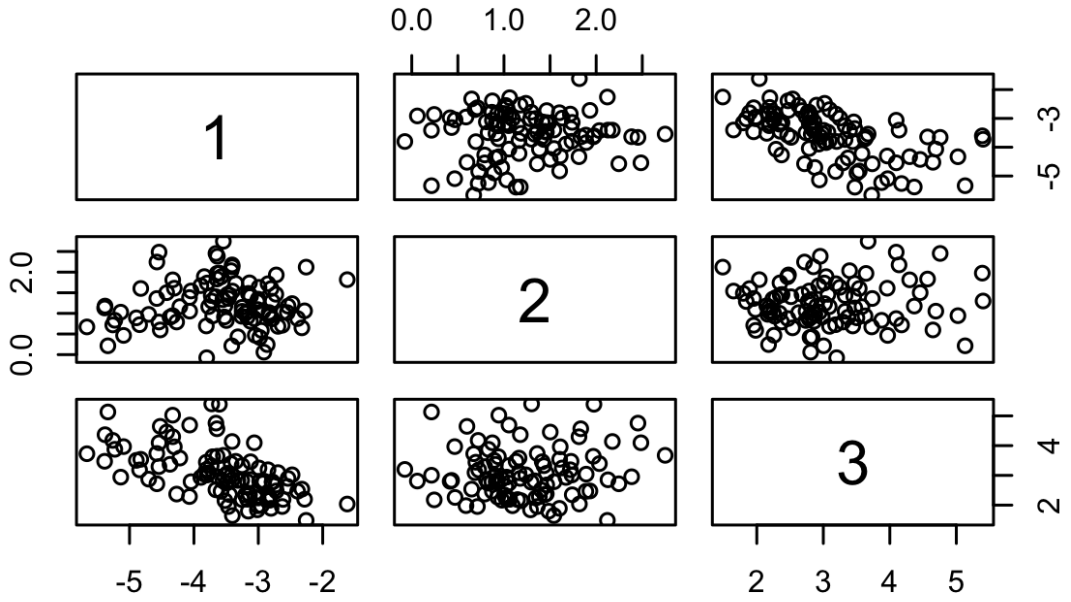
```
> mvntest<-Multivariate.normal.test(testmat3, ntest = 15, Q = 0.05)
$Interesting.directions
         [,1]       [,2]        [,3]       [,4]        [,5]
u1  0.2291419 -0.4733499 -0.23612522  0.3464874 -0.11981937
u1 -0.4995580  0.2594091  0.09979288  0.1255213  0.44920523
u1  0.1044430  0.2227213 -0.09820005 -0.1060298  0.05190422
u1 -0.3132834  0.4355368  0.09087080  0.4140339 -0.32993141
u1 -0.3437754  0.4520845 -0.29147183 -0.3087195  0.13809828
          [,6]       [,7]         [,8]
u1 -0.02596647 -0.66546913 -0.299574847
u1  0.23364579  0.48764812  0.404052101
u1  0.77760355  0.55757852 -0.018605954
u1  0.49396550  0.42381019  0.002810148
u1 -0.61205087  0.02689782  0.320583260

$rejectingplots
[1]  1  4  6 14 15

$pvals
 [1] 0.0000 0.1438 0.1045 0.0000 0.8663 0.0000 0.7538 0.7103
 [9] 0.3159 0.7800 0.3119 0.2859 0.6582 0.0000 0.0000
```



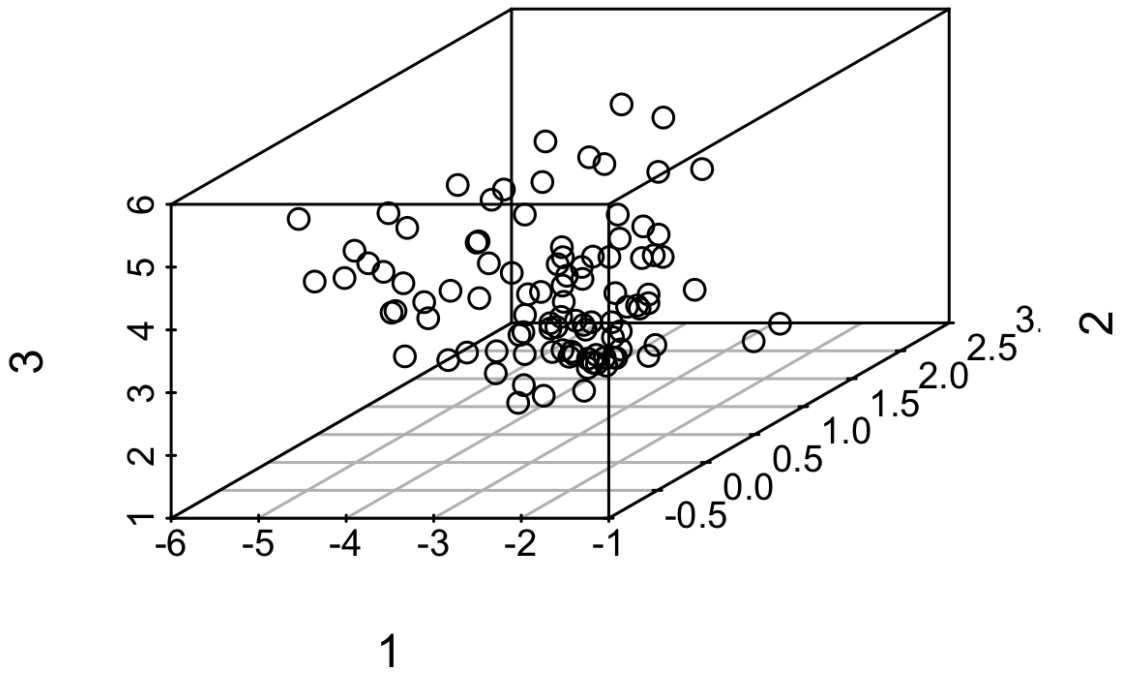
#Orthogonalize vectors

```
> orthmat<-orthogonalize(mvntest$Int[1:3,])
          [,1]       [,2]       [,3]      [,4]       [,5]
[1,] 0.2291419 -0.4733499 -0.2361252 0.3464874 -0.1198194
           [,6]       [,7]       [,8]
[1,] -0.02596647 -0.6654691 -0.2995748
```

```
> expl<-explore(orthmat,testmat3,T,F,F)
```

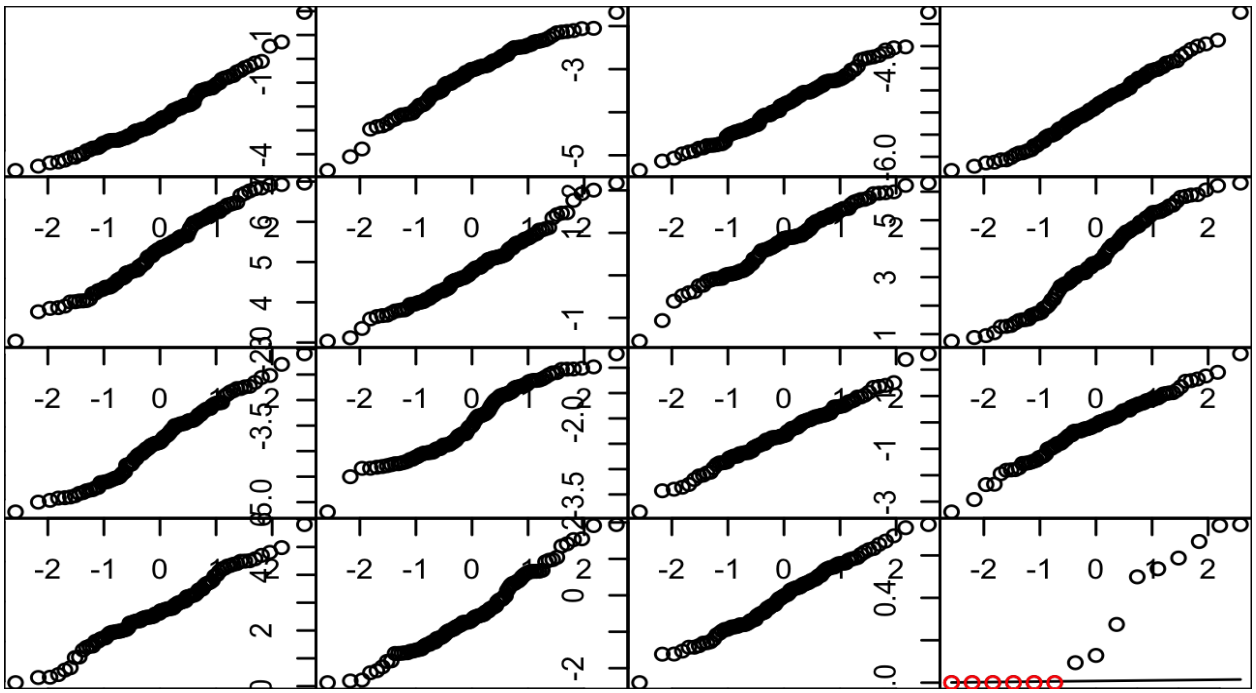



```
> expl<-explore(orthmat,testmat3,F,T,F,c(1,2,3))
```

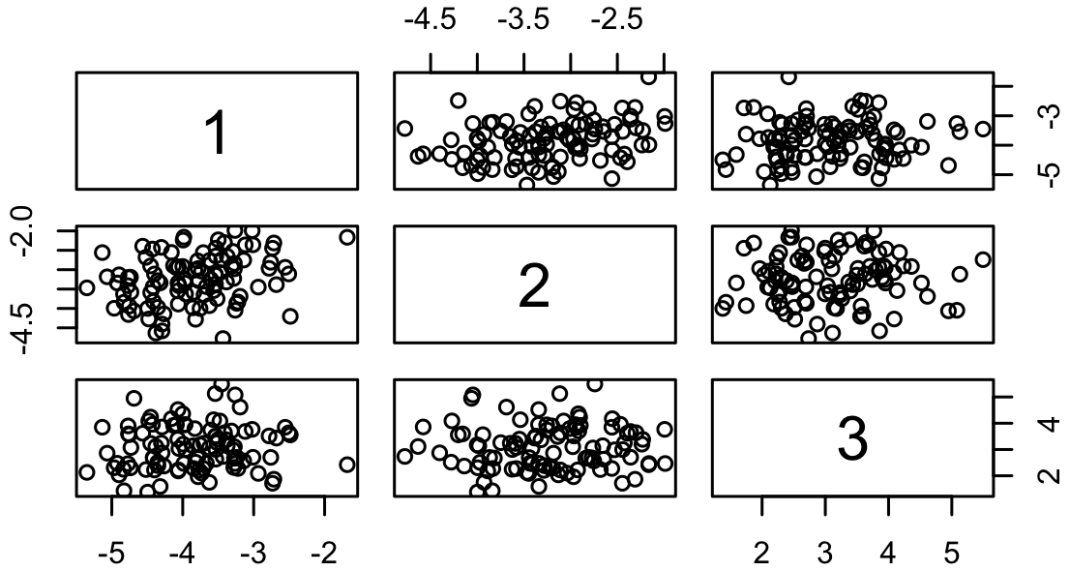


Suite1 Group 4:

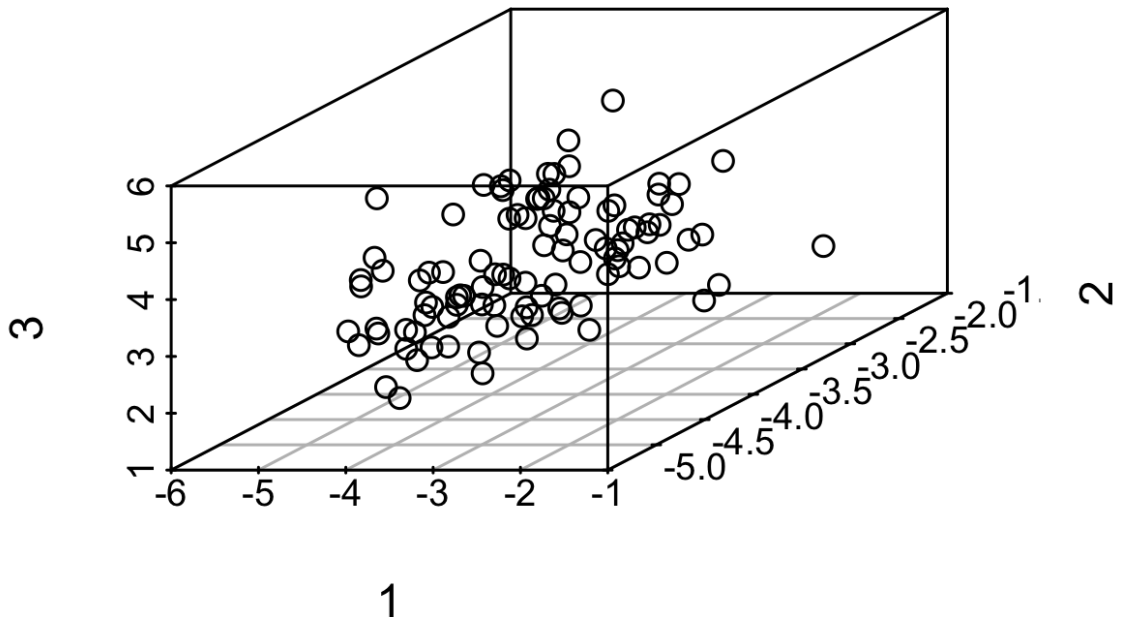
```
> mvntest<-Multivariate.normal.test(testmat4, ntest = 15, Q = 0.05)
$Interesting.directions
         [,1]       [,2]       [,3]        [,4]       [,5]
u1 -0.11277399  0.2139449 -0.64028112 -0.199587001 -0.25466792
u1  0.05500256 -0.4197648 -0.63265144  0.009740219  0.05066097
u1  0.15124795  0.2148695 -0.21814316 -0.264519733  0.15380178
u1 -0.45750840 -0.5526772  0.42988193 -0.346986471 -0.38731369
u1  0.07629654  0.3894471  0.05073182 -0.572924644 -0.63579385
u1  0.52197228 -0.1614776 -0.03224238  0.389662125  0.19662727
         [,6]       [,7]       [,8]
u1 -0.5746191  0.07036482 -0.30285417
u1 -0.2183462 -0.23131099 -0.56274607
u1  0.5140900  0.58235329  0.43164700
u1 -0.1559367  0.05242089 -0.05440398
u1 -0.1479739 -0.28379090 -0.07089456
u1  0.5424113  0.24485348 -0.39467485

$rejectingplots
[1]  3  4  6 10 11 13

$pvals
 [1] 0.7455 0.6656 0.0000 0.0000 0.4990 0.0000 0.5368 0.2747
 [9] 0.0941 0.0000 0.0000 0.7415 0.0000 0.1284 0.5876
```



```
> orthmat<-orthogonalize(mvntest$Int[1:3,])
          [,1]      [,2]       [,3]      [,4]       [,5]
[1,] -0.112774 0.2139449 -0.6402811 -0.199587 -0.2546679
          [,6]       [,7]       [,8]
[1,] -0.5746191 0.07036482 -0.3028542
> expl<-explore(orthmat,testmat4,T,F,F)
```

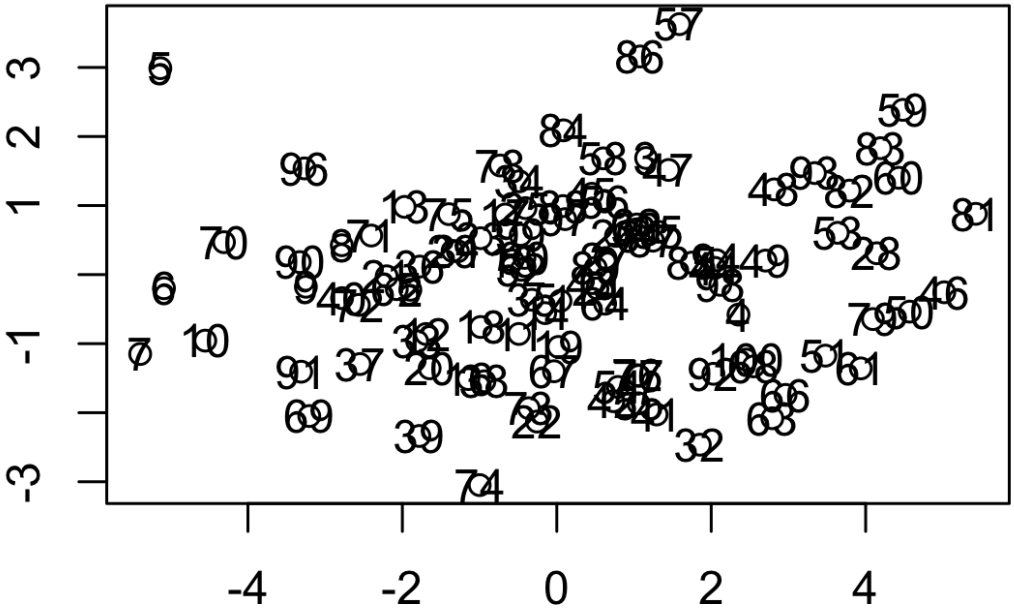



```
> expl<-explore(orthmat,testmat4,F,T,F,c(1,2,3))
```



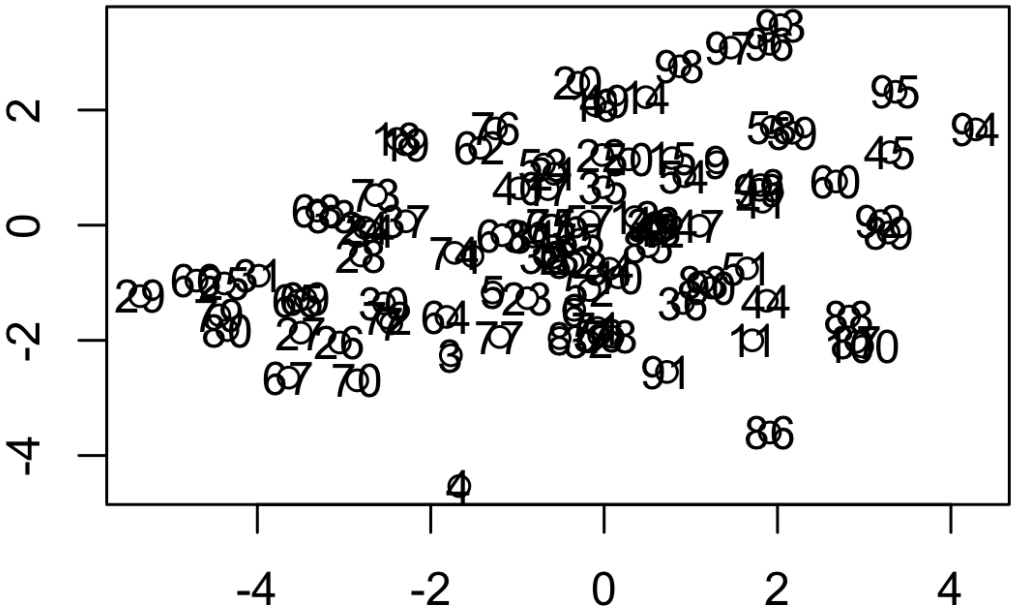
Suite 2 Group 1:

```
> asym.contamination.plot(testmat1,0,c(.2,.7),0.95,T,F,F)
```



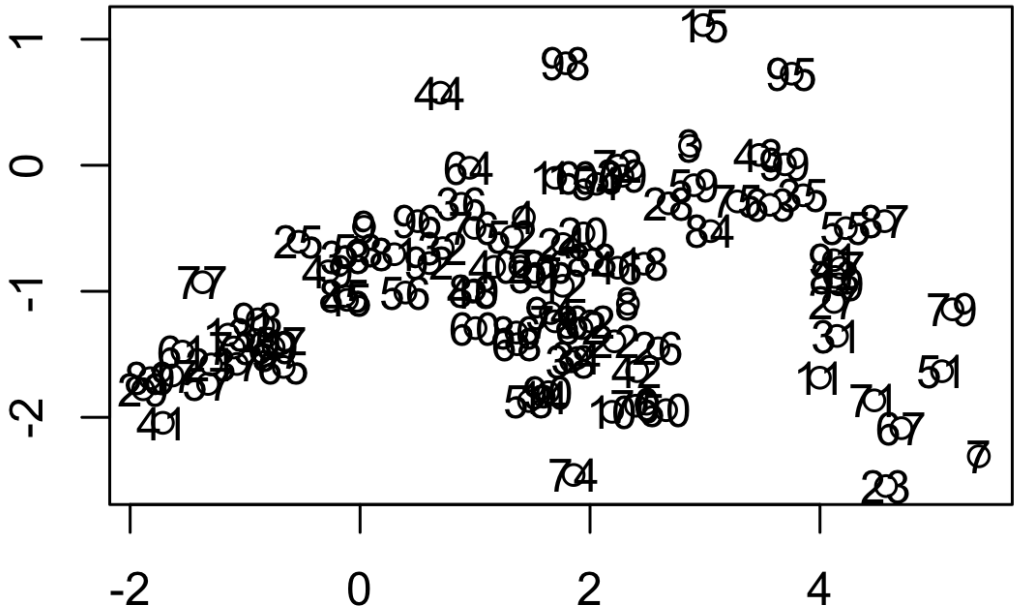
Suite 2 Group 2:

```
> asym.contamination.plot(testmat2,0,c(.2,.7),0.95,T,F,F)
```



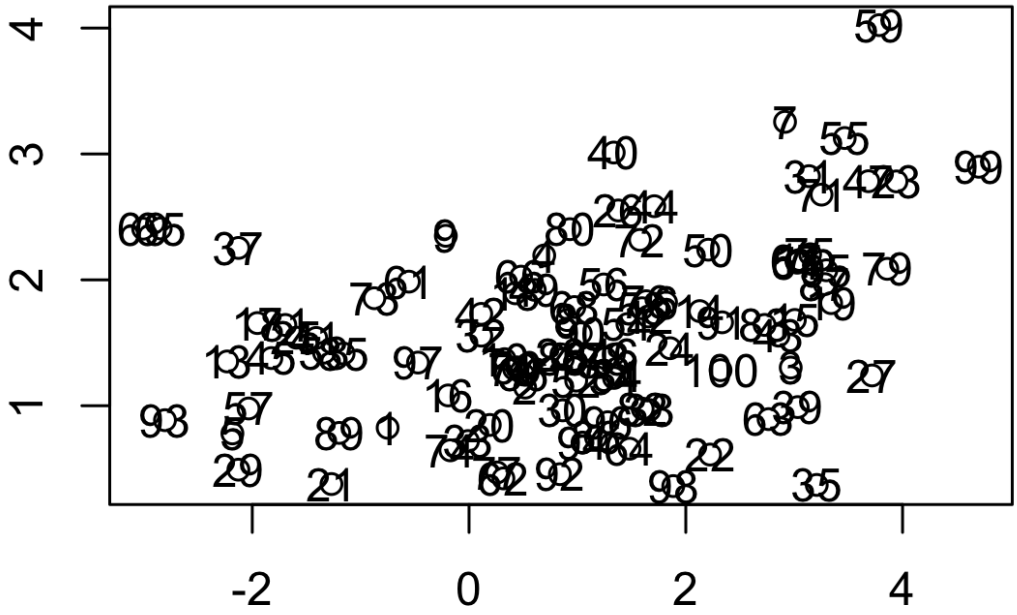
Suite 2 Group 3:

```
> asym.contamination.plot(testmat3,0,c(.2,.7),0.95,T,F,F)
```



Suite 2 Group 4:

```
> asym.contamination.plot(testmat4,0,c(.2,.7),0.95,T,F,F)
```

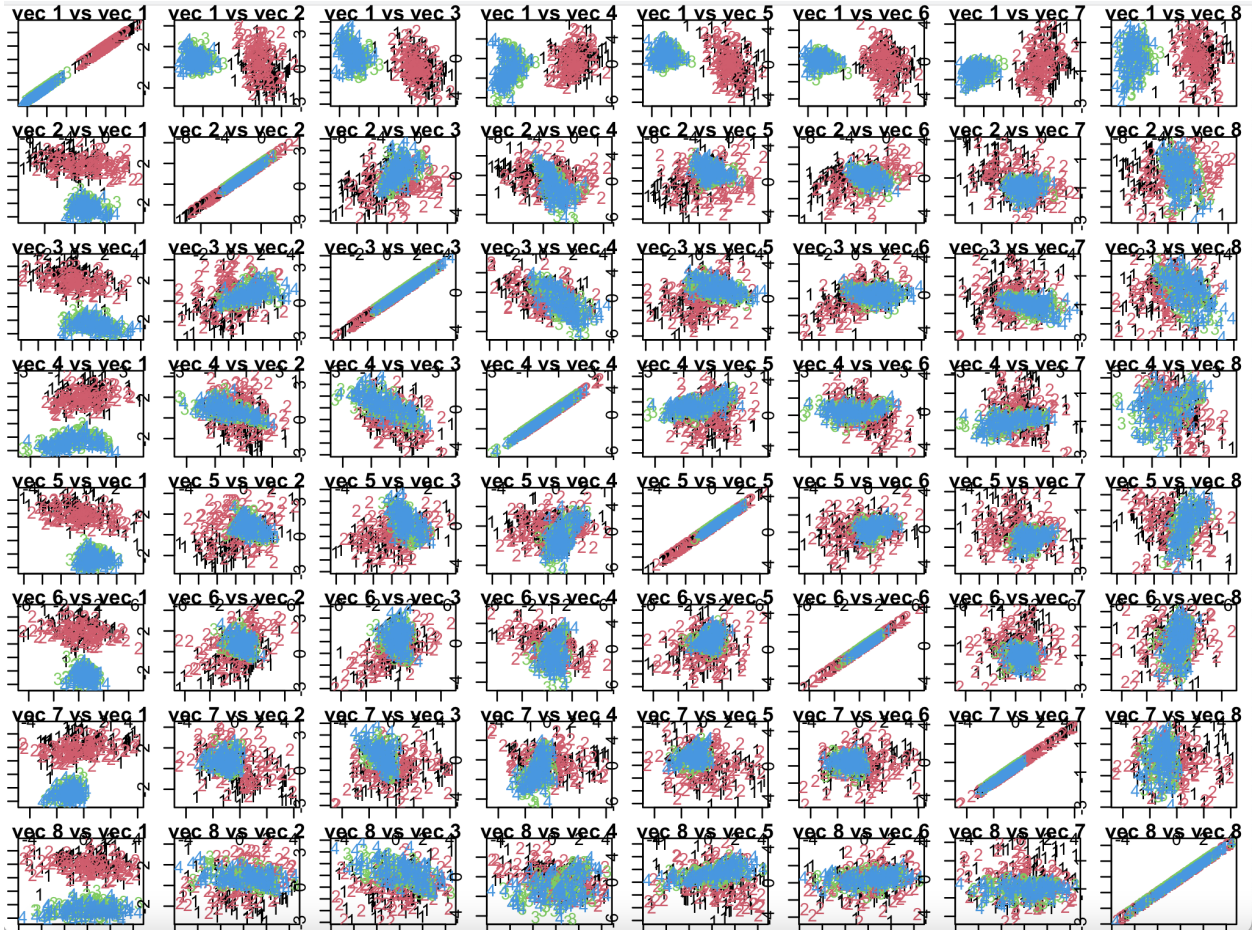


Suite 3 (Run on entire data set except factor columns have been removed):
#remove factor x1 and x2 columns including grouping col as id col

```
> Ptestmat = testmat[,-c(9:10)]
> quartz(width = 8, height = 6)
> par(mar=c(0.75,0.75,0.75,0.75))
```

#9 is the id col or grouping variable and numvec=8 is number of column indices starting at col1 that correspond to the numeric variables in the dataset that will have cov computed.

> Perm.cov.test(Ptestmat,9,8)



iii. Based on results of 7aii apply appropriate packages of LDA, QDA, tree, rpart, and randomForest to discriminate between the four groups, Where appropriate make an ROC plot to decide between

#change testmats to include the grouping column and remove factors!

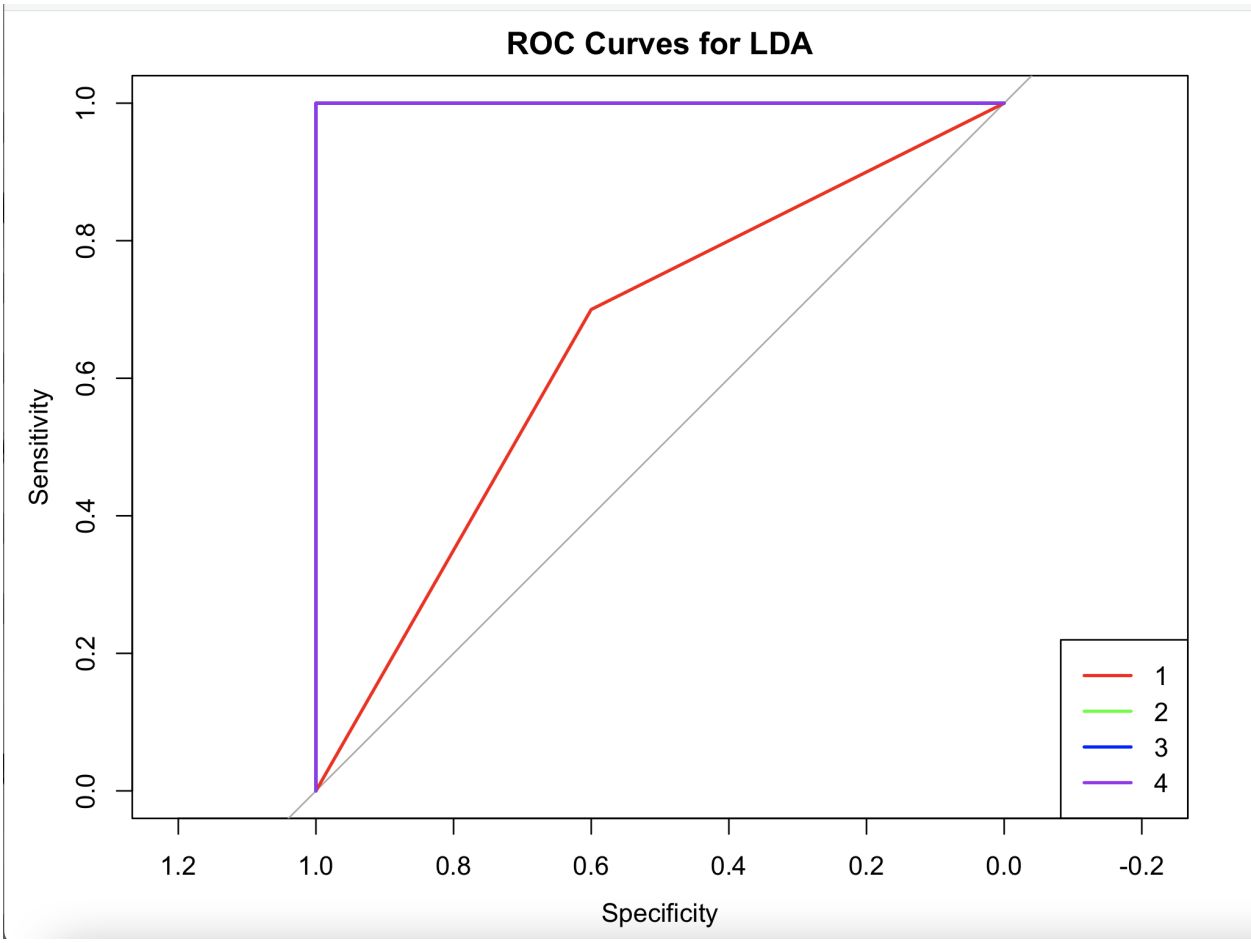
```
> testmat1 = as.matrix(subset(testmat[,-c(9:10)], testmat[,11] == 1))
> testmat2 = as.matrix(subset(testmat[,-c(9:10)], testmat[,11] == 2))
> testmat3 = as.matrix(subset(testmat[,-c(9:10)], testmat[,11] == 3))
> testmat4 = as.matrix(subset(testmat[,-c(9:10)], testmat[,11] == 4))
```

#splitting each group randomly into 80% training and 20% testing for specific group (using same random sample train index for every group)

```
> set.seed(123)
> train_index <- sample(1:100, 0.8 * 100)
```

```
> train1 <- testmat1[train_index, ]
> test1 <- testmat1[-train_index, ]
> train2 <- testmat2[train_index, ]
> test2 <- testmat2[-train_index, ]
> train3 <- testmat3[train_index, ]
> test3 <- testmat3[-train_index, ]
> train4 <- testmat4[train_index, ]
> test4 <- testmat4[-train_index, ]

#combine the trains and tests into one train and test data matrix
>train = rbind(train1, train2, train3, train4)
>test = rbind(test1, test2, test3, test4)
> colnames(train)[9] <- "Group"
> colnames(test)[9] <- "Group"

# Create a factor variable for species
> train = data.frame(train)
> test = data.frame(test)
> train$Group <- as.factor(train$Group)
> test$Group <- as.factor(test$Group)

# load libraries to run  LDA, QDA, tree, rpart, and randomForest
library(MASS)
library(rpart)
library(randomForest)
library(pROC)

# Linear Discriminant Analysis (LDA)
> lda_model <- lda(Group ~ ., data = train)
> lda_pred <- as.numeric(predict(lda_model, newdata = test)$class)

#need to run multiclass.roc because there are 4 classes within $Group
> lda_roc = multiclass.roc(test$Group, lda_pred)

#find AUC
> auc(lda_roc)
Multi-class area under the curve: 0.8625

#plots ROC curves for each individual classes
> rs <- lda_roc[['rocs']]
> quartz(width =8,height=6)
> plot(rs[[1]], type = "n", main = "ROC Curves for LDA")
> colors <- c("red", "green", "blue", "purple")
> for (i in 1:length(rs)) { lines(rs[[i]], col = colors[i], lwd = 2)}
```

> legend("bottomright", legend = levels(test$Group), col = colors, lty = 1, lwd = 2)



 # Quadratic Discriminant Analysis (QDA)

>qda_model <- qda(Group ~ ., data = train)

>qda_pred <- as.numeric(predict(qda_model, newdata = test)$class)

#need to run multiclass.roc because there are 4 classes within $Group

> qda_roc = multiclass.roc(test$Group, qda_pred)

#find AUC

> auc(qda_roc)

Multi-class area under the curve: 0.8458

#plots ROC curves for each individual classes

> rs <- qda_roc[['rocs']]

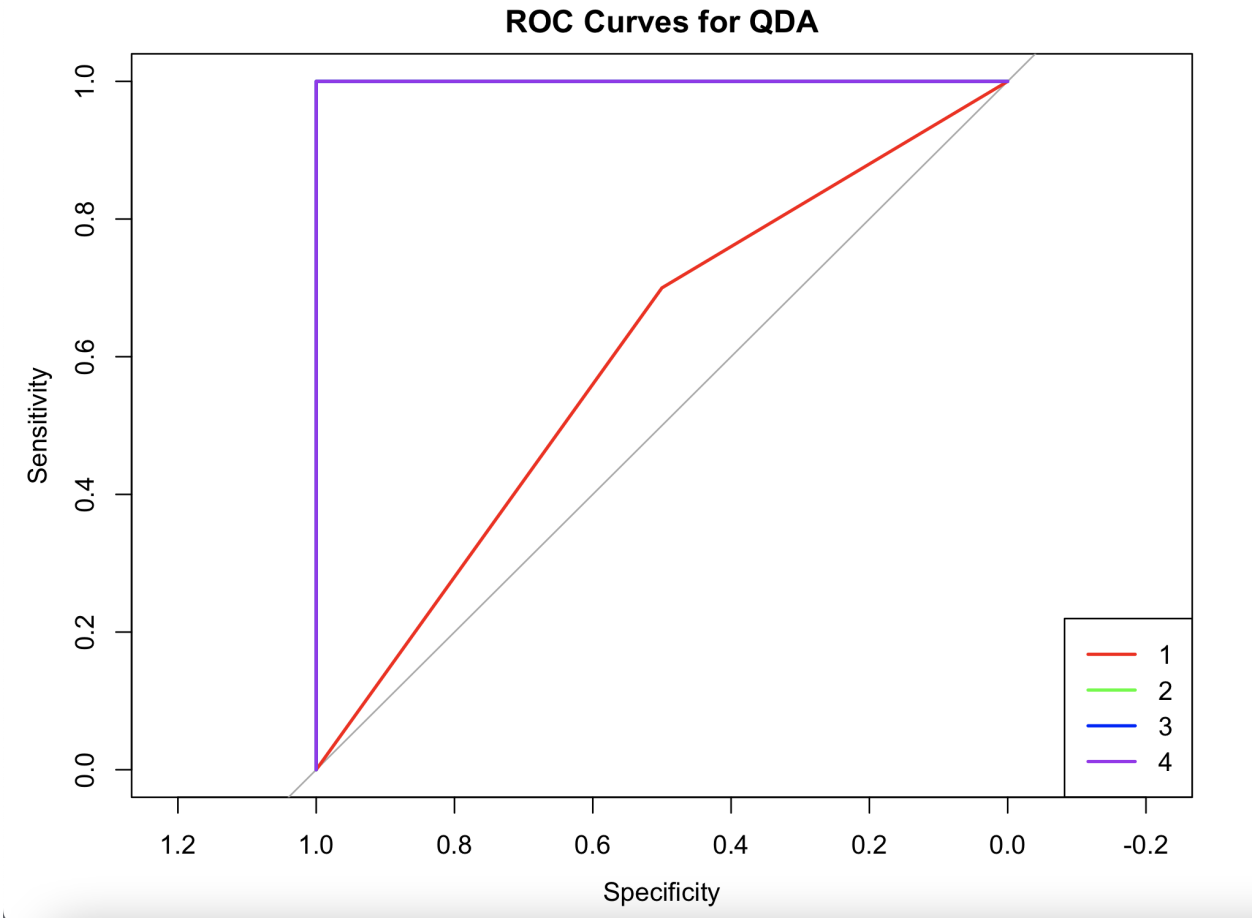
> quartz(width =8,height=6)

> plot(rs[[1]], type = "n", main = "ROC Curves for QDA")

> colors <- c("red", "green", "blue", "purple")

> for (i in 1:length(rs)) { lines(rs[[i]], col = colors[i], lwd = 2)}

```
> legend("bottomright", legend = levels(test$Group), col = colors, lty = 1, lwd = 2)
```



 # Decision Tree (rpart)

```
> tree_model <- rpart(Group ~ ., data = train, method = "class")
> tree_pred <- as.numeric(predict(tree_model, newdata = test, type = "class"))
```

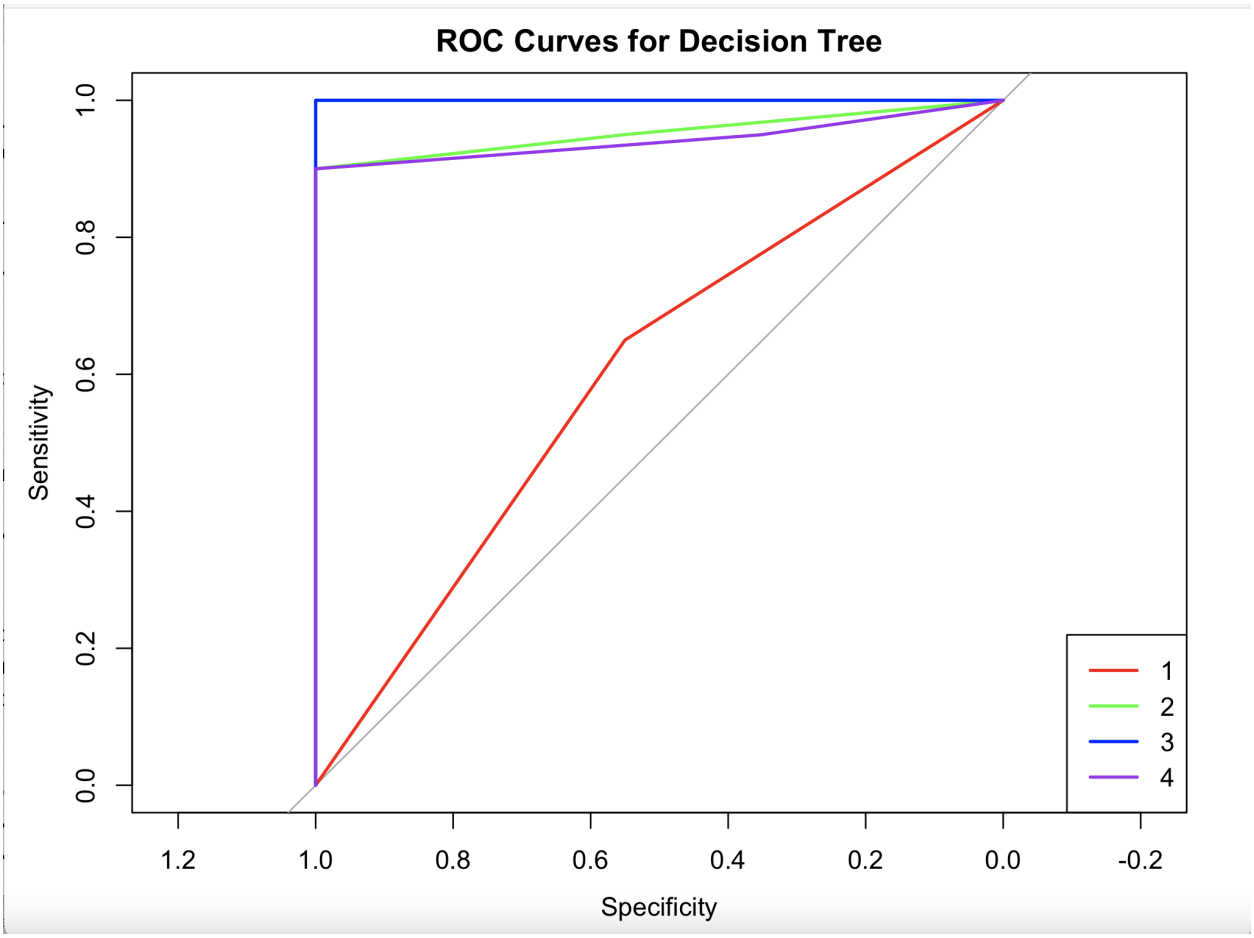
#need to run multiclass.roc because there are 4 classes within $Group

```
> tree_roc <- multiclass.roc(test$Group, tree_pred)
> auc(tree_roc)
```

Multi-class area under the curve: 0.8346

#plots ROC curves for each individual classes

```
> rs <- tree_roc[['rocs']]
> quartz(width =8,height=6)
> plot(rs[[1]], type = "n", main = "ROC Curves for Decision Tree")
> colors <- c("red", "green", "blue", "purple")
> for (i in 1:length(rs)) { lines(rs[[i]], col = colors[i], lwd = 2) }
> legend("bottomright", legend = levels(test$Group), col = colors, lty = 1, lwd = 2)
```



 # Random Forest

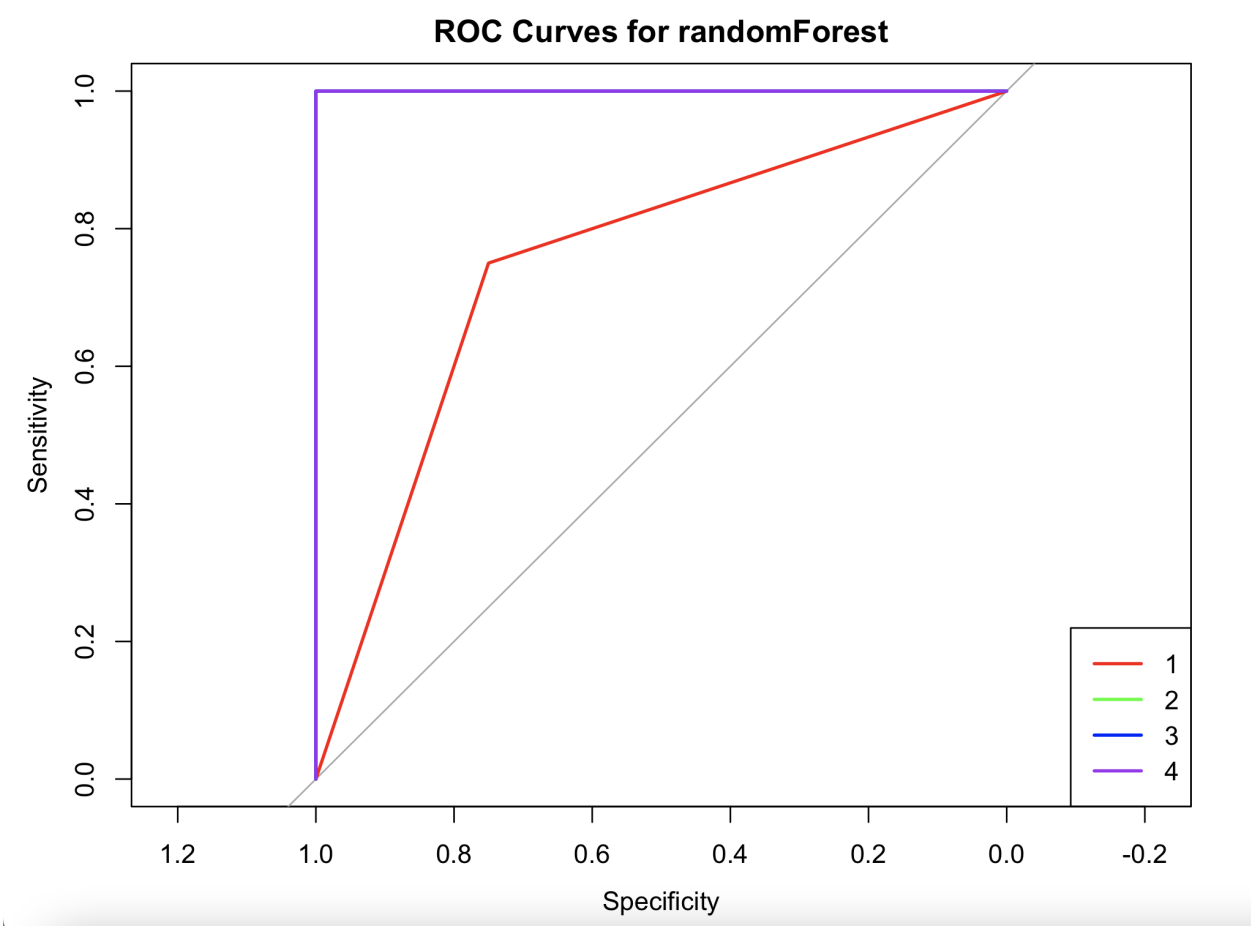
```
> rf_model <- randomForest(Group ~ ., data = train, importance = TRUE)
> rf_pred <- as.numeric(predict(rf_model, newdata = test))
```

#need to run multiclass.roc because there are 4 classes within $Group

```
> rf_roc <- multiclass.roc(test$Group, rf_pred)
> auc(rf_roc)
Multi-class area under the curve: 0.8792
```

#plots ROC curves for each individual classes

```
> rs <- rf_roc[['rocs']]
> quartz(width =8,height=6)
> plot(rs[[1]], type = "n", main = "ROC Curves for randomForest")
> colors <- c("red", "green", "blue", "purple")
> for (i in 1:length(rs)) {
+     lines(rs[[i]], col = colors[i], lwd = 2)
+ }
> legend("bottomright", legend = levels(test$Group), col = colors, lty = 1, lwd = 2)
```



***** EXPLANATION OF RESULTS *****

The randomForest() function provided the best classification model since it has the highest AUC at 0.8792. The AUC (area under the ROC curve) is a common metric used to evaluate the performance of binary and multiclass classification models since it is a measure of the model's ability to distinguish between the classes in the dataset, where a higher AUC value indicates a better model performance. randomForest likely performed better because it can capture non linear relationships better (than the other models), it is robust to outliers and it aggregates the predictions of multiple decision trees, which can help to reduce variance and increase model accuracy, while LDA and QDA make predictions based on a single model.

8. Cluster analysis (Class 11)

a. Suite 10: my.mvknorm, kmeans, hclus, cutree

i. What does each do

My.mvknorm: performs mixture modelling with multivariate normal distribution. A method to estimate the parameters of a statistical model where the data is assumed to come from a mixture of several normal distributions.

a. Input arguments: takes as input a matrix of data (x0), the number of components to be fitted (k), stopping threshold (thresh.stop) and scaling the data (do.scale).

b. The function initializes the means and covariances of the normal distributions in the mixture model
c. Outputs the estimated means (mu), covariances (sig), mixture weights (pivec), likelihood of the model , probability matrix of each data point belonging to each component, and the cluster assignments (z) for each data point.

Kmeans: (*unsupervised machine learning algorithm***)** clustering algorithm that determines K clusters based on euclidean distances. Clustering models aim to group data into distinct "clusters" or groups. It classifies objects in multiple clusters, such that objects within the same cluster are as similar as possible (high intra-class similarity). In contrast, objects from different clusters are as dissimilar as possible (low inter-class similarity). In k-means clustering, each cluster is represented by its centroid corresponding to the mean of points assigned to it.

Hclus: Hierarchical cluster analysis: Performs hierarchical clustering on a set of observations. Hierarchical clustering is a technique that groups similar objects together by constructing a hierarchy of clusters.
***hclust(d, method = "complete", members = NULL)*

a. Input Arguments: takes a matrix or a data frame of observations as input and produces a tree-based representation of the clustering structure.
b. The output of the hclust function can be plotted as a dendrogram using the plot function. The dendrogram shows the hierarchical structure of the clusters, where the height of each branch represents the distance between the clusters that are being merged.

Cutree: extracts clusters from a hierarchical clustering object created by the *hclust function*.
***cutree(tree, k = NULL, h = NULL)*

a. Input Arguments: give a hierarchical clustering object and a number of clusters; returns a vector of integers indicating the cluster membership for each observation in the original dataset.
    i. Tree: hierarchical clustering object; a tree as produced by *hclust()*
    ii. K: an integer scalar or vector with the desired number of groups
    iii. H: numeric scalar or vector with heights where the tree should be cut
b. returns a vector with group memberships if k or h are scalar,
    i. Else: returns a matrix with group memberships where each column corresponds to the elements of k or h, respectively.