Kaggle 2

Nicholas Scholl

2023-10-03

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
test_rna <- as.matrix(read.csv("test_set_rna.csv", row.names = 1))
train_rna <- as.matrix(read.csv("training_set_rna.csv", row.names = 1))
train_adt <- as.matrix(read.csv("training_set_adt.csv", row.names = 1))
```

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(rmarkdown)
```

loading in the necessary packages we will use.

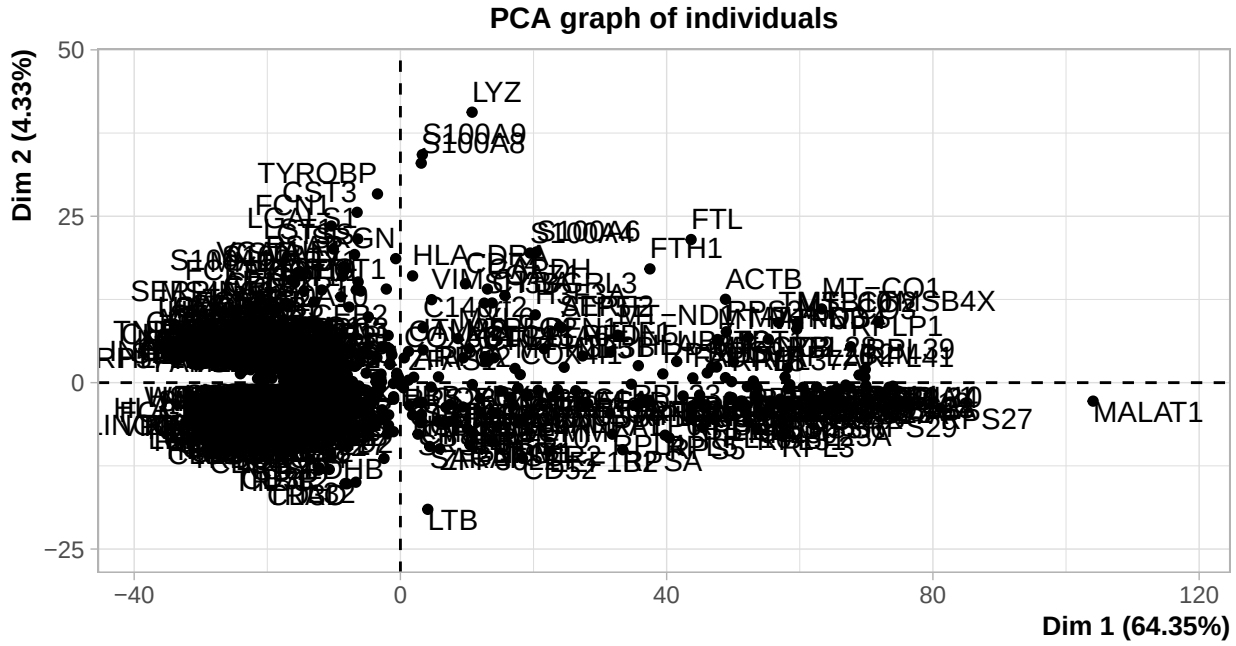
```
A <- rbind(train_adt, train_rna)
B <- rbind(matrix(0, 25, 1000), test_rna)
```

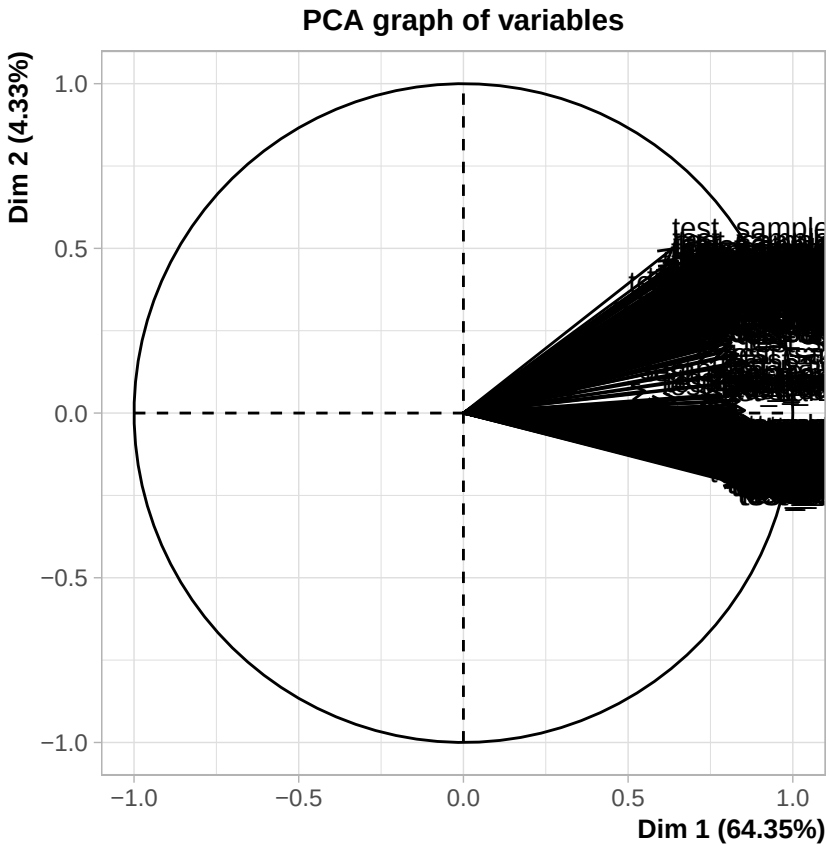
This is code from the 'getting started' section, here combining the train data sets together with A and then binding them together again with the test RNA set with B.

```
#A <- scale(A, center = TRUE, scale = FALSE)
#train_adt <- scale(train_adt, center = TRUE, scale = FALSE)
#train_rna <- scale(train_rna, center = TRUE, scale = FALSE)
```

Scaling and centering the data with MLR produced results in the low to mid 70's in terms of accuracy when uploading to kaggle. Using unscaled data and uncentered data with MLR could produce a result that is 80%, thus showing that unscaled and uncentered data is more accurate since the data is already pre-processed with log normalizing.

```
library(FactoMineR)
PCA_test_RNA <- PCA(test_rna)
```





Here, factors are displayed to show variance, seeing that 80 factors can explain the variance of the data, this can be used for the SVD equation. This chart was originally created for PCA, this was proven difficult to implement without the use of a package and opting to make SVD algorithm's proved to be more effective. This was also proven using trial and error switching the K value for SVD to 10, 100, 150, and 300. 80 produced the best result for SVD.

```
library(irlba)
```

```
## Loading required package: Matrix
```

```
SVD_train <- irlba(A, 80)
#PCA_model <- prcomp_irlba(train_rna, n = 10, scale = TRUE)

#estimate_SVD_train <- SVD_train$u %*% Matrix::Diagonal(x = SVD_train$d) %*% t(SVD_train$v)


# normal equation for finding A_{testRNA} = U_{train}V_{test}

# solve for V_{test}:
a <- t(SVD_train$u[26:664,]) %*% SVD_train$u[26:664,]
b <- t(test_rna) %*% SVD_train$u[26:664, ]
V_test <- apply(b, 1, function(b0) solve(a, b0))
ADT_test <- SVD_train$u[1:25, ] %*% V_test

#A is a matrix of both the train adt and train rna
```

This is the SVD equation, utilizing the library irlba for our main function irlba(). First calculating the u and v matrix from that equation and reference it later when to use the solve equation in this block. Utilizing the matrix multiplication to solve for the unknown x or in this case, the ADT matrix for test. Solving the a and b matrices to use them in the final calculation for unknown X. This is stored in the unknown X to 'V_test' and then multiplying that by the test RNA data set to get what will be test ADT set. Several different iterations below have been tried utilizing the u and v matrix in MLR, regressing on the diagonal matrix and various others. All proven to be difficult to implement in R with errors.

```
cumulative_var <- cumsum(SVD_train$d^2) / sum(SVD_train$d^2)
```

Here the block is computing the variance explained by each component. The goal here is to check how well our K performed and if it did a good job of explaining the variance of the data set. This is done to cross validate the results from our SVD equation. Then dividing the cumulative sum over the sum of our diagonal matrices to get a percentage of how well the variance was explained. In it, we confirm that 80 is the best performing K value we can use for our SVD equation.

```
library(caret)
library(tibble)
library(data.table)
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
trans_trna <- t(train_rna)
trans_tadt <- t(train_adt)

train_correlation <- cor(trans_tadt, trans_trna, use = "complete.obs")
#condition <-  train_correlation < 1
#condition2 <- train_correlation > 0.3
#filtered_tc <- train_correlation[which(condition)]
#filtered_tc <- train_correlation[which(condition2)]
#filtered_tc <- as.data.frame.table(filtered_tc)
train_corr_df <- as.data.frame.table(train_correlation)
upper <- 1
lower <- 0.2
filter_cor_df <-as.data.frame.table(filter(train_corr_df, between(train_corr_df$Freq, lower, upper)))

keep <- unique(filter_cor_df$Freq.Var2)

keep2 <- rownames(train_rna) %in% keep


filtered_train_RNA <- train_rna[keep2,]
filtered_test_RNA <- test_rna[keep2,]
```

Please note: Rachel Orzechowski assisted me in this code chunk. SHE DID NOT SHARE CODE, she shared ideas of how to calculate the correlation matrix and recommended equations to use and with some troubleshooting along with Sarah Knight.

This code chunk filters out proteins that are not correlated together between the train adt and train RNA sets. First calculating the correlation between all the data points with their counterparts and make that into a data frame table. Next, is to filter out values between 0.20 and 1 so proteins with some to very strong correlation are left. Next filtering out repetitive and duplicate data with unique() and finally make a new data set with the proteins that match between keep2 and their respective RNA sets. This filtering was by far the best means to pre-process the data producing results of 80.704% accuracy after using MLR.

```
j <- filtered_train_RNA %*% t(filtered_train_RNA)
k <- train_adt %*% t(filtered_train_RNA)
mat2 <- matrix(0, 368, 25)
for(i in 1:nrow(k)){


mat2[, i] <- solve(j, k[i,])
}
```

Matrix multiplication with the new filtered data sets from above and for each row in the k matrix (train_adt %*% the transpose matrix of filtered train RNA) then using the solve function to add a new entry into our blank matrix mat2 to generate the x for the matrix multiplication problem. This is done to build a complete data set for mat. This produces the results mentioned above of 80.704%

```
filtered_submit <- t(mat2) %*% filtered_test_RNA
```

Here is simply using the 'x' from our previous code blocks and multiplying them together to get what the test ADT data set that will be submitted to kaggle.

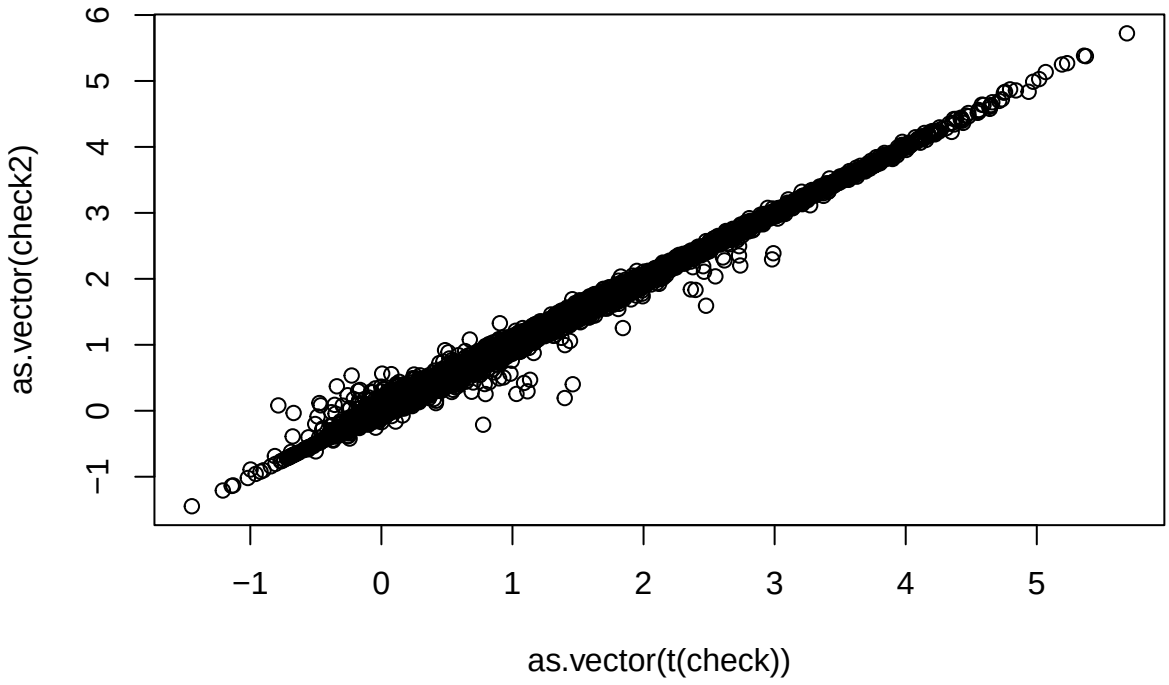
```
mat <- matrix(0, 639, 25)
for(i in 1:nrow(train_adt)){
a <- train_rna %*% t(train_rna)
b <- train_rna %*% train_adt[i, ]

mat[, i] <- solve(a, b)
}
```

Here the code block utilize simple MLR for unfiltered and unprocessed data. Each row of train ADT and utilizing the solve function to create a new column in a matrix called mat to store the results of matrix multiplication to produce the x variable in our matrix multiplication problem. This was the first attempt used in the project, producing results of 80.02% accuracy providing a good baseline to compare results of SVD and correlation filtering.

```
check <- predict(lm(t(train_adt) ~ t(train_rna)))
check2 <- t(t(train_rna) %*% mat)
```

We add these checks to validate the results of our MLR model. We see if our MLR algorithm produces similar results as using the linear model function in R. In the next code chunk, we will see how correlated they are. Strong correlation implies that our MLR algorithm is working as intended and can produce strong results.

```
plot(as.vector(t(check)), as.vector(check2))
```




```
#plot(check2)
```

Here we plot what we mentioned above to check the results of the mat matrix from our simple MLR approach. We can see a strong correlation between the two of them, this implies that our model performs well against the linear model's coefficients.

```
sample_size <- floor(0.75 * ncol(train_adt))
set.seed(123)
train_indexa <- sample(seq_len(ncol(train_adt)), size = sample_size)

validation_trainADT <- train_adt[, train_indexa ]
validation_testADT <- train_adt[, -train_indexa]

train_indexr <- sample(seq_len(ncol(train_rna)), size = sample_size)

validation_trainRNA <- train_rna[, train_indexa ]
validation_testRNA <- train_rna[, -train_indexa]
```

In this chunk, we are randomly splitting our ADT and RNA training sets for cross validation. We split it into 1/4 and 3/4 portions. The 3/4 will act as our training set and the 1/4 will be our test set for validation.

```
j <- validation_trainRNA %*% t(validation_trainRNA)
k <- validation_trainADT %*% t(validation_trainRNA)
check3 <- matrix(0, 639, 25)
for(i in 1:nrow(k)){
```

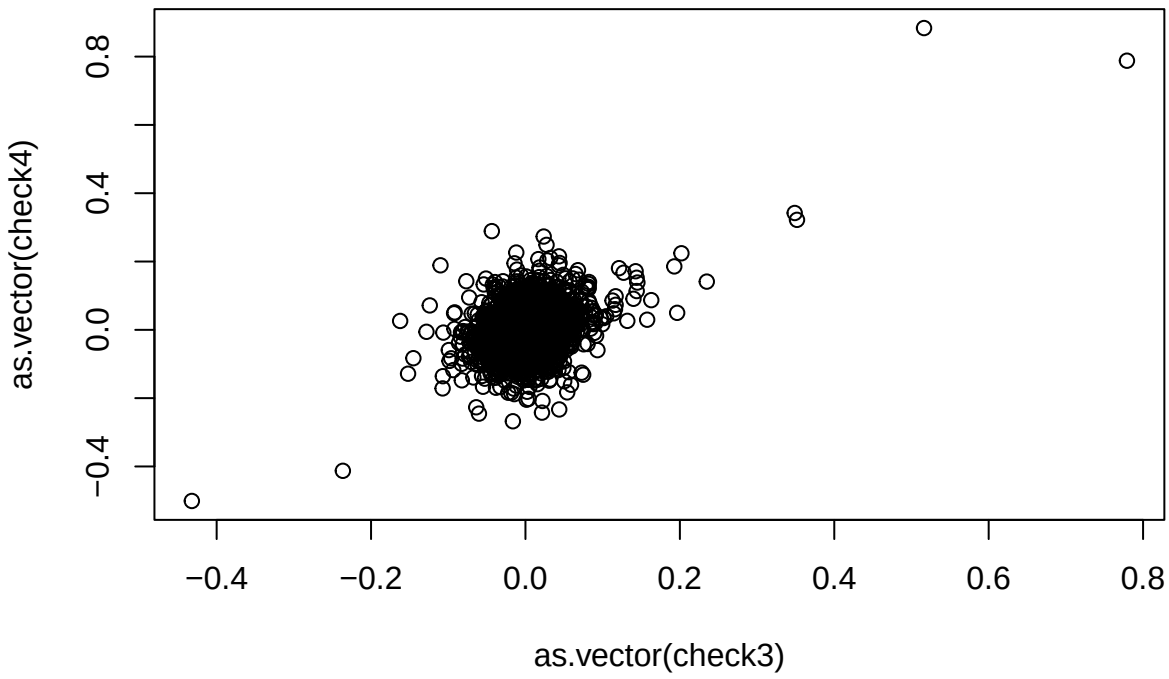
```
check3[, i] <- solve(j, k[i,])
}
```

Using our MLR function on our training sets to produce a check result.

```
j <- validation_testRNA %*% t(validation_testRNA)
k <- validation_testADT %*% t(validation_testRNA)
check4 <- matrix(0, 639, 25)
for(i in 1:nrow(k)){


check4[, i] <- solve(j, k[i,])
}
```

Using our MLR function except this time on our test sets to produce the check result.

```
plot(as.vector(check3), as.vector(check4))
```



In here we see that we have some correlation between the checks implying some success but still not great performance. This could be explained by the different sizes in the test and train for validation.

```
test_set <- t(mat) %*% test_rna
```

Multiply mat as our x variable to our test RNA set to produce the best result for the test ADT set that will be used for the kaggle submission. This was usually around 80% accurate so it produces better results compared to SVD but not as well compared to MLR with correlation filtered data.

```
sample_submission <- reshape2::melt(filtered_submit)
head(sample_submission)
```

```
##   Var1          Var2     value
## 1    1 test_sample_1 1.4337014
## 2    2 test_sample_1 2.6265302
## 3    3 test_sample_1 1.5209180
## 4    4 test_sample_1 0.6328582
## 5    5 test_sample_1 2.6861123
## 6    6 test_sample_1 1.3398805
```

Melting the test submission to transform it into a way that we can submit on kaggle.

```
sample_submission <- data.frame(
  "ID" = paste0("ID_", 1:nrow(sample_submission)),
  "Expected" = sample_submission$value)
head(sample_submission)
```

```
##     ID  Expected
## 1 ID_1 1.4337014
## 2 ID_2 2.6265302
## 3 ID_3 1.5209180
## 4 ID_4 0.6328582
## 5 ID_5 2.6861123
## 6 ID_6 1.3398805
```

We modify our data to have ID's and expected values for our submission so that it can be accepted into kaggle and properly ranked in the leader boards.

```
write.csv(sample_submission, "filtered_with_cor.csv", row.names = FALSE)
```

Lastly, we write our submission into a CSV file so that it can be submitted.

This project was challenging the way that the baseline of unprocessed data and using simple MLR proved to be better than tuning the hyperparamaters of other of other methods of SVD. SVD did show promise producing results around 78% but still could not break 80% accuracy of using MLR with no pre-processing techniques. Several iterations of SVD were used with little to no success besides changing the K value. PCA was attempted but without a package in R, proved to be too difficult to code in. The most successful process was using correlation to filter out uncorrelated proteins and using MLR, this was achieved with guidance from classmates and the professor. In the below section, is commented out attempts of the previously mentioned algorithms.

Tried equations and methods, this is meant to demonstrate effort that I have put into this project and other methods I have tried using SVD and MLR.

```
# diagna <- Matrix::Diagonal(x = SVD_train$d)
#
# library(irlba)
```

```
# SVD_train <- irlba(A, 10)
# #PCA_model <- prcomp_irlba(train_rna, n = 10, scale = TRUE)
#
# estimate_SVD_train <- SVD_train$u %*% diagna %*% t(SVD_train$v)
# estimate_SVD_train2 <- estimate_SVD_train[1:664, 1:500]
# estimate_SVD_train3 <- as.matrix(estimate_SVD_train2@x)
#
#
# # normal equation for finding A_{testRNA} = U_{train}V_{test}
#
# # solve for V_{test}:
# a <- t(estimate_SVD_train2[26:664,]) %*% estimate_SVD_train2[26:664,]
# b <- t(test_rna) %*% estimate_SVD_train2[26:664, ]
# V_test <- apply(b, 1, function(b0) solve(a, b0))
# ADT_test <- estimate_SVD_train3[1:25, ] %*% as.matrix(V_test)
```

The attempt here was to use the U and V matrix provided in SVD as the ones to use in MLR, we attempt to first create the a and b matrix and then apply that over the estimate_svd_train2. This proved to be difficult as R errors were common and ulitmately scrapped.

```
# library(irlba)
# SVD_train <- irlba(A, 10)
# PCA_model <- prcomp_irlba(train_rna, n = 10, scale = TRUE)
#
# estimate_SVD_train <- SVD_train$u %*% Matrix::Diagonal(x = SVD_train$d) %*% t(SVD_train$v)
#
#
#  normal equation for finding A_{testRNA} = U_{train}V_{test}
#
#  solve for V_{test}:
# a <- t(SVD_train$u[26:664,]) %*% SVD_train$u[26:664,]
# b <- t(test_rna) %*% SVD_train$u[26:664, ]
# V_test <- apply(b, 1, function(b0) solve(a, b0))
# ADT_test <- SVD_train$u[1:25, ] %*% V_test
```

In this chunk we use the diagonal for utilization in the equation, this ultimately proved to be harder than expected to extract the diagonal for utilization. This attempt to include it in the SVD equation was noted but ultimately scrapped.

```
# library(dplyr)
# prin_trainrna <- prcomp(train_rna, scale. = TRUE, center = TRUE)
# random_object <- read.csv("training_set_adt.csv")
# fact_protein <- random_object[, 1] %>%
#   factor(., levels = unique(random_object[, 1]))
```

In this chunk, the attempt is to create a PCA model using the function prcomp() and separate the proteins from the training set of ADTs for a later use.

```
# eig <- prin_trainrna$sdev^2
# eig
# summary(prin_trainrna)
```

In this chunk we are separating the eigen values for a later use in making a scree plot. This was successful but did not work in the next chunk. The standard deviation is squared for the eigen value and printed for validation.

```
# loading_Scores_PC_1 <- prin_trainrna$rotation[, 1]
# fac_scores_PC_1 <- abs(loading_Scores_PC_1)
# fac_scores_PC_1_ranked <- names(sort(fac_scores_PC_1,decreasing = T))
# prin_trainrna$rotation[fac_scores_PC_1_ranked,1]
# loading_Scores_PC_2 <- prin_trainrna$rotation[, 2]
# fac_scores_PC_2 <- abs(loading_Scores_PC_2)
# fac_scores_PC_2_ranked <- names(sort(fac_scores_PC_2,decreasing = T))
# prin_trainrna$rotation[fac_scores_PC_2_ranked,2]
```

This is the chunk for attempting to load in the eigen values for the production of a scree plot to determine a good K value for PCA. The errors here were R errors and around the grammar of R. This was troubleshot for a while but was ultimately scrapped.

```
# plot(prin_trainrna, type = "l", main ="Scree plot for PCA")
```

This is an attempt to plot the scree plot. The issue was the R grammar.

```
# scores <- data.frame(random_object, prin_trainrna$X[1:2])
#
# plot_2 <-ggplot(scores,aes(x=PC1,y=PC2,color=fact_protein)) + geom_point(size =2) + labs(title="Plott
#
# print(plot_2)
#
# predict(scores)
```

This time we tried to separate the PCA the X value from the PCA chunk and store that in an object scores. There were attempts to specify plots like geom_point, and also including predict scores for validation.