1. Coding with R

2. The current working directory is displayed by the RStudio IDE within the title region of the Console pane. You can also check your current working directory by running the command getwd() in the console
3.
4. Open the Excel file containing your data: select and copy the data (ctrl + c)
5. Remember to import separately qualitative data Y and qualitative data X
6.
7. IMPORTANT: Rename LABELS of VARIABLES using the code:
8. **V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21 V22 V23 V24**
9.
10. **Rename also the CATEGORIES using the CODE a1, a2, a3 ect**
11. **if you have more than one Qualitative Column rename Using a1, a2, a3 next category b1, b2, b3, ect……**

12. #Type the R code below to import the copied data from the clipboard into R and store the data in a data frame (my_data):
13.
14. X <- read.table(file = "clipboard", sep = "\t", header=TRUE)
15. X<- read.csv("filename.csv", row.names = 1, header= TRUE)

Change column name

colnames(XY)[1] <- "A"

16. #rConvert first column in data.frame to row index [duplicate]
17.
18. > X <- data.frame(X[,-1], row.names = X$Samples)

19. Remove object from Project

20. rm(X)

21. #Select specific row or Column

22. #only columns
23. X1<-data.frame(X[c(1,3:5,7)])
24. #for rows and specific columns
25. X<-data.frame(X[c(1,3:5,7),(3:7)])

26. #Missing data

27. #In R, missing values are represented by the symbol NA (not available). Impossible values (e.g., dividing by zero) are represented by the symbol NaN (not a number).
28. https://cran.r-project.org/web/packages/finalfit/vignettes/missing.html
29.

30. SORT and MATCH and JOIN DATA

31. #Methods to sort a dataframe:
32. #order() function (increasing and decreasing order)
33. #arrange() function from dplyr package
34. #setorder() function from data.table package
35.
36.
37. # sort by colname
38. Xsort <- data.table::setorder(X, namecol)
39. df <-data_frame[order(data_frame$c1),]
40. # Sort by c3 and c4
41. df <-data_frame[order(data_frame$c3, data_frame$c4),]
42. # Sort by c3(descending) and c4(acending)
43. df <-data_frame[order(-data_frame$c3, data_frame$c4),]
44.
45.
46.
47. #match column name
48. x<-colnames(X)
49. y<-colnames(Y)
50. myx<-match(y, x)
51.
52.

53. # Reorder columns based on values in a particular row (df=data frame or your matrix X or XY).

54. x <- structure(list(aa = c(3L, 5L, 7L, 33L), bb = c(4L, 4L, 8L, 63L),
55. cc = c(5L, 3L, 6L, 55L)), .Names = c("aa", "bb", "cc"),
56. class = "data.frame", row.names = c("1", "2", "3", "100"))
57. x[,order(-x[nrow(x),])]

58.
59. ### Reorder the columns of the dataframe in Alphabetical order
60. df[,order(colnames(df))]
61. #### Reorder the columns of the dataframe in Alphabetical order
62. df[,order(colnames(df),decreasing = TRUE)]
63.
64. # sort df by row name (first row in example)
65. dfR<-df[,order(df[which(rownames(df) == '1'),], decreasing=TRUE)]
66.
67. #Reorder the columns of the dataframe in data type order
68.
69. #list of data type
70. classes = data.frame(sapply(df,class))
71. #transpose the data.frame
72. classes <-t(classes)
73. #bind rows
74. dfVar<-rbind(classes, df)
75. rownames(dfVar) <- NULL
76. # Reorder the columns of the dataframe in data type order
77. dfVarO<-dfVar[, order(dfVar[which(rownames(dfVar) == '1'),])]
78.
79.
80.
81.
82. #Converting character matrix (chr, qualitative data) to numerical matrix for pls
83. XY<-data.matrix(C, rownames.force = NA)
84.
85. #Import data in R from excel going to file->import dataset->excel…….. Follow the instruction in the wizard window

86. #Analyzing Qualitative data Y on Excel using PIVOT command or with R following the instructions in the following websites:
87. https://uc-r.github.io/descriptives_categorical
88. https://cran.r-project.org/web/packages/vcdExtra/vignettes/vcd-tutorial.pdf

89. https://www.r-bloggers.com/2019/06/exploratory-data-analysis-with-categorical-data
90. # counts for gender categories
91. table(XY$Cat1)
92. # cross classication counts for gender by marital status
93. table(X$Cat1, X$Cat2)
94. #percentages of gender categories
95. table2 <- table(X$Cat1)
96. prop.table(table2)
97.
98.
99.
100. #Check the structure
101. str(X)
102.
103. #Do summary statistics of the quantitative data matrix X using
104. summary(X)
105.
106. #Do summary statistics with more parameters of the quantitative data matrix X using
107. #(remember always activate the library you dont have library -> install the package)
108. library(psych) #also HMiSC does describe so deactivate it
109. describe(X)
110.
111. #Do boxplot of the quantitative data matrix X using
112. https://www.r-graph-gallery.com/boxplot.html
113.
114. **#Do normalization using excel or with mdatools**
115. library(mdatools)
116. # autoscaling (mean centering and standardization)
117. XN = prep.autoscale(X, center = TRUE, scale = TRUE)
118.
119.
120. **(stripchart applied to boxplot!!) #All lines below together**
121. https://stackoverflow.com/questions/23675735/how-to-add-boxplots-to-scatterplot-with-jitter
122.
123. boxplot(XN)

124. stripchart(XN, method = "jitter", pch = 19, col = 4, vertical = TRUE,   add = TRUE)
125.
126. boxplot(X)
127. stripchart(X, method = "jitter", pch = 19, col = 4, vertical = TRUE,   add = TRUE)
128.
129. BoxData<- boxplot(X)
130.
131. **#DO analysis of outlierS**
132. https://www.r-bloggers.com/2020/01/how-to-remove-outliers-in-r/
133. https://www.r-bloggers.com/2021/09/how-to-remove-outliers-in-r-3/
134. https://statsandr.com/blog/outliers-detection-in-r/
135.
136. boxplot(X)$out
137. library(outliers)
138. outlier(X, logical = TRUE)
139.
140. #Merge matrices
141. Use command merge
142. https://www.rdocumentation.org/packages/Matrix.utils/versions/0.9.8/topics/merge.Matrix
143. X <- data.frame(X1, X2, X3, X4, X5, X6, X7, X8)
144. Xm <- as.matrix(X)
145.
146. #Transpose Matrix
147. t() function. As previously, mentioned, you can use t(YourMatrix) to get the transpose of your matrix “YourMatrix”.
148. **#ANOVA** using multiple variable (where V1, V2, V3, V4, V5, V6, V7 are the variable and Cat the groups vector, ALL in the same matrix)
149.
150. anX_multi<- aov(formula = cbind(V1, V2, V3, V4, V5, V6, V7) ~ Cat, data = XY)
151. summary(anX_multi)
152.
153. #ANOVA for matrix X using one variable (where Letters are the variables and Cat the groups vector, both in the same matrix XY that in this case is a mix of qualitative Y and quantitative X)
154.
155. anV1X <- aov(V1 ~ Cat, data = XY)
156. summary(anV1X)

```
157.  TukeyHSD(anV1X, which = "Cat")
158.
159.  anV2X <- aov(V2 ~ Cat, data = XY)
160.  summary(anV2X)
161.  TukeyHSD(anV2X, which = "Cat")
162.
163.  anV3X <- aov(V3 ~ Cat, data = XY)
164.  summary(anV3X)
165.  TukeyHSD(anV3X, which = "Cat")
166.
167.  anV4X <- aov(V4 ~ Cat, data = XY)
168.  summary(anV4X)
169.  TukeyHSD(anV4X, which = "Cat")
170.
171.  anV5X <- aov(V5 ~ Cat, data = XY)
172.  summary(anV5X)
173.  TukeyHSD(anV5X, which = "Cat")
174.
175.  anV6X <- aov(V6 ~ Cat, data = XY)
176.  summary(anV6X)
177.  TukeyHSD(anV6X, which = "Cat")
178.
179.  anV7X <- aov(V7 ~ Cat, data = XY)
180.  summary(anV7X)
181.  TukeyHSD(anV7X, which = "Cat")
182.
183.
```

184. For visualizing boxplot with categorical data (this example is for the matrix “X”)

```
185.  library("ggpubr")
186.
187.  x <- which(names(XY) == "Cat") # name of grouping variable
188.  y <- which(names(XY) == "V1" # names of variables to test
189.  | names(XY) == "V2"
190.  | names(XY) == "V3"
191.  | names(XY) == "V4"
192.  | names(XY) == "V5"
193.  | names(XY) == "V6"
194.  | names(XY) == "V7")
195.  method1 <- "anova" # one of "anova" or "kruskal.test"
196.  method2 <- "t.test" # one of "wilcox.test" or "t.test"
197.  my_comparisons <- list(c("a1", "a2"), c("a1", "a3"), c("a1", "a4"), c("a2",
    "a3"), c("a2", "a4"), c("a3", "a4")) # comparisons for post-hoc tests
```

198.
199. To produce graph with p-value

```
200. for (i in y) {
201.   for (j in x) {
202.     p <- ggboxplot(XY,
203.       x = colnames(XY[j]), y = colnames(XY[i]),
204.       color = colnames(XY[j]),
205.       legend = "none",
206.       palette = "npg",
207.       add = "jitter"
208.     )
209.     print(
210.       p + stat_compare_means(aes(label = paste0(..method.., ", p-value = ",
   ..p.format..)),
211.         method = method1, label.y = max(XY[, i], na.rm = TRUE)
212.       )
213.       + stat_compare_means(comparisons = my_comparisons, method =
   method2, label = "p.format") # remove if p-value of ANOVA or Kruskal-Wallis
   test >= alpha
214.     )
215.   }
216. }
```

217.
218.
219.
220.
221.
222. ANOVA follow the example on http://www.sthda.com/english/wiki/two-way-anova-test-in-r
223. ANOVA follow the example https://statsandr.com/blog/how-to-do-a-t-test-or-anova-for-many-variables-at-once-in-r-and-communicate-the-results-in-a-better-way/
224.

225. If you have problems with plots digit that

226. par(mar=c(1,1,1,1))
227. **Or work with the zoom command from menu**

228. BIVARIATE ANALYSIS -----Do correlation plot of the quantitative data matrix X using

229.

230. **#Do normalization using excel or with mdatools**

231. library(mdatools)

232. # autoscaling (mean centering and standardization)

233. XN = prep.autoscale(X, center = TRUE, scale = TRUE)

234.

235. plot(X)

236.

237. plot(XN)

238.

239. Do correlation plot with graph and r values of the quantitative data matrix X using

240. (remember always activate the library you need if you dont have library install the package)

241. library("PerformanceAnalytics")

242. chart.Correlation(X, histogram=TRUE, pch=19)

243.

244. Store correlation and p value matrix.. (Hmisc package and read the output as matrix.)

245. library(Hmisc)

246. corX<- rcorr(as.matrix(X),type="pearson")

247.

248. Export correlation matrix to manage by excel using

249. write.csv(corX$P, file = "corX.csv")

250. MULTIVARIATE ANALYSIS with mdatools library

251. DO PCA

252. library(mdatools)

253. XPCA = pca(X, 7, scale = TRUE, info = "X PCA model")

254. summary(XPCA)

255. plot(XPCA, show.labels = TRUE)

256. plotBiplot(XPCA, show.labels = TRUE)

257.

258.

259. for making Scatterplot Matrices

260. pairs(XPCA$calres$scores, bg = c("red", "green3", "blue", "black")[unclass(XY$Cat1)])
261. #with group colours
262. l <- length(unique(XY$A))
263. pairs(XPCA$calres$scores, col = hcl.colors(l, "Temps")[XY$A])
264.
265.

266. CHECKING DISTANCES

267. **#orthogonal distance Q** (The distance shows how well the object is fitted by the PCA model and allows to detect objects that do not follow a common trend, captured by the PCs) and **Hotelling T2 distance or a score distance**. (The T2 distance allows to see extreme objects — which are far from the origin)
268. c = categorize(XPCA)
269. plotResiduals(XPCA, show.labels = TRUE, cgroup = c)
270.

271. # change both levels to 1% (default are 5%)

272. XPCA1 = setDistanceLimits(XPCA, alpha = 0.01, gamma = 0.01)
273. plotResiduals(XPCA1, show.labels = TRUE, cgroup = c)
274.
275.
276.
277. #Follow this example to do PCA 2D (3D doesn't work) with cluster analysis in very quirky way
278. https://planspace.org/2013/02/03/pca-3d-visualization-and-clustering-in-r
279.
280. The package pca3d quickly generates 2D and 3D graphics of PCA.
281. https://cran.r-project.org/web/packages/pca3d/vignettes/pca3d.pdf
282.

283. MAKING 3D GRAPH

284. https://r-graph-gallery.com/index.html
285. https://r-graph-gallery.com/3d.html
286.
287.
288. #3D graph
289.
290. ```{r}
291.
292. #library
293. library(rgl)

~~~

This is to output a rgl plot in a rmarkdown document.
setupKnitr()

Data: the iris data is provided by R
data <- data.frame(XPCArescal$scores[,1:3])

Add a new column with color
mycolors <- c('royalblue1', 'darkcyan', 'oldlace')
data$color <- mycolors[as.numeric(XY$A)]

Plot
plot3d(
  x=data$Comp.1, y=data$Comp.2, z=data$Comp.3,
  col = data$color,
  type = 's',
  radius = .1,
  xlab="PC1", ylab="PC2", zlab="PC3", row)

To display in an R Markdown document:
rglwidget()

To save to a file:
htmlwidgets::saveWidget(rglwidget(width = 520, height = 520),
                        file = "HtmlWidget/3dscatter.html",
                        libdir = "libs",
                        selfcontained = FALSE
)


```



#3d surface plot with R and plotly
```{r}

# Library
library(plotly)

# Data: volcano is provided by plotly data have to be as.matrix
volc<-volcano
data2<-as.matrix(X [,5:9])
~~~

~~~
# Plot
p <- plot_ly(z =data2, type = "surface")
p

# save the widget
# library(htmlwidgets)
# saveWidget(p, file=paste0( getwd(), "/HtmlWidget/3dSurface.html"))

```


#animated

```{r}
library(rgl)
library(magick)

# Let's use the iris dataset
# iris

# This is ugly
colors <- c("royalblue1", "darkcyan", "oldlace")
iris$color <- colors[ as.numeric( as.factor(iris$Species) ) ]

# Static chart
plot3d( iris[,1], iris[,2], iris[,3], col = iris$color, type = "s", radius = .2 )

# We can indicate the axis and the rotation velocity
play3d( spin3d( axis = c(0, 0, 1), rpm = 20), duration = 10 )

# Save like gif
movie3d(
  movie="3dAnimatedScatterplot",
  spin3d( axis = c(0, 0, 1), rpm = 7),
  duration = 10,
  dir = "~/Desktop",
  type = "gif",
  clean = TRUE
)



```
~~~

382.
383.

384. Install package pca3d then

385. library(pca3d)
386. pcaX <- prcomp(X,, scale.=TRUE)
387.
388. Coloring samples based on categories remember to use Y matrix for categories
389. XCat <- factor(XY[,1])
390.
391. pca3d(XPCA$loadings, show.labels = TRUE)
392. pca3d(XPCA$calres$scores, show.labels = TRUE)
393.
394.
395. #Go to folder where project is working (where you saved the project and see the picture)
396. The next coding is to see ellipses
397. pca3d(pcaX, group=XCat, show.labels = TRUE, show.ellipses=TRUE, ellipse.ci=0.75, show.plane=FALSE)
398.
399. snapshotPCA3d(file="ellipses.png")
400.

401. CLUSTER ANALYSIS

402. https://www.r-bloggers.com/2021/04/cluster-analysis-in-r
403. or
404. http://www.sthda.com/english/articles/25-clusteranalysis-in-r-practical-guide/
405.
406.
407. Normalization is very important in cluster analysis, sometimes we have variables in different scales, need to normalized based on scale function before clustering the data sets.
408. Normalization is mandatory for cluster analysis.
409.
410. #Hierarchical agglomerative clustering (XN was normalized using mdatools or excel)
411.
412. disXN = dist(XN)
413. hcXN = hclust(disXN)
414.
415. plot(hcXN)
416.

417. Look at all elements in the object X.hclust using
418. hcXN$order
419.
420. #Cluster membership
421. member = cutree(hcXN,4)
422. table(member)
423.
424. #HeatMAP
425. hm<- data.matrix(XN)
426. heatmap(hm)
427. ht<- heatmap(hm)
428.
429.

430. PLS-DA

431.
432. https://mdatools.com/docs/index.html
433.
434. calibrate the models:
435. library(mdatools)
436. daX = plsda(X, XY$Cat, ncomp.selcrit = "min", scale = TRUE, cv = 1)
437. daX = plsda(X, XY$Cat, ncomp.selcrit = "min", ncomp = 10, scale = TRUE, cv = list('ven', nseg = 5))
438.
439. Check Number of Components:
440. daX$ncomp
441. plot(daX, ncomp = 7)
442.
443.
444. Check Number of Components that gives the minimus misclassification:
445. summary(daX, ncomp = 7)
446.
447. summary(daX$calres, ncomp = 7)
448.
449. plot(daX$calres, ncomp = 7)
450.
451. plotMisclassified(daX$calres, ncomp = 7)
452.
453. summary(daX$cvres, ncomp = 7)

454.
455. plot(daX$cvres, ncomp = 7)
456.
457. plotMisclassified(daX$cvres, ncomp = 7)
458.
459.
460. **In all Multivariate test you calibrate and validate …. using PLSDA with have to show two set** looking at the summary for each of the mode (Cal and Val).
461. As you can see, summary for multi class PLS-DA simply shows one set of results for each class. The performance statistics include explained X and Y variance (cumulative), values for confusion matrix (true positives, false positives, true negatives, false negatives) as well as specificity, sensitivity and accuracy values.
462. summary(daX, ncomp = 4)
463.
464. #you can also get a confusion matrix for particular result.
465. getConfusionMatrix(daX$res$cal, ncomp = 4)
466. getConfusionMatrix(daX$res$cv, ncomp = 4)
467.
468. #you can show how sensitivity, specificity and total amount of misclassified samples depending on number of components by using corresponding plots.
469.
470. par(mfrow = c(3, 2))
471. plotMisclassified(daX, ncomp = 4)
472. plotSensitivity(daX, ncomp = 4)
473. plotSpecificity(daX, ncomp = 4)
474.
475. #HAving details of the samples classified and misclassified in CAL or CV
476.
477. print(daX$res$cal$misclassified)
478. print(daX$res$cal$y.pred)
479.
480. print(daX$res$cv$misclassified)
481. print(daX$res$cv$y.pred)
482.
483. #SVM
484.
485. library(e1071)

```
library(rpart)
data(Glass, package="mlbench")
 ## split data into a train and test set
index <- 1:nrow(Glass)
testindex <- sample(index, trunc(length(index)/3))
testset <- Glass[testindex,]
trainset <- Glass[-testindex,]

# svm
svm.model <- svm(Type ~ ., data = trainset, cost = 100, gamma = 1)
svm.pred <- predict(svm.model, testset[,-10])

# rpart
rpart.model <- rpart(Type ~ ., data = trainset)
rpart.pred <- predict(rpart.model, testset[,-10], type = "class")

## compute svm confusion matrix
table(pred = svm.pred, true = testset[,10])

## compute rpart confusion matrix
table(pred = rpart.pred, true = testset[,10])
```

#K-fold cross-validation nested PLS-DA

```

#libraries

library(mdatools)
library(psych)
library(rms)
library(ggpubr)
library(PerformanceAnalytics)
library(Hmisc)
library(pca3d)
library(rpart)
library(e1071)

#split dataset in training and test

index <- 1:nrow(h144N)
testindex <- sample(index, trunc(length(index)/3))
```

```
testset1<- h144N[testindex,]
trainset1<- h144N[-testindex,]
index <- 1:nrow(h144N)
testindex <- sample(index, trunc(length(index)/3))
testset2<- h144N[testindex,]
trainset2<- h144N[-testindex,]
index <- 1:nrow(h144N)
testindex <- sample(index, trunc(length(index)/3))
testset3<- h144N[testindex,]
trainset3<- h144N[-testindex,]
index <- 1:nrow(h144N)
testindex <- sample(index, trunc(length(index)/3))
testset4<- h144N[testindex,]
trainset4<- h144N[-testindex,]
index <- 1:nrow(h144N)
testindex <- sample(index, trunc(length(index)/3))
testset5<- h144N[testindex,]
trainset5<- h144N[-testindex,]
index <- 1:nrow(h144N)
testindex <- sample(index, trunc(length(index)/3))
testset6<- h144N[testindex,]
trainset6<- h144N[-testindex,]
index <- 1:nrow(h144N)
testindex <- sample(index, trunc(length(index)/3))
testset7<- h144N[testindex,]
trainset7<- h144N[-testindex,]
index <- 1:nrow(h144N)
testindex <- sample(index, trunc(length(index)/3))
testset8<- h144N[testindex,]
trainset8<- h144N[-testindex,]
index <- 1:nrow(h144N)
testindex <- sample(index, trunc(length(index)/3))
testset9<- h144N[testindex,]
trainset9<- h144N[-testindex,]
index <- 1:nrow(h144N)
testindex <- sample(index, trunc(length(index)/3))
testset10<- h144N[testindex,]
trainset10<- h144N[-testindex,]

#PLSDA model on training

daT1 = plsda(trainset1[,-1], trainset1[,1],  scale = TRUE, cv = list("rand",nseg=4,nrep=10))
```

```
daT2 = plsda(trainset2[,-1], trainset2[,1],  scale = TRUE, cv = list("rand",nseg=4,nrep=10))
daT3 = plsda(trainset3[,-1], trainset3[,1],  scale = TRUE, cv = list("rand",nseg=4,nrep=10))
daT4 = plsda(trainset4[,-1], trainset4[,1],  scale = TRUE, cv = list("rand",nseg=4,nrep=10))
daT5 = plsda(trainset5[,-1], trainset5[,1],  scale = TRUE, cv = list("rand",nseg=4,nrep=10))
daT6 = plsda(trainset6[,-1], trainset6[,1],  scale = TRUE, cv = list("rand",nseg=4,nrep=10))
daT7 = plsda(trainset7[,-1], trainset7[,1],  scale = TRUE, cv = list("rand",nseg=4,nrep=10))
daT8 = plsda(trainset8[,-1], trainset8[,1],  scale = TRUE, cv = list("rand",nseg=4,nrep=10))
daT9 = plsda(trainset9[,-1], trainset9[,1],  scale = TRUE, cv = list("rand",nseg=4,nrep=10))
daT10 = plsda(trainset10[,-1], trainset10[,1],  scale = TRUE, cv = list("rand",nseg=4,nrep=10))

#prediction on test

T1= predict(daT1, testset1[,-1], testset1[,1])
T2= predict(daT2, testset2[,-1], testset1[,1])
T3= predict(daT3, testset3[,-1], testset1[,1])
T4= predict(daT4, testset4[,-1], testset1[,1])
T5= predict(daT5, testset5[,-1], testset1[,1])
T6= predict(daT6, testset6[,-1], testset1[,1])
T7= predict(daT7, testset7[,-1], testset1[,1])
T8= predict(daT8, testset8[,-1], testset1[,1])
T9= predict(daT9, testset9[,-1], testset1[,1])
T10= predict(daT10, testset10[,-1], testset1[,1])



```

592. DOE ------ experimental design

593. **https://cran.r-project.org/web/views/ExperimentalDesign.html**
594. **We start using**
595. **http://www.ru.ac.bd/stat/wp-content/uploads/sites/25/2019/03/502_07_00_Lawson_Design-and-Analysis-of-Experiments-with-R-2017.pdf**
596. **https://cran.r-project.org/web/packages/daewr/vignettes/daewr.pdf**

```
597.
598.    install.packages("daewr")
599.    install.packages("DoE.base")
600.    install.packages("FrF2")
601.    install.packages("rsm")
602.
603.    library(daewr)
604.    library(DoE.base)
605.    library(FrF2)
606.    library(rsm)
607.
608.
609.
```

610. FULL vs Factorial Factorial design

```
611.
612.    ff23 <- FrF2( 8, 3, randomize = FALSE)
613.    ff23
614.
615.
616.
617.
618.    ff23f <- FrF2( 4, 3, randomize = FALSE)
619.    ff23f
620.    y <- runif(4, 0, 1)
621.    aliases( lm( y~ (.)^4, data = ff23f))
622.
623.    design <- FrF2( 16, 5, generators = "ABCD", randomize = FALSE)
624.    design
625.    y <- runif(16, 0, 1)
626.    aliases( lm( y~ (.)^4, data = design))
627.
```

628. Plackett-Burman designs
629. it can be created easily using the FrF2 package. The
630. example below illustrates the use of the pb function in that package to create
631. the design with 11 factors using 12 runs.
632. library(FrF2)
633. pb(nruns = 12, randomize=FALSE)
634.
635.
636. Exercise full factorial
637. volt
638. modv <- lm(y ~ A*B*C, data=volt, contrast=list(A=contr.FrF2, B=contr.FrF2, C=contr.FrF2))
639. summary(modv)
640. par(mfrow = c(2,2))
641. IAPlot(modv)
642. IAPlot(modv, select = c(1,3))
643.
644. Exercise full factorial
645. chem
646. modf <- lm(y ~ A*B*C*D, data = chem)
647. summary(modf)
648. fullnormal(coef(modf)[-1],alpha=.025)
649.
650. Exercise full factorial
651. data(BoxM)
652. Gaptest(BoxM)
653.
654. Exercise fractional factorial
655.
656. soup <- FrF2(16, 5, generators = "ABCD", factor.names =list(Ports=c(1,3), Temp=c("Cool","Ambient"), MixTime=c(60,80),BatchWt=c(1500,2000), delay=c(7,1)), randomize = FALSE)
657. y <- c(1.13, 1.25, .97, 1.70, 1.47, 1.28, 1.18, .98, .78, 1.36, 1.85, .62, 1.09, 1.10, .76, 2.10)
658. soup <- add.response(soup , y)
659. mod1 <- lm(y ~ (.)^2, data = soup)
660. mod1
661. summary(mod1)
662.
663. soupc<-FrF2(16,5,generators="ABCD",randomize=FALSE)
664. soupc<-add.response(soupc, y)
665.

```r
soupc<-FrF2(16,5,generators="ABCD",randomize=FALSE)
soupc<-add.response(soupc, y)

modc<-lm(y~(.)^2, data=soupc)
LGB(coef(modc)[-1], rpt = FALSE)

#response surface experiment

#Exercise central composite designs 2 factors
#https://www.youtube.com/watch?v=5Zb-3gZlL1E
# Lez74 http://www.lithoguru.com/scientist/statistics/course.html

#-------------------------------------#
#--- Response Surface Modeling in R ---#
#-------------------------------------#

#First, install and load the "rsm" package

# install.packages("rsm")
library(rsm)

# Example generating a Box-Behnken design with three factors and two center points (no)
bbd(3, n0 = 2, coding = list(x1 ~ (Force - 20)/3, x2 ~ (Rate - 50)/10, x3 ~ Polish - 4))


# Example data set
data = ChemReact
plot(data)


# The data set was collected in two blocks.
# Block1 is a 2-level, two-factor factorial design with three repeated center points.
# Block 2 is the Central Composite Design (circomscribed) with 3 center points.
# The variables are Time = 85 +/- 5 and Temp = 175 +/- 5,
# Thus, the coded variables are  x1 = (Time-85)/5 and x2 = (Temp-175)/5
CR <- coded.data(ChemReact, x1 ~ (Time - 85)/5, x2 ~ (Temp - 175)/5)
CR[1:7,]

```

```
# Note: If the data are already coded, use as.coded.data() to convert to the proper coded data object

# Let's work as though the first block (full factorial) has been finished,
# and we'll fit a linear model, first order (FO), to it (Yield is the response)
CR.rsm1 <- rsm(Yield ~ FO(x1, x2), data = CR, subset = (Block == "B1"))
summary(CR.rsm1)

#The fit is not very good.  Let's include the interaction term (TWI) and update the model, or start over with a new model (these two lines do the same thing)
CR.rsm1.5 <- update(CR.rsm1, . ~ . + TWI(x1, x2))
CR.rsm1.5 <- rsm(Yield ~ FO(x1, x2)+TWI(x1, x2), data = CR, subset = (Block == "B1"))
summary(CR.rsm1.5)
#This is no better!  The reason is the strong quadratic response, with the peak near the center.

# Now let's assume the second block has been collected.  We use the SO (second order) function, which includes FO and TWI
CR.rsm2 <- rsm(Yield ~ Block + SO(x1, x2), data = CR)
summary(CR.rsm2)

# The secondary point is a maximum (both eigenvalues are negative) and within the experimental design range (no extrapolation)

# Also note that the block is significant, meaning that the processes shifted between the first set of data and the second.  This is not good.  The coefficient is -4.5, meaning the yield shifted down by 4.5% between the two blocks - a more significant effect than either temperatue or time!  This is most easily seen by looking at the repeat center points.

# We can plot the fitted response as a contour plot.
contour(CR.rsm2, ~ x1 + x2, at = summary(CR.rsm2)$canonical$xs)


#OTHER example
library(rsm)
cube (2, n0 = 4)
ccd.pick(k=2)

#Copy this matrix for example
x1  x2      Time   Temp  y
-1  -1      80     179    76.5
```

```
-1  1    80    180   77
1   0.1  90    179   78
1   1    90    180   79.5
0   0    85    175   79.9
0   0    85    175   80.3
0   0    85    175   80
0   0    85    175   79.7

ccd1 <- read.table(file = "clipboard",  sep = "\t", header=TRUE)
ccd1<- as.coded.data(cdd1, x1 ~ (Time-85)/5, x2 ~ (Temp-175)/5)

Model_Y1<- rsm(y ~ SO(x1, x2), data = ccd1)
Model_Y3<- rsm(y ~ FO(x1, x2) + PQ(x1, x2), data = ccd1)

par(mfrow = c(1, 2))
contour(Model_Y3, ~ x1+x2, image = TRUE, yagp=c(168,182, 2),
  xlabs=c("Time", "Temp"))
points(ccd1$Time, ccd1$Temp)
 persp(Model_Y3,  x1~x2, col = terrain.colors(50), contours = "colors")

max <- data.frame(x1 = 0.361, x2 = 0.257)




```

central composite designs 3 factors

```
library(rsm)
ccd.pick(k=3)

data(Treb)
treb.quad <- rsm(y ~ SO(x1, x2, x3), data = Treb)
summary(treb.quad)

par (mfrow=c(2,2))
contour(treb.quad, ~ x1+x2+x3 )
```

3D response surface experiment

```
par (mfrow=c(1,3))
persp(treb.quad, ~ x1+x2+x3, zlab="Distance", contours=list(z="bottom"))
```

3D response surface experiment one factor

```
par (mfrow=c(1,3))
contour(treb.quad, x1~x3, at=list(x2=1))
persp(treb.quad, x1~x3, at=list(x2=1),zlab="Distance", contours=list(z="bottom"))

```

DETERMINING OPTIMUM OPERATING CONDITIONS

```
ridge<-steepest(treb.quad, dist=seq(0, 1.412, by=.1), descent=FALSE)

ridge


```

MIXTURE EXPERIMENTS

```

library(mixexp)
SLD(3,2)
des<-SLD(3,3)
DesignPoints(des)

library(daewr)
data(pest)
DesignPoints(pest)
 spc <- lm(y ~ x1 + x2 + x3 + x1:x2 + x1:x3 + x2:x3 + x1:x2:x3 -1, data = pest)
summary(spc)

MixturePlot(des = pest,mod = 2)

EffPlot(des=pest,mod=2,dir=1)

```

Other example of ANALYSIS OF MIXTURE EXPERIMENTS

```

data(polvdat)
sqm <- lm(y ~ x1 + x2 + x3 + x1:x2 + x1:x3 + x2:x3 + x1:x2:x3 -1, data = polvdat)
summary(sqm)

MixturePlot(des = polvdat, mod = 4, lims=c(0,.8,.1,.95, .05, .50), constrts=TRUE, pseudo=TRUE)


```

#Analyzing literatures blibliometrix

```

library(bibliometrix)
```

```
M <- convert2df(file = "HABSscopus.bib" , dbsource = "scopus", format = "bibtex")
results <- biblioAnalysis(M, sep = ";")
S <- summary(object = results, k = 10, pause = FALSE)
plot(x = results, k = 10, pause = FALSE)
ID <- cocMatrix(M, Field = "ID", sep = ";")
sort(Matrix::colSums(ID), decreasing = TRUE)[1:20]
rm(A)
AB <- cocMatrix(M, Field = "AB", sep = " ")
sort(Matrix::colSums(AB), decreasing = TRUE)[1:100]

# Create keyword co-occurrences network

NetMatrix <- biblioNetwork(M, analysis = "co-occurrences", network = "keywords", sep = ";")

# Plot the network
net=networkPlot(NetMatrix, normalize="association", weighted=T, n = 30, Title = "Keyword Co-occurrences", type = "fruchterman", size=T,edgesize = 5,labelsize=0.7)

# Conceptual Structure using keywords (method="CA")

CS <- conceptualStructure(M,field="ID", method="CA", minDegree=4, clust=5, stemming=FALSE, labelsize=10, documents=10)

#bio3d working with PDB files
# List of bio3d functions with brief description
help(package=bio3d)

# Detailed help on a particular function, e.g. 'pca.xyz'
help(pca.xyz)

pdb <- read.pdb("4q21")

attributes(pdb)

head(pdb$atom)

# Print $atom data for the first two atoms
pdb$atom[1:2, ]

```

```
# Print a subset of $atom data for the first two atoms
pdb$atom[1:2, c("eleno", "elety", "x","y","z")]

# Note that individual $atom records can also be accessed like this
pdb$atom$elety[1:2]

# Which allows us to do the following (see Figure 1.)
plot.bio3d(pdb$atom$b[pdb$calpha], sse=pdb, typ="l", ylab="B-factor")

# Examine the row and column dimensions
dim(pdb$xyz)

pdb$xyz[ 1, atom2xyz(1:2) ]

# Select all C-alpha atoms (return their indices)
ca.inds <- atom.select(pdb, "calpha")
ca.inds

# Print details of the first few selected atoms
head( pdb$atom[ca.inds$atom, ] )

# And selected xyz coordinates
head( pdb$xyz[, ca.inds$xyz] )

# Select chain A
a.inds <- atom.select(pdb, chain="A")

# Select C-alphas of chain A
ca.inds <- atom.select(pdb, "calpha", chain="A")

# We can combine multiple selection criteria to return their intersection
cab.inds <- atom.select(pdb, elety=c("CA","CB"), chain="A", resno=10:20)

# Select all atoms except waters
nowat.inds <- atom.select(pdb, "water", inverse=TRUE)

# Select protein + GDP
sele <- atom.select(pdb, "protein", resid="GDP", operator="OR")

sele <- atom.select(pdb, "protein", elety=c("N", "CA", "C"), resno=50:60, verbose=T)

a.inds <- atom.select(pdb, "protein")
b.inds <- atom.select(pdb, resid="GDP")
```

```
sele <- combine.select(a.inds, b.inds, operator="OR")

aa321(aa3)

# Output a backbone only PDB file to disc
b.inds <- atom.select(pdb, "back")
backpdb <- trim.pdb(pdb, b.inds)
write.pdb(backpdb, file="4q21_back.pdb")

# Selection statements can be passed directly to trim.pdb()
backpdb <- trim.pdb(pdb, "backbone")

# The 'value=TRUE' option of atom.select() will result in a PDB object being returned
backpdb <- atom.select(pdb, "backbone", value=TRUE)

# Renumber all residues
write.pdb(backpdb, resno=backpdb$atom$resno+10)

# Assign chain B to all residues
write.pdb(backpdb, chain="B")

pdb <- read.pdb("4lhy")

# select chains A, E and F
inds <- atom.select(pdb, chain=c("A", "E", "F"))

# trim PDB to selection
pdb2 <- trim.pdb(pdb, inds)

# assign new chain identifiers
pdb2$atom$chain[ pdb2$atom$chain=="E" ] <- "B"
pdb2$atom$chain[ pdb2$atom$chain=="F" ] <- "C"

# re-number chain B and C
pdb2$atom$resno[ pdb2$atom$chain=="B" ] <- pdb2$atom$resno[ pdb2$atom$chain=="B" ] - 156
pdb2$atom$resno[ pdb2$atom$chain=="C" ] <- pdb2$atom$resno[ pdb2$atom$chain=="C" ] - 156

# assign the GDP residue a residue number of 500
pdb2$atom$resno[ pdb2$atom$resid=="GDP" ] <- 500

# use chain D for the GDP residue
```

```r
pdb2$atom$chain[ pdb2$atom$resid=="GDP" ] <- "D"

# Center, to the coordinate origin, and orient, by principal axes,
# the coordinates of a given PDB structure or xyz vector.
xyz <- orient.pdb(pdb2)

# write the new pdb object to file
write.pdb(pdb2, xyz=xyz, file="4LHY_AEF-oriented.pdb")

# read two G-protein structures
a <- read.pdb("4q21")
b <- read.pdb("4lhy")

a1 <- trim.pdb(a, chain="A")

b1 <- trim.pdb(b, chain="A")
b2 <- trim.pdb(b, chain="E")
b3 <- trim.pdb(b, chain="F")


# Redundant testing excluded FASTA
# Read sequence alignment
file <- system.file("examples/kif1a.fa",package="bio3d")
aln <- read.fasta(file)
# Read aligned PDBs
pdbs <- read.fasta.pdb(aln)
# Structure/sequence names/ids
basename( pdbs$id )
# Alignment positions 335 to 339
pdbs$ali[,335:339]
pdbs$resid[,335:339]
pdbs$resno[,335:339]
pdbs$b[,335:339]
# Alignment C-alpha coordinates for these positions
pdbs$xyz[, atom2xyz(335:339)]
# See 'fit.xyz()' function for actual coordinate superposition
# e.g. fit to first structure
# xyz <- fit.xyz(pdbs$xyz[1,], pdbs)
# xyz[, atom2xyz(335:339)]

# concatenate PDBs
new <- cat.pdb(a1, b1, b2, b3, rechain=TRUE)
unique(new$atom$chain)
```

```r

# write new PDB object to file
write.pdb(new, file="4Q21-4LHY.pdb")

# Align and superpose two or more structures
pdbs <- pdbaln(c("4q21", "521p"), fit=TRUE)

# read two G-protein structures
a <- read.pdb("4q21")
b <- read.pdb("4lhy")

# perform iterative alignment
aln <- struct.aln(a, b)

# store new coordinates of protein B
b$xyz <- aln$xyz

# indices at which the superposition should be based
a.ind <- atom.select(a, chain="A", resno=87:103, elety="CA")
b.ind <- atom.select(b, chain="A", resno=93:109, elety="CA")

# perform superposition
xyz <- fit.xyz(fixed=a$xyz, mobile=b$xyz,
               fixed.inds=a.ind$xyz,
               mobile.inds=b.ind$xyz)

# write coordinates to file
write.pdb(b, xyz=xyz, file="4LHY-at-4Q21.pdb")

# read G-protein structure
pdb <- read.pdb("4q21")
bs <- binding.site(pdb)

# residue names of identified binding site
print(bs$resnames)

b <- read.pdb("4lhy")

# atom selection
a.inds <- atom.select(b, chain="A")
b.inds <- atom.select(b, chain=c("E", "F"))

# identify interface residues
```

```r
bs <- binding.site(b, a.inds=a.inds, b.inds=b.inds)

# use b-factor column to store interface in PDB file
b$atom$b[ bs$inds$atom ] <- 1
b$atom$b[ -bs$inds$atom ] <- 0

# write to file
write.pdb(b, file="4LHY-interface.pdb")

b <- read.pdb("4lhy")

# atom selection
a.inds <- atom.select(b, chain="A")
b.inds <- atom.select(b, chain=c("E", "F"))

# identify interface residues
bs <- binding.site(b, a.inds=a.inds, b.inds=b.inds)

# use b-factor column to store interface in PDB file
b$atom$b[ bs$inds$atom ] <- 1
b$atom$b[ -bs$inds$atom ] <- 0

# write to file
write.pdb(b, file="4LHY-interface.pdb")

# The xyz component contains 20 frames
pdb$xyz

# Select a subset of the protein
ca.inds <- atom.select(pdb, "calpha")

# Access C-alpha coordinates of the first 5 models
#pdb$xyz[1:5, ca.inds$xyz]

library(bio3d.geostas)

# Domain analysis
gs  <- geostas(pdb)

# Fit all frames to the 'first' domain
domain.inds <- gs$inds[[1]]

xyz <- pdbfit(pdb, inds=domain.inds)

```

```
# write fitted coordinates
write.pdb(pdb, xyz=xyz, chain=gs$atomgrps, file="1d1d_fit-domain1.pdb")

# plot geostas results
plot(gs, contour=FALSE)

# Invariant core
core <- core.find(pdb)

# fit to core region
xyz <- pdbfit(pdb, inds=core)

# write fitted coordinates
write.pdb(pdb, xyz=xyz, file="1d1d_fit-core.pdb")

# Read PDB from online database
pdb <- read.pdb("2dn1")

# Examine biological unit matrices
pdb$remark$biomat

biounit(pdb)

bio <- biounit(pdb)
names(bio)

# Download some example PDB files
ids <- c("1TND_B","1AGR_A","1FQJ_A","1TAG_A","1GG2_A","1KJY_A")
raw.files <- get.pdb(ids)

# Extract and align the chains we are interested in
files <- pdbsplit(raw.files, ids)
pdbs <- pdbaln(files,
   exefile="C:/Users/marce/Documents/R/bioinformatics/muscle3.exe")
# Calculate sequence identity
pdbs$id <- basename.pdb(pdbs$id)
seqidentity(pdbs)

## Calculate RMSD
rmsd(pdbs, fit=TRUE)

## Quick PCA (see Figure 9)
pc <- pca(pdbfit(pdbs), rm.gaps=TRUE)
```

```
plot(pc)

## Quick NMA of all structures (see Figure 10)
library(bio3d.nma)
modes <- nma(pdbs)
plot(modes, pdbs, spread=TRUE)

library(rmarkdown)
render("Bio3D_pdb.Rmd", "all")

# Information about the current Bio3D session
sessionInfo()

attach(transducin)

pdb <- read.pdb("1tag")
seq <- pdbseq(pdb)
blast <- blast.pdb(seq)

# See Figure 2.
hits <- plot.blast(blast, cutoff=240)

head(hits$hits)

# Download PDBs and split by chain ID
files <- get.pdb(hits, path="raw_pdbs", split = TRUE)

# Extract and align sequences
pdbs <- pdbaln(files, exefile="C:/Users/marce/Documents/R/bioinformatics/muscle3.exe")
core <- core.find(pdbs)

# See Figure 3.
col=rep("black", length(core$volume))
col[core$volume<2]="pink"; col[core$volume<1]="red"
plot(core, col=col)

xyz <- pdbfit( pdbs, core.inds )

rd <- rmsd(xyz)
hist(rd, breaks=40, xlab="RMSD (Å)", main="Histogram of RMSD")

```

```
# RMSD clustering
hc.rd <- hclust(as.dist(rd))

pdbs$id <- substr(basename(pdbs$id), 1, 6)
hclustplot(hc.rd, colors=annotation[, "color"], labels=pdbs$id, cex=0.5,
        ylab="RMSD (Å)", main="RMSD Cluster Dendrogram", fillbox=FALSE)

# Ignore gap containing positions
gaps.res <- gap.inspect(pdbs$ali)
gaps.pos <- gap.inspect(pdbs$xyz)

# Tailor the PDB structure to exclude gap positions for SSE annotation
id <- grep("1TAG", pdbs$id)
inds <- atom.select(pdb, resno = pdbs$resno[id, gaps.res$f.inds])
ref.pdb <- trim.pdb(pdb, inds = inds)

# Plot RMSF with SSE annotation and labeled with residue numbers (Figure 8.)
rf <- rmsf(xyz[, gaps.pos$f.inds])
plot.bio3d(rf, resno=ref.pdb, sse=ref.pdb, ylab="RMSF (Å)",
        xlab="Residue No.", typ="l")

tor <- torsion.pdb(pdb)

# Basic Ramachandran plot (Figure 9)
plot(tor$phi, tor$psi, xlab="phi", ylab="psi")

# Locate the two structures in pdbs
ind.a <- grep("1TAG_A", pdbs$id)
ind.b <- grep("1TND_B", pdbs$id)

# Exclude gaps in the two structures to make them comparable
gaps.xyz2 <- gap.inspect(pdbs$xyz[c(ind.a, ind.b), ])
a.xyz <- pdbs$xyz[ind.a, gaps.xyz2$f.inds]
b.xyz <- pdbs$xyz[ind.b, gaps.xyz2$f.inds]

# Compare CA based pseudo-torsion angles between the two structures
a <- torsion.xyz(a.xyz, atm.inc=1)
b <- torsion.xyz(b.xyz, atm.inc=1)
d.ab <- wrap.tor(a-b)
d.ab[is.na(d.ab)] <- 0

```

```
# Plot results with SSE annotation
plot.bio3d(abs(d.ab), resno=pdb, sse=pdb, typ="h", xlab="Residue No.",
           ylab = "Difference Angle")

a <- dm.xyz(a.xyz)
b <- dm.xyz(b.xyz)

plot.dmat( (a - b), nlevels=10, grid.col="gray", xlab="1tag", ylab="1tnd")

# Do PCA
pc.xray <- pca.xyz(xyz[, gaps.pos$f.inds])
pc.xray

plot(pc.xray, col=annotation[, "color"])

plot(pc.xray, pc.axes=1:2, col=annotation[, "color"])

# Left-click on a point to label and right-click to end
identify(pc.xray$z[,1:2], labels=basename.pdb(pdbs$id))

par(mfrow = c(3, 1), cex = 0.75, mar = c(3, 4, 1, 1))
plot.bio3d(pc.xray$au[,1], resno=ref.pdb, sse=ref.pdb, ylab="PC1")
plot.bio3d(pc.xray$au[,2], resno=ref.pdb, sse=ref.pdb, ylab="PC2")
plot.bio3d(pc.xray$au[,3], resno=ref.pdb, sse=ref.pdb, ylab="PC3")

hc <- hclust(dist(pc.xray$z[,1:2]))
grps <- cutree(hc, h=30)
cols <- c("red", "green", "blue")[grps]
plot(pc.xray, pc.axes=1:2, col=cols)


#Enhanced Normal Modes Analysis with Bio3D


modes <- nma(pdb)

print(modes)

plot(modes, sse=pdb)

# Calculate modes with various force fields
modes.a <- nma(pdb, ff="calpha")
modes.b <- nma(pdb, ff="anm")# Make a PDB trajectory
```

```
mktrj(modes, mode=7)

# Vector field representation (see Figure 3.)
pymol(modes, mode=7)


modes.c <- nma(pdb, ff="pfanm")
modes.d <- nma(pdb, ff="reach")
modes.e <- nma(pdb, ff="sdenm")

# Root mean square inner product (RMSIP)
r <- rmsip(modes.a, modes.b)

plot(r, xlab="ANM", ylab="C-alpha FF")

# Download PDB, calcualte normal modes of the open subunit
pdb.full   <- read.pdb("1sx4")
pdb.open   <- trim.pdb(pdb.full, atom.select(pdb.full, chain="A"))
modes      <- nma(pdb.open)



# Calculate the cross-correlation matrix
cm <- dccm(modes)

# Plot a correlation map with plot.dccm(cm)
plot(cm, sse=pdb.open, contour=F, col.regions=bwr.colors(20),
  at=seq(-1,1,0.1) )

# View the correlations in the structure (see Figure 5.)
pymol(cm, pdb.open, type="launch")

# Deformation energies
defe <- deformation.nma(modes)
defsums <- rowSums(defe$ei[,1:3])

# Fluctuations
flucts <- fluct.nma(modes, mode.inds=seq(7,9))

# Write to PDB files (see Figure 6.)
write.pdb(pdb=NULL, xyz=modes$xyz, file="R-defor.pdb", b=defsums)
write.pdb(pdb=NULL, xyz=modes$xyz, file="R-fluct.pdb", b=flucts)

# Closed state of the subunit
```

```r
pdb.closed <- trim.pdb(pdb.full, atom.select(pdb.full, chain="H"))

# Align closed and open PDBs
aln <- struct.aln(pdb.open, pdb.closed, max.cycles=0)
pdb.closed$xyz <- aln$xyz

# Caclulate a difference vector
xyz <- rbind(pdb.open$xyz[aln$a.inds$xyz], pdb.closed$xyz[aln$a.inds$xyz])
diff <- difference.vector(xyz)

# Calculate overlap
oa <- overlap(modes, diff)

plot(oa$overlap, type='h', xlab="Mode index", ylab="Squared overlap", ylim=c(0,1))
points(oa$overlap, col=1)
lines(oa$overlap.cum, type='b', col=2, cex=0.5)
text(c(1,5)+.5, oa$overlap[c(1,5)], c("Mode 1", "Mode 5"), adj=0)


```

#Mining text Analyzing text tidy approach for articles details

IMPORTANT FOR LINUX UBUNTU

```r
# for working with library(bibliometrix):
# install dependencies
# sudo apt-get update -y
#sudo apt-get install -y r-cran-factominer

# some text
text <- c("Because I could not stop for Death -",
          "He kindly stopped for me -",
          "The Carriage held but just Ourselves -",
          "and Immortality")
```

#IMPORT documents from pdf

```r

library(tidyverse)
library(pdftools)
#document split in pages
HA1_2015 <- pdf_text("F:/Work/ARTtoDO/ClampH1N1/2015RBS_HA1.pdf")
```

```r
#document split in pages
HA1_2015str <- pdf_text("F:/Work/ARTtoDO/ClampH1N1/2015RBS_HA1.pdf") %>% str_split("\n")

#All line together page by page
HA1<-str_squish("HA1_2015")

#final data frame
final_df <- as_tibble(HA1_2015)


#Working with multiple .pdf files
#sew wd to where files are located
setwd("F:/Work/ARTtoDO/ClampH1N1")

#create character list of all files ending with .pdf in folder
files <- list.files("F:/Work/ARTtoDO/ClampH1N1/", pattern = "pdf$")

#use lapply() to apply pdf_text or other pdftools function interactively cross each of the files
all_pdf<- lapply(files, pdf_text)#
#export as excel table each pdf file
library(writexl)
df<-data.frame(all_pdf[[1]])
write_xlsx(df, "C:/Users/marce/Documents/R/scientometrics/name.xlsx")

#working with Text Mining(TM) package
library(tm)
files <- list.files("F:/Work/ARTtoDO/ClampH1N1/", pattern = "pdf$")
corp <- Corpus(URISource(files),
               readerControl = list(reader = readPDF))

# create a term-document matrix, or TDM for short. A TDM stores counts of terms for each document.

opinions.tdm <- TermDocumentMatrix(corp,
                                   control =
                                    list(removePunctuation = TRUE,
                                         stopwords = TRUE,
                                         tolower = TRUE,
                                         stemming = TRUE,
                                         removeNumbers = TRUE,
                                        bounds = list(global = c(3, Inf))))



#the removePunctuation function
corp <- tm_map(corp, removePunctuation, ucp = TRUE)
opinions.tdm <- TermDocumentMatrix(corp,
                              control =
                                list(stopwords = TRUE,
                                     tolower = TRUE,
                                     stemming = TRUE,
                                     removeNumbers = TRUE,
                                     bounds = list(global = c(3, Inf))))
inspect(opinions.tdm[1:10,])

findFreqTerms(opinions.tdm, lowfreq = 100, highfreq = Inf)

ft <- findFreqTerms(opinions.tdm, lowfreq = 100, highfreq = Inf)
as.matrix(opinions.tdm[ft,])

ft.tdm <- as.matrix(opinions.tdm[ft,])
sort(apply(ft.tdm, 1, sum), decreasing = TRUE)

# more examples Working with multiple pdf files
library(pdftools)
library(tm)
library(wordcloud)
library(RColorBrewer)
files <- list.files("~/R/scientometrics", pattern = "pdf$") # Vector of pdf file names
pdfs <- lapply(files, pdf_text)# loads all three files
length(pdfs)# verify how many files are in the pdfs
lapply(pdfs, length) # check the length of each pdf file
#Create a corpus with the vector that has the three files
pdfdatabase <- Corpus(URISource(files), readerControl = list(reader = readPDF))# creating a PDF database

#Lets clean up the corpus
#create your own list of stopwords, it has to be performed on the Corpus
pdfdatabase<- tm_map(pdfdatabase, removeWords, c("abuse", "access", "affect"))
#remove english stopwords
```

```
1394.  pdfdatabase <- tm_map(pdfdatabase, removeWords,
       stopwords("english"))
1395.  # Remove numbers
1396.  pdfdatabase <- tm_map(pdfdatabase, removeNumbers)
1397.#convert Vcorpus to TIBBLE and data.frame
1398.
1399.
1400.
1401.
1402.#Only words that appear at least two times are counted in this
       example.
1403.  pdfs.tdm <- TermDocumentMatrix(pdfdatabase,control =
       list(removePunctuation = TRUE, stopwords = TRUE, tolower = TRUE,
       stemming = FALSE, removeNumbers = TRUE, bounds = list(global =
       c(2,Inf))))
1404.  #Examine  10 words at a time in across documents. The range below
       specifies the first 10.
1405.
1406.  inspect(pdfs.tdm[1:10,])
1407.
1408.  #Frequent terms that appear at least 20 times across all documents
1409.
1410.  findFreqTerms(pdfs.tdm, lowfreq = 20, highfreq = Inf)
1411.
1412.#Compare the distribution of frequently appearing words
       across the three documents.
1413.
1414.  ft <- findFreqTerms(pdfs.tdm, lowfreq = 20, highfreq = Inf)
1415.  as.matrix(pdfs.tdm[ft,])
1416.
1417.  #Sum the count of all frequently occurring words
1418.
1419.  ft.tdm <- as.matrix(pdfs.tdm[ft,])
1420.  sort(apply(ft.tdm, 1, sum), decreasing = TRUE)
1421.
1422.  #Lets complete the second part of this exercise by conducting correlation
       analysis and creating graphs and charts. Lets find frequent terms that appear
       at least 10 times.
1423.
1424.  findFreqTerms(pdfs.tdm, lowfreq = 10)
1425.
```

```r
#Examine frequent Terms and their association. In this example we are looking at the frequent terms related to bullying. The correlation limit that is being examined is a correlation of 75% or greater.

findAssocs(pdfs.tdm, terms = "bullying", corlimit = 0.75)

#To create a word cloud or bar chart you must convert the term document matrix to a data frame.

m <- as.matrix(pdfs.tdm)
v <- sort(rowSums(m),decreasing=TRUE)
d <- data.frame(word = names(v),freq=v)

#Create word cloud

set.seed(1234)
wordcloud(words = d$word, freq = d$freq, min.freq = 1,
          max.words=200, random.order=FALSE, rot.per=0.35,
          colors=brewer.pal(8, "Dark2"))

#Create Bar chart

barplot(d[1:11,]$freq, las = 2, names.arg = d[1:11,]$word,
        col ="lightblue", main ="Most frequent words",
        ylab = "Word frequencies")

#Converting PDFs to data.frame
library(dplyr)
text_df <- tibble(line = 1:2, text = dfvc)




#Tidy text dataset, we first need to put it into a data frame
library(dplyr)
text_df <- tibble(line = 1:4, text = text)

#Split in single word
library(tidytext)
library(janeaustenr)
library(dplyr)
library(stringr)

```

```

allPDF <- read.table(file = "clipboard",  sep = "\t", header=TRUE)

#Art is the name of the column
pdf_T<- allPDF %>%  unnest_tokens(word, Art)

#Remove stop words such as “the”, “of”, “to”, and so forth in English
data(stop_words)
tidyT <- pdf_T %>% anti_join(stop_words)
#find the most common words in all the books as a whole.
tidyT %>% count(word, sort = TRUE)


#Plot words change filter number according to the previous step
library(ggplot2)
tidyT %>%
  count(word, sort = TRUE) %>%
  filter(n > 10) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(n, word)) +
  geom_col() +
  labs(y = NULL)

#Wordclouds
library(wordcloud)
tidyT %>%
  anti_join(stop_words) %>%
  count(word) %>%
  with(wordcloud(word, n, max.words = 100))

#work with physics books
library(gutenbergr)
physics <- gutenberg_download(c(37729, 14725, 13476, 30155),
                              meta_fields = "author")

#Split using two words changing the n = we can work with 3 words (trigram) ect
#Art is the name of the column
pdf_2T<- allPDF %>%  unnest_tokens(bigram, Art, token = "ngrams", n = 2)
pdf_2T %>%
  count(bigram, sort = TRUE)

```

```
#working with 2 words by removing “stop-words”
# https://www.tidytextmining.com/ngrams.html
bigrams_separated <- pdf_2T %>%
  separate(bigram, c("word1", "word2"), sep = " ")

bigrams_filtered <- bigrams_separated %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)

# new bigram counts:
bigram_counts <- bigrams_filtered %>%
  count(word1, word2, sort = TRUE)

bigram_counts

#reunite  words after cleaning
bigrams_united <- bigrams_filtered %>%
  unite(bigram, word1, word2, sep = " ")

# filter for only relatively common combinations
library(igraph)
bigram_graph <- bigram_counts %>%
  filter(n > 2) %>%
  graph_from_data_frame()
library(ggraph)
set.seed(2017)
bigram_graph

#Plotting 1 interaction between words
ggraph(bigram_graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point() +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1)

#Plotting 2 interaction between words
set.seed(2020)
a <- grid::arrow(type = "closed", length = unit(.15, "inches"))

ggraph(bigram_graph, layout = "fr") +
  geom_edge_link(aes(edge_alpha = n), show.legend = FALSE,
                 arrow = a, end_cap = circle(.07, 'inches')) +
  geom_node_point(color = "lightblue", size = 5) +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
  theme_void()


#collect it into a function so that we easily perform it on other text datasets.
#To make it easy to use the count_bigrams() and visualize_bigrams()

library(dplyr)
library(tidyr)
library(tidytext)
library(ggplot2)
library(igraph)
library(ggraph)

count_bigrams <- function(dataset) {
  dataset %>%
    unnest_tokens(bigram, text, token = "ngrams", n = 2) %>%
    separate(bigram, c("word1", "word2"), sep = " ") %>%
    filter(!word1 %in% stop_words$word,
           !word2 %in% stop_words$word) %>%
    count(word1, word2, sort = TRUE)
}

visualize_bigrams <- function(bigrams) {
  set.seed(2016)
  a <- grid::arrow(type = "closed", length = unit(.15, "inches"))

  bigrams %>%
    graph_from_data_frame() %>%
    ggraph(layout = "fr") +
    geom_edge_link(aes(edge_alpha = n), show.legend = FALSE, arrow = a) +
    geom_node_point(color = "lightblue", size = 5) +
    geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
    theme_void()
}
#then applying functions
library(stringr)
tri_bigrams <- pdf_3T  %>%
  count_bigrams()

tri_bigrams %>%
  filter(n > 40,
  !str_detect(word1, "\\d"),
  !str_detect(word2, "\\d")),
  !str_detect(word2, "\\d")) %>%
  visualize_bigrams()


```