Music Genre Identification Report

Table of Contents

I. Overview

Spotify is one of the most popular streaming services with over 20 million songs and thousands of songs uploaded every day. This project aims to automate the process of classifying some genres of songs (Rock, Electronic, Hip-hop, Pop) on Spotify based on musical features of the audio. Using linear discriminant analysis (LDA) and quadratic discriminant analysis (QDA) methods, our two main questions include:

1. If we are limited to a single feature to decide the genre, what decisions should we make and how good will our classifications be?
2. If we can consider multiple features, what decisions should we make and how good will our classifications be? How many features do you need to adequately classify genres?

II. Data Preparation and Wrangling

1. Data Wrangling

We start with loading the two data sets, `feature_csv` and `tracks.csv`.

Since the `tracks.csv` contains data of other genres, we filter out the 4 genres we aim to classify, Hip-hop, Rock, Pop and Electronic, and select only the two columns we need, `track_id` and `track_genre_top`.

We then merge the two data sets.

```
#Load the datasets
feature_df <- read.csv('/Data/bonifontea/da350/features.csv')
tracks_df <- read.csv('/Data/bonifontea/da350/tracks.csv')
```

```
#filter tracks_df to get only 4 genres
tracks_chosen_df <- tracks_df %>%
  filter(track_genre_top == 'Pop' | track_genre_top == 'Electronic' |
track_genre_top == 'Hip-Hop' | track_genre_top == 'Rock') %>%
  select(track_id, track_genre_top)

main_df <- merge(x = feature_df, y = tracks_chosen_df, by = 'track_id', all.x
= TRUE)
main_df <- na.omit(main_df)

#Change data types to suitable types
main_df$track_id <- as.factor(main_df$track_id)
main_df$track_genre_top <- as.factor(main_df$track_genre_top)
```

2. Create training and test sets

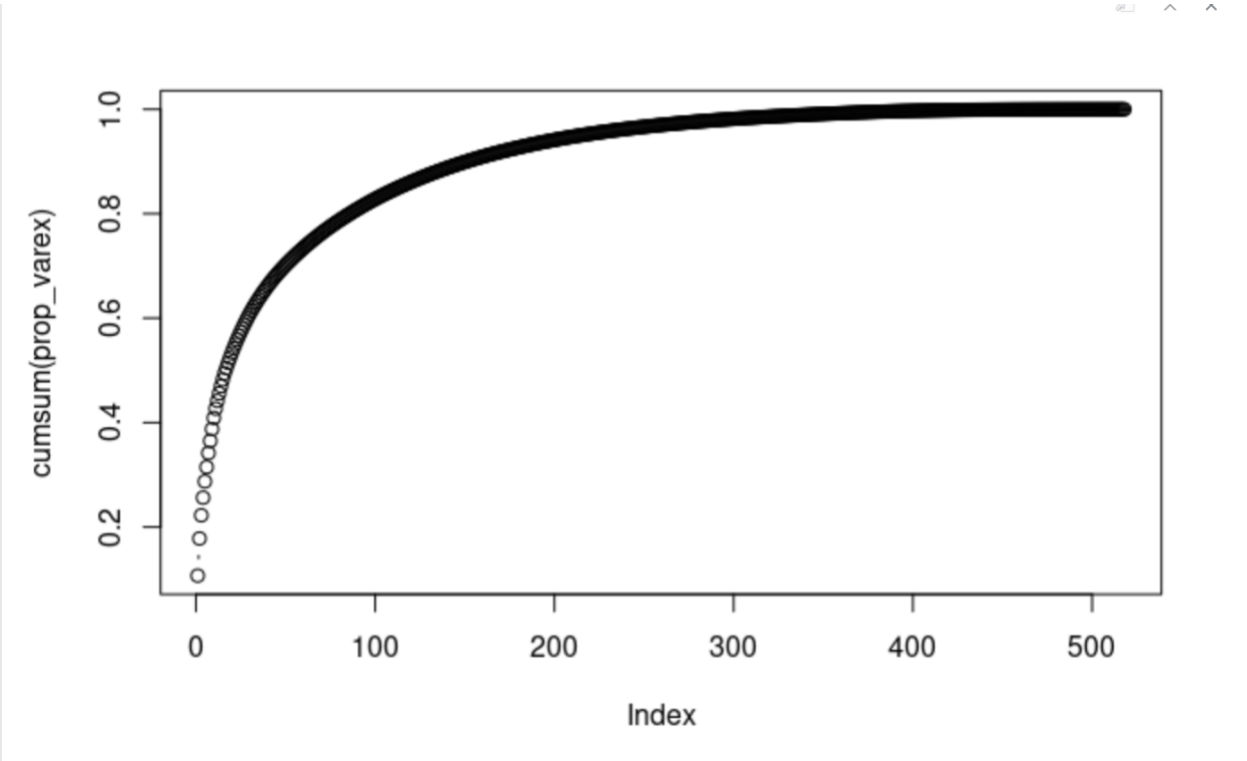
We use 70% of the original data for the training set and 30% for test set. We leave the test set for model evaluation at the end of the report.

```
set.seed(1)
index = createDataPartition(y = main_df$track_genre_top, p = 0.7, list =
FALSE)
train_df = main_df[index,]
rownames(train_df) = train_df$track_id
test_df = main_df[-index,]
rownames(test_df) = test_df$track_id
```

3. Dimension Reduction using Principle Component Analysis (PCA)

Since LDA works better with smaller samples. we reduce the dimension of the data set.

```
train.pca <- prcomp(train_df[,2:(ncol(train_df)-1)], scale. = TRUE)

pca_var = train.pca$sdev^2
prop_varex = pca_var/sum(pca_var)
plot(cumsum(prop_varex), type = 'b')

cumsum(prop_varex)[343]
```



We decide on choosing 343 principal components as it explains more than 90% of the variation in the data set.

```
#create a data frame from pca
train.pca_df <- data.frame(train.pca$x[,1:343])
train.pca_df$genre <- train_df$track_genre_top
```

III. General Model

1. LDA with all variables

```
ctrl = trainControl(method = 'repeatedcv', number = 10, repeats = 5)

lda.fit_no_pca = train(track_genre_top ~., data =
train_df[,2:ncol(train_df)], method = 'lda', trControl = ctrl)

lda.fit_no_pca$results #Returns training accuracy and Kappa of classification
```

	parameter <chr>	Accuracy <dbl>	Kappa <dbl>	AccuracySD <dbl>	KappaSD <dbl>
1	none	0.7787901	0.6511588	0.01061962	0.01646179

With repeated cross validation, the Accuracy score is 0.7787 and Kappa is 0.6511.

2. LDA with Principal Components (PCs)

We run LDA with 343 principal components.

```
lda.fit = train(genre ~., data = train.pca_df, method = 'lda', trControl =
ctrl)

lda.fit$results #Returns training accuracy and Kappa of classification
```

	parameter <chr>	Accuracy <dbl>	Kappa <dbl>	AccuracySD <dbl>	KappaSD <dbl>
1	none	0.7770808	0.6475138	0.007368067	0.01140765

With repeated cross validation, the Accuracy score is 0.777 and Kappa is 0.6475.

3. QDA with PCs

```
qda.fit = train(genre ~., data = train.pca_df, method = 'qda', trControl =
ctrl)

qda.fit$results #Returns training accuracy and Kappa of classification
```

	parameter <chr>	Accuracy <dbl>	Kappa <dbl>	AccuracySD <dbl>	KappaSD <dbl>
1	none	0.5509485	0.3493256	0.01207655	0.0131621

With repeated cross validation, the Accuracy score is 0.5509 and Kappa is 0.3493.

It's clear that LDA works better in this situation because we only have a small data set. We will continue our analysis with LDA.

4. More metrics and plots

As the goal is to choose specific features and the run time is not too bad, we will continue with all variables.

```
confusionMatrix(predict(lda.fit_no_pca, train_df[,1:ncol(train_df)]),
train_df$track_genre_top)
lda.fit_no_pca$finalModel$counts
```

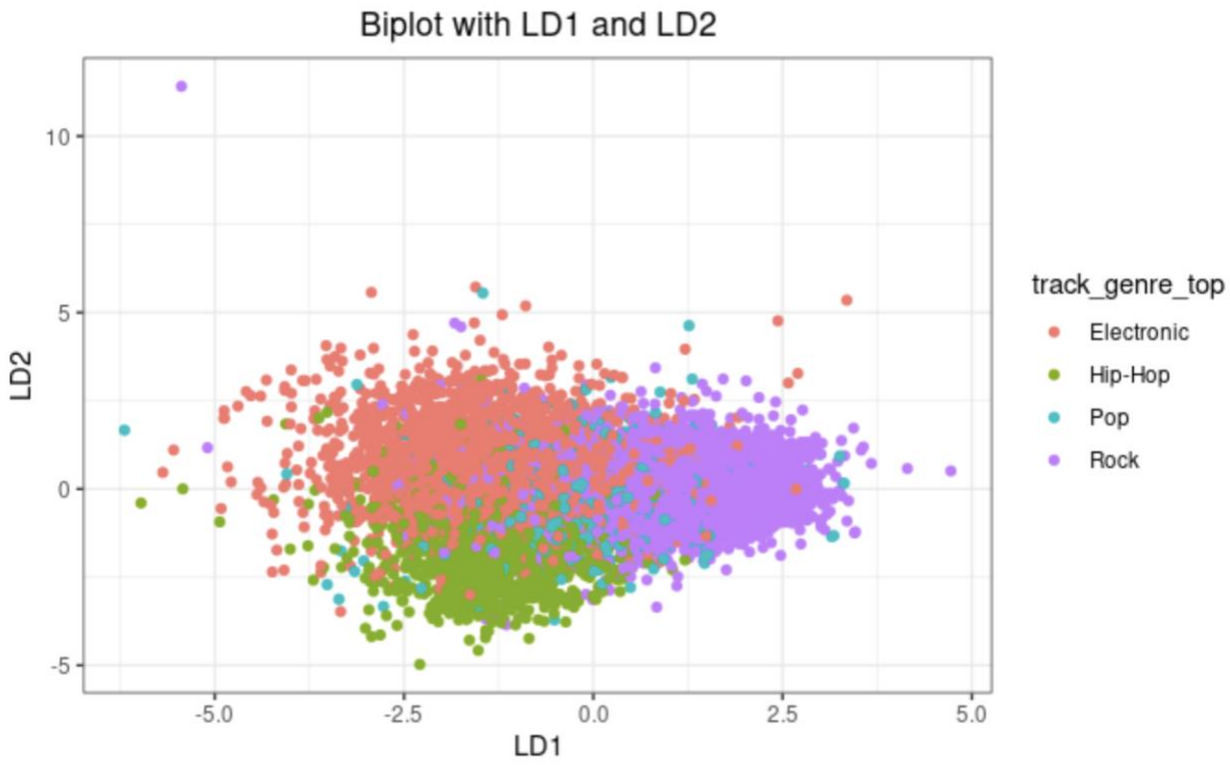
```
Confusion Matrix and Statistics

            Reference
Prediction   Electronic Hip-Hop  Pop Rock
  Electronic       5233     506  372  556
  Hip-Hop           383    1762  124  161
  Pop               256      70  481  247
  Rock              689     149  656 8958

Overall Statistics

               Accuracy : 0.7977
                 95% CI : (0.7921, 0.8031)
    No Information Rate : 0.4816
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.681

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: Electronic Class: Hip-Hop Class: Pop Class: Rock
Sensitivity                     0.7976        0.70848    0.29455      0.9028
Specificity                     0.8979        0.96313    0.96979      0.8601
Pos Pred Value                  0.7849        0.72510    0.45636      0.8571
Neg Pred Value                  0.9047        0.96011    0.94107      0.9050
Prevalence                      0.3184        0.12071    0.07926      0.4816
Detection Rate                  0.2540        0.08552    0.02335      0.4348
Detection Prevalence            0.3236        0.11794    0.05116      0.5073
Balanced Accuracy               0.8477        0.83581    0.63217      0.8815
Electronic    Hip-Hop        Pop       Rock
      6561       2487       1633       9922
```

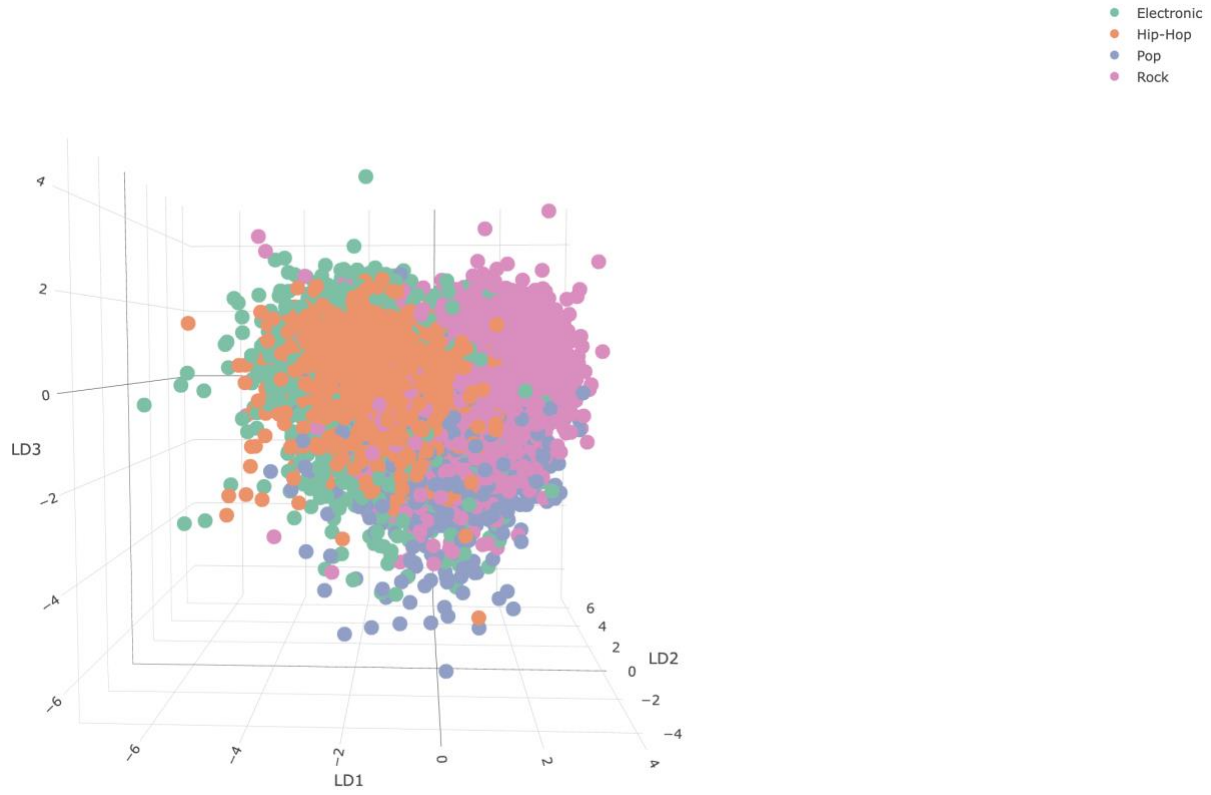
Above is the confusion matrix using LDA with all variables. The Accuracy score without cross validation is 0.7977. This is quite good given the Kappa of 0.681, indicating that the model is much better than random guessing. However, the Sensitivity (True Positive rate) of Pop is a little bit low compared to the Sensitivity (True Negative rate) of other genres. This may happen because of class imbalance. We would pay attention to this and try to use some methods to handle class imbalance later.

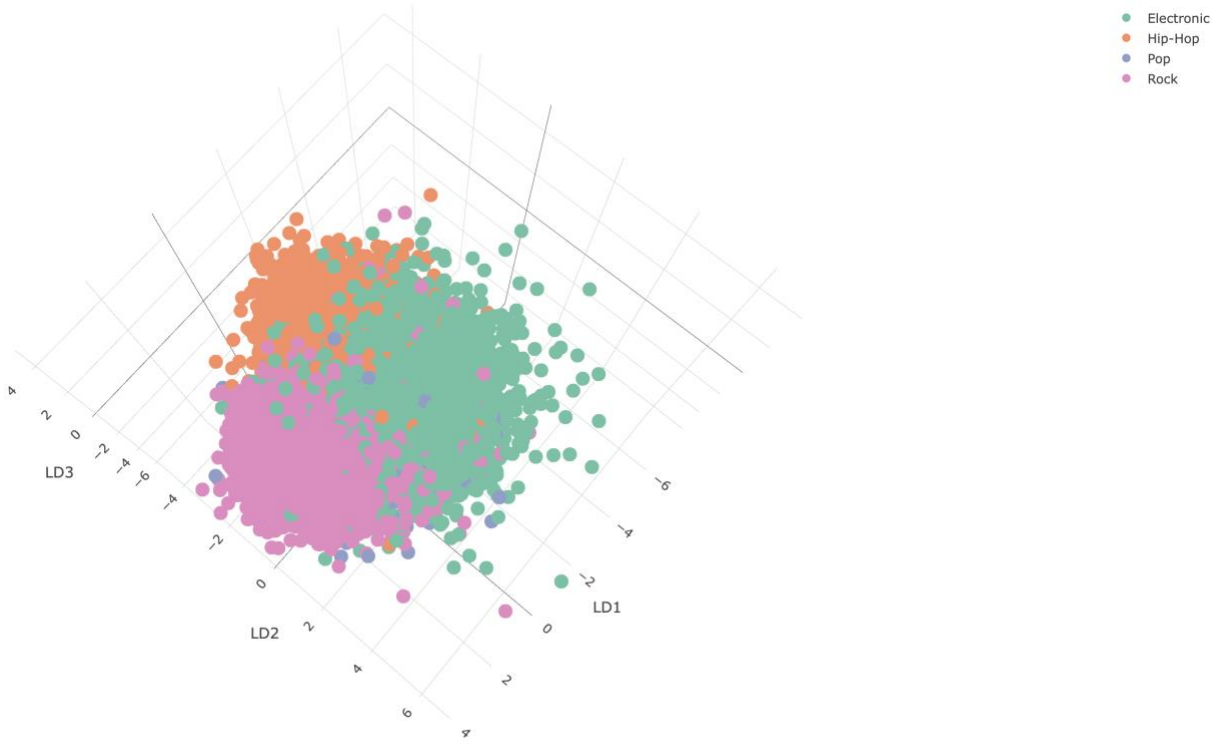
We plot the classification in 2d and 3d.

```
#run lda again with MASS because it's easier to extract the coefficients
#the result is the same
lda_by_MASS<- lda(track_genre_top ~., data = train_df[,2:ncol(train_df)])
```

```
predict.lda <- predict(lda_by_MASS,train_df)

lda_plot <- cbind(train_df,predict.lda$x)
ggplot(lda_plot, aes(LD1,LD2)) +
  geom_point(aes(color = track_genre_top)) +
  labs(title = "Biplot with LD1 and LD2") +
  theme_bw() +
  theme(plot.title = element_text(hjust=0.5))

plot_ly(lda_plot, x = ~LD1, y = ~LD2, z = ~LD3, color = ~track_genre_top) %>%
  add_markers() %>%
  layout(scene = list(xaxis = list(title = 'LD1'),
                      yaxis = list(title = 'LD2'),
                      zaxis = list(title = 'LD3')))
```




Electronic
Hip-Hop
Pop
Rock
LD3
LD1
LD2


Electronic
Hip-Hop
Pop
Rock
LD3
LD2
LD1

The plots clearly show that `Electronic`, `Hip Hop`, and `Rock` are somewhat separated from each other. In contrary, `Pop` is mixed with other genres which goes along with its low Sensitivity.

IV. Find the most important feature

1. Run `varImp`

The `varImp` function tracks the changes in model statistics for each variable to find out the importance of each. It depends on the model for which statistics it uses. With LDA, the importance is estimated using a ROC curve analysis.

```
importance_value <- varImp(lda.fit_no_pca, scale = FALSE)
print(importance_value)
importance_df <- importance_value$importance
```

	Electronic <dbl>	Hip.Hop <dbl>	Pop <dbl>	Rock <dbl>
mfcc_median_3	0.6552550	0.8384901	0.6006888	0.8384901
mfcc_mean_3	0.6501168	0.8321000	0.6064392	0.8321000
mfcc_median_20	0.7137438	0.7835526	0.6653369	0.7835526
spectral_centroid_...	0.6559855	0.7828758	0.7065884	0.7828758
mfcc_mean_20	0.7112418	0.7809503	0.6591585	0.7809503
mfcc_std_15	0.5723172	0.7809105	0.6211898	0.7809105
mfcc_std_14	0.5770306	0.7762736	0.6423154	0.7762736
mfcc_std_16	0.5774851	0.7751650	0.6172763	0.7751650
spectral_rolloff_st...	0.6503551	0.7702088	0.6640152	0.7702088
mfcc_skew_3	0.6246969	0.7701624	0.6279835	0.7701624

The most important variable is `mfcc_median_3`. We will validate this by fitting LDA with `mfcc_median_3` and `mfcc_mean_20`, the fifth important variable, to see if there are any differences in the result.

```
lda.fit_important = train(track_genre_top ~ mfcc_median_3, data =
train_df[,2:ncol(train_df)], method = 'lda', trControl = ctrl)
lda.fit_important2 = train(track_genre_top ~ mfcc_median_20, data =
train_df[,2:ncol(train_df)], method = 'lda', trControl = ctrl)

lda.fit_important$results
lda.fit_important2$results
```

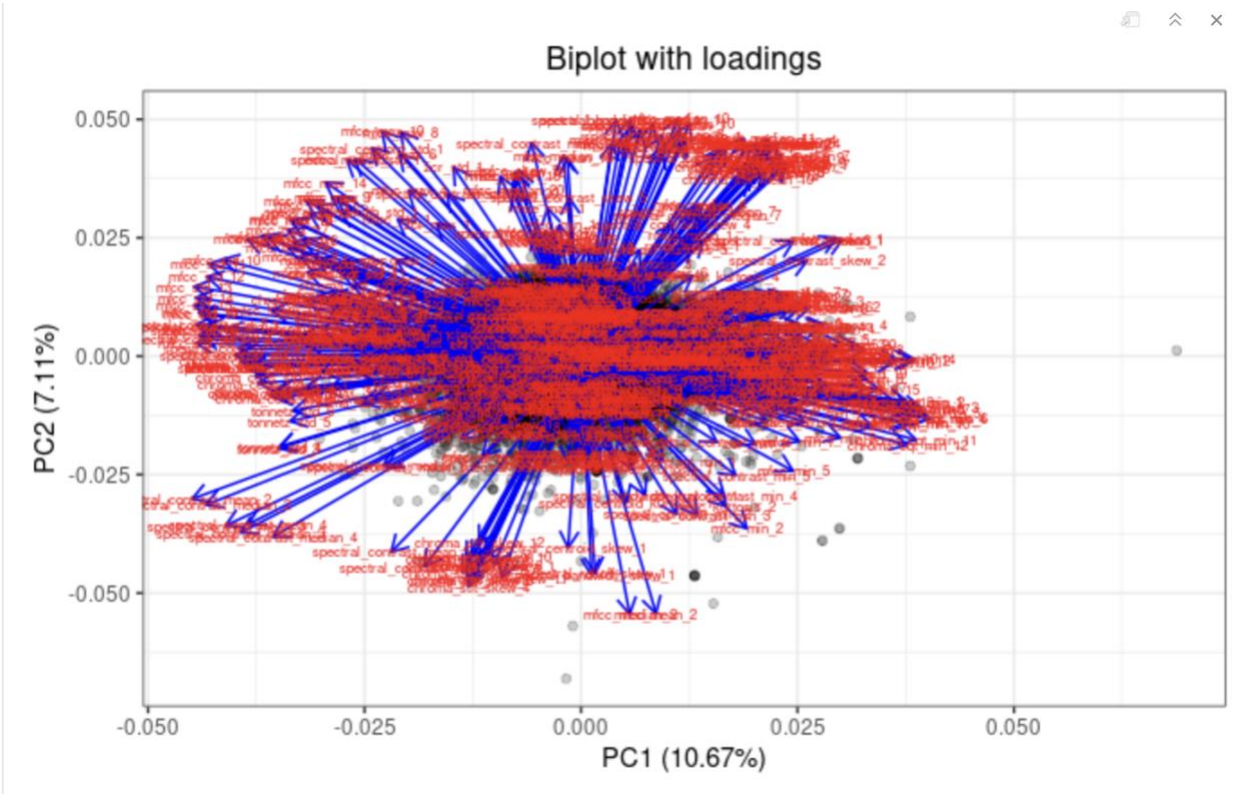
	parameter <chr>	Accuracy <dbl>	Kappa <dbl>	AccuracySD <dbl>	KappaSD <dbl>
1	none	0.6159977	0.3353169	0.006368475	0.01127824

	parameter <chr>	Accuracy <dbl>	Kappa <dbl>	AccuracySD <dbl>	KappaSD <dbl>
1	none	0.5832654	0.2706259	0.009082992	0.01609036

With cross validation, the result shows that using `mfcc_median_3` achieves higher Accuracy and Kappa score than using `mfcc_mean_20`. This is consistent with the result generated by the varImp() function. We would try finding the most important variables based on the loading vectors in PCA and compare with the metrics here to see which one is better.

2. Using PCs

```
autoplot(train.pca, loadings = TRUE, loadings.colour = 'blue',
         loadings.label = TRUE, loadings.label.size = 2, alpha= 0.2) +
  labs(title = "Biplot with loadings") +
  theme_bw() +
  theme(plot.title = element_text(hjust=0.5))
```



It's hard to see which variables has high loading values in this plot. Since the first principal component captures the largest variance, we would take the its loading vector.

```
#create a data frame of the loading vector of the first principal component
pc1_df <- data.frame(loading = train.pca$rotation[,1])
head(arrange(pc1_df, loading))
head(arrange(pc1_df, desc(loading)))
```

	loading <dbl>
spectral_contrast_...	-0.09210346
mfcc_std_13	-0.09163242
mfcc_std_15	-0.09162643
mfcc_std_14	-0.09124777
mfcc_std_20	-0.09122710
spectral_contrast_...	-0.09113914

	loading <dbl>
chroma_cqt_min_6	0.08272852
chroma_cqt_min_4	0.08232194
chroma_cqt_min_5	0.08123402
chroma_cqt_min_3	0.08059237
chroma_cqt_min_11	0.07934045
mfcc_min_14	0.07909072

`special_contrast_mean_2` has the lowest loading value and `chroma_cqt_min_6` has the highest loading value. We would try to fit LDA on each of the variables.

```
lda.fit_important3 = train(track_genre_top ~ chroma_cqt_min_6, data =
train_df[,2:ncol(train_df)], method = 'lda', trControl = ctrl)

lda.fit_important4 = train(track_genre_top ~ spectral_contrast_mean_2 +
chroma_cqt_min_6, data = train_df[,2:ncol(train_df)], method = 'lda',
trControl = ctrl)

lda.fit_important3$results
lda.fit_important4$results
```

	parameter <chr>	Accuracy <dbl>	Kappa <dbl>	AccuracySD <dbl>	KappaSD <dbl>
1	none	0.5093144	0.1064663	0.007837433	0.01397362

	parameter <chr>	Accuracy <dbl>	Kappa <dbl>	AccuracySD <dbl>	KappaSD <dbl>
1	none	0.5122458	0.1208009	0.007287785	0.01260208

With cross validation, the Accuracy scores of `chroma_cqt_min_6` and `special_contrast_mean_2` are 0.5093 and 0.5122. The Kappa scores are low (0.106 and 0.12). Clearly, the `varImp` function does a better job.

3. More Metrics and Visualizations using LDA with only `mfcc_median_3`

So far, `mfcc_median_3` does the best job with the highest Accuracy and Kappa score. We would take a look at its confusion matrix.

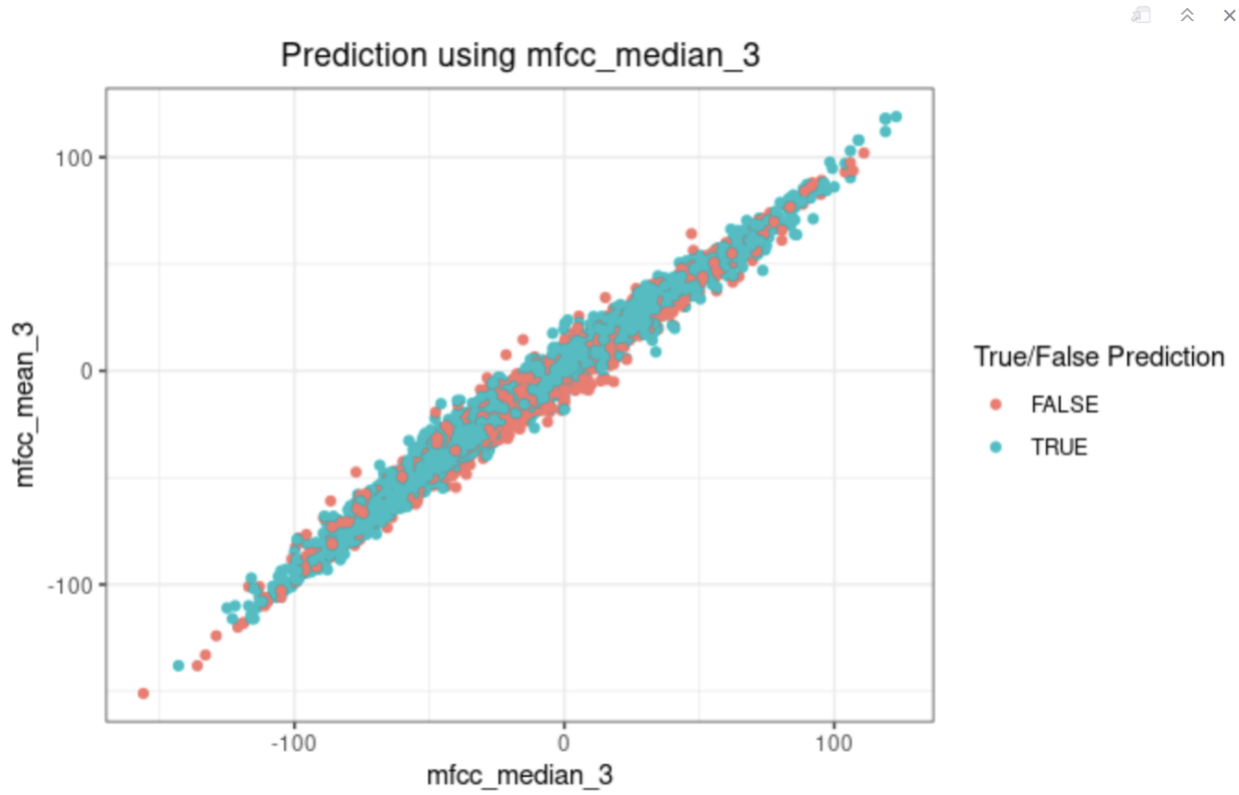
```
confusionMatrix(predict(lda.fit_important,train_df[,2:ncol(train_df)]),
train_df$track_genre_top)
```

```
Confusion Matrix and Statistics

            Reference
Prediction   Electronic Hip-Hop  Pop Rock
  Electronic       4174    1253  659 1403
  Hip-Hop             0       0    0    0
  Pop                 0       0    0    0
  Rock             2387    1234  974 8519

Overall Statistics

               Accuracy : 0.6161
                 95% CI : (0.6094, 0.6227)
    No Information Rate : 0.4816
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3354

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Electronic Class: Hip-Hop Class: Pop Class: Rock
Sensitivity                     0.6362         0.0000    0.00000      0.8586
Specificity                     0.7639         1.0000    1.00000      0.5698
Pos Pred Value                  0.5574            NaN        NaN      0.6496
Neg Pred Value                  0.8180         0.8793    0.92074      0.8127
Prevalence                      0.3184         0.1207    0.07926      0.4816
Detection Rate                  0.2026         0.0000    0.00000      0.4135
Detection Prevalence            0.3635         0.0000    0.00000      0.6365
Balanced Accuracy               0.7001         0.5000    0.50000      0.7142
```

The Accuracy score without cross validation is 0.6161, and the Kappa score is 0.3354. The Kappa score is lower than the Kappa score when using LDA on all variables. This means that the model is doing better than random guessing but still not as good as the previous model. With just one variable, the classes are all predicted as `Electronic` and `Rock`. This happens because of the class imbalance leading to the low prior probabilities of `Hip Hop` and `Pop`. The Sensitivity scrores of `Hip Hop` and `Pop` are, therefore, 0, while the Sensitivity scores of `Electronic` and `Rock` is quite good, 0.6362 and 0.8586 respectively. The Specificity scores are great for all classes.

```
#plot based on most important using varImp
important.lda <- predict(lda.fit_important, train_df)
predict_status <- important.lda == train_df$track_genre_top

ggplot(train_df) +
  geom_point(aes(mfcc_median_3, mfcc_mean_3, color = important.lda ==
```

```
train_df$track_genre_top)) +
  labs(title = "Prediction using mfcc_median_3", colour = 'True/False
Prediction') +
  theme_bw() +
  theme(plot.title = element_text(hjust=0.5))
```



The plot above shows whether a prediction is True or False. We can see clearly that there is a handful of false predictions.

V. Find multiple important features

1. LDA with top 20 most important features

We start with the top 20 most important features suggested by the `varImp` function to see how the performance metrics look like.

```
lda.fit_top20 = train(track_genre_top ~ mfcc_median_3 + mfcc_mean_3 +
mfcc_median_20 + spectral_centroid_std_1 + mfcc_mean_20 + mfcc_std_15 +
mfcc_std_14 + mfcc_std_16 + spectral_rolloff_std_1 + mfcc_skew_3 +
mfcc_std_17 + mfcc_std_13 + spectral_contrast_std_4 + mfcc_max_16 +
mfcc_skew_1 + mfcc_median_18 + mfcc_std_11 + mfcc_std_18 + mfcc_mean_18 +
mfcc_std_12, data = train_df[,2:ncol(train_df)], method = 'lda', trControl =
ctrl)

lda.fit_important2$results
```

Description: df [1 × 5]

	parameter <chr>	Accuracy <dbl>	Kappa <dbl>	AccuracySD <dbl>	KappaSD <dbl>
1	none	0.5832654	0.2706259	0.009082992	0.01609036

With cross validation, the Accuracy score is 0.5832 and the Kappa score is 0.27. Those are quite low even compared to the model with only one variable. Fitting LDA with about more variables may be a better idea because of the number of PCs we choose (343).

2. Find a reasonable number of features

We try out top 350, 300, 250, 200, 150 most important features. The `varImp` function only returns top 20 most important features. For other features, it returns an unsorted data frame with variable importance score for each class. The way `varImp` ranks the importance based on these scores is to sum them up and add some extra statistics to them. To replicate that, we would just get a total of the importance scores for each variable.

```
importance_df <- data.frame(importance_value$importance) %>%
  mutate(total = Hip.Hop + Electronic + Pop + Rock) %>%
  arrange(desc(total))
head(importance_df,10)
```

	Electronic <dbl>	Hip.Hop <dbl>	Pop <dbl>	Rock <dbl>	total <dbl>
mfcc_median_20	0.7137438	0.7835526	0.6653369	0.7835526	2.946186
mfcc_median_3	0.6552550	0.8384901	0.6006888	0.8384901	2.932924
mfcc_mean_20	0.7112418	0.7809503	0.6591585	0.7809503	2.932301
spectral_centr...	0.6559855	0.7828758	0.7065884	0.7828758	2.928325
mfcc_mean_3	0.6501168	0.8321000	0.6064392	0.8321000	2.920756
mfcc_median_18	0.6867671	0.7550274	0.6974697	0.7550274	2.894292
mfcc_mean_18	0.6831988	0.7507366	0.6969302	0.7507366	2.881602
mfcc_max_16	0.6754972	0.7562330	0.6837182	0.7562330	2.871681
spectral_rollof...	0.6503551	0.7702088	0.6640152	0.7702088	2.854788
mfcc_max_10	0.6825164	0.7278304	0.7103247	0.7278304	2.848502

1–10 of 10 rows

The result is quite similar compared to the top 10 given by `varImp`. The order is slightly different but that doesn't matter for us since we are planning to choose more variables and the variables are treated with the same weight when used for LDA.

```
train_df_most_imp <- train_df %>%
  select('track_genre_top',rownames(importance_df[1:350,]))

lda.fit_top350 = train(track_genre_top ~ ., data = train_df_most_imp, method
= 'lda', trControl = ctrl)
lda.fit_top300 = train(track_genre_top ~ ., data = train_df_most_imp[,1:301],
method = 'lda', trControl = ctrl)
```

```
lda.fit_top250 = train(track_genre_top ~ ., data = train_df_most_imp[,1:251],
method = 'lda', trControl = ctrl)
lda.fit_top200 = train(track_genre_top ~ ., data = train_df_most_imp[,1:201],
method = 'lda', trControl = ctrl)
lda.fit_top150 = train(track_genre_top ~ ., data = train_df_most_imp[,1:151],
method = 'lda', trControl = ctrl)

lda.fit_top350$results
lda.fit_top300$results
lda.fit_top250$results
lda.fit_top200$results
lda.fit_top150$results
```

	parameter <chr>	Accuracy <dbl>	Kappa <dbl>	AccuracySD <dbl>	KappaSD <dbl>
1	none	0.7771878	0.6482034	0.007794546	0.01224352

	parameter <chr>	Accuracy <dbl>	Kappa <dbl>	AccuracySD <dbl>	KappaSD <dbl>
1	none	0.7753238	0.6449703	0.007441176	0.01169693

	parameter <chr>	Accuracy <dbl>	Kappa <dbl>	AccuracySD <dbl>	KappaSD <dbl>
1	none	0.7712857	0.6374814	0.007155133	0.01112403

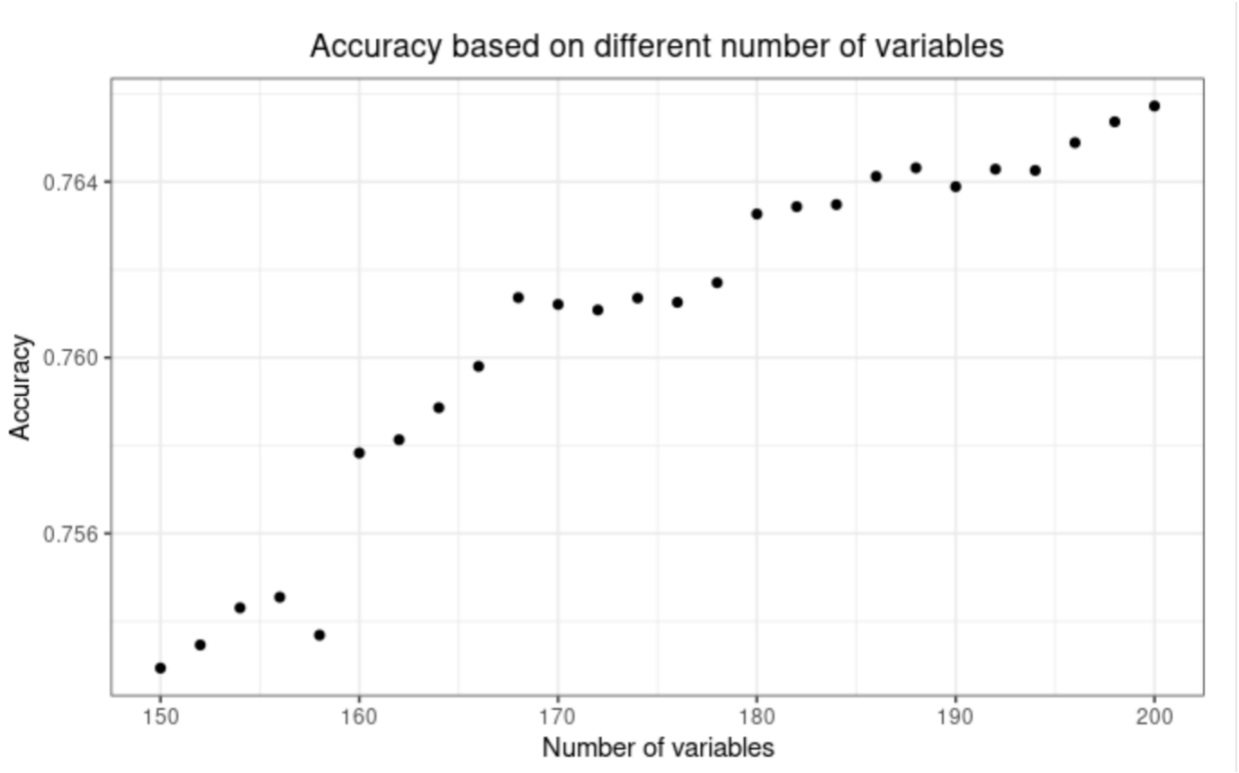
	parameter <chr>	Accuracy <dbl>	Kappa <dbl>	AccuracySD <dbl>	KappaSD <dbl>
1	none	0.7653349	0.627234	0.00810317	0.01287844

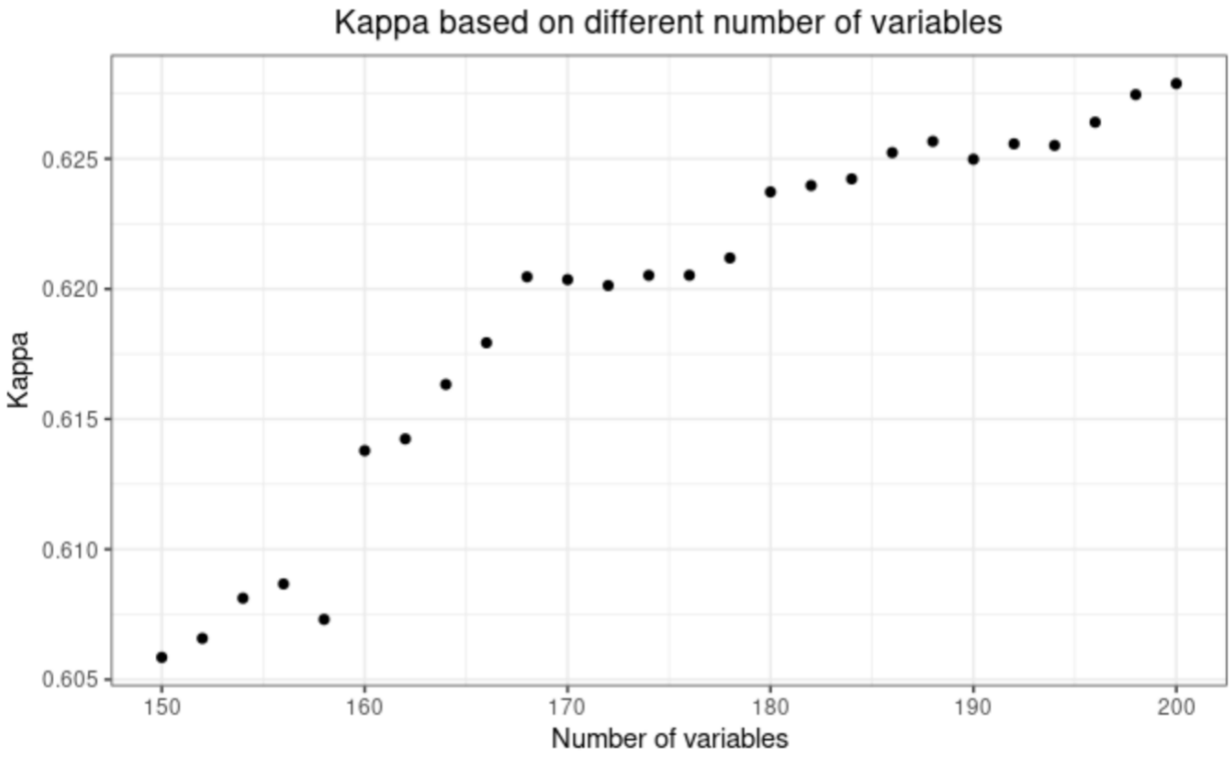
	parameter <chr>	Accuracy <dbl>	Kappa <dbl>	AccuracySD <dbl>	KappaSD <dbl>
1	none	0.7534343	0.6066198	0.007384651	0.01144854

With cross validation, the result generated by the top 350 and 300 are almost the same as the result using all variables. There are noticeable differences in the result of the top 200 and top 150 so there may be a point where there is a significant improvement. We would keep top 300 under consideration because given enough time and computational resources, we would definitely want such a good result. We would further explore from top 150 to 200 by fitting LDA on that range with a step of 2.

```
start = Sys.time()
lda.fit_all_important <- lapply(seq(150,200,2), function(x)
train(track_genre_top ~ ., data = train_df_most_imp[,1:(x+1)], method =
'lda', trControl = ctrl))
end = Sys.time()
end-start

lda_imp_df <- data.frame(num_of_var = seq(150,200,2), accuracy =
unlist(lapply(1:26, function(x)
```

```
lda.fit_all_important[[x]]$results$Accuracy)), kappa = unlist(lapply(1:26,
function(x) lda.fit_all_important[[x]]$results$Kappa)))

ggplot(lda_imp_df) +
  geom_point(aes(num_of_var,accuracy)) +
  labs(title = "Accuracy based on different number of variables", x = 'Number
of variables',y = 'Accuracy') +
  theme_bw() +
  theme(plot.title = element_text(hjust=0.5))


ggplot(lda_imp_df) +
  geom_point(aes(num_of_var,kappa)) +
  labs(title = "Kappa based on different number of variables", x = 'Number of
variables',y = 'Kappa') +
  theme_bw() +
  theme(plot.title = element_text(hjust=0.5))
lda_by_MASS<- lda(track_genre_top ~ ., data = train_df_most_imp[,1:161])

predict.lda <- predict(lda_by_MASS,train_df)

lda_plot <- cbind(train_df,predict.lda$x)
ggplot(lda_plot, aes(LD1,LD2)) +
  geom_point(aes(color = track_genre_top))
```





With cross validation, fitting LDA on top 160 most important features yield a significant improvement. We would choose this number of features to continue working on. However, we would still keep top 300 most important features in consideration.

3. More metrics and plots for LDA with top 160

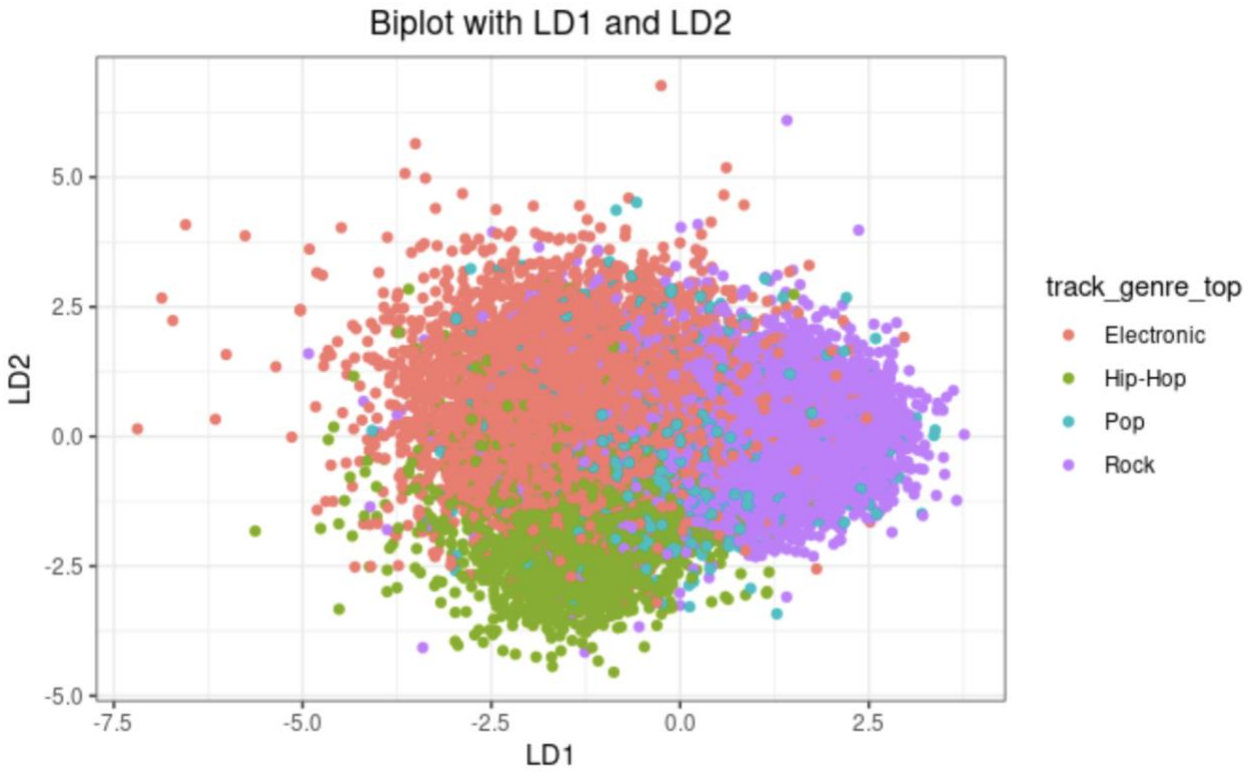
```
lda.fit_top160 = train(track_genre_top ~ ., data = train_df_most_imp[,1:161],
method = 'lda', trControl = ctrl)

lda.fit_top160$finalModel$counts

confusionMatrix(predict(lda.fit_top160, train_df_most_imp[,1:161]),
train_df_most_imp[,1:161]$track_genre_top)
```

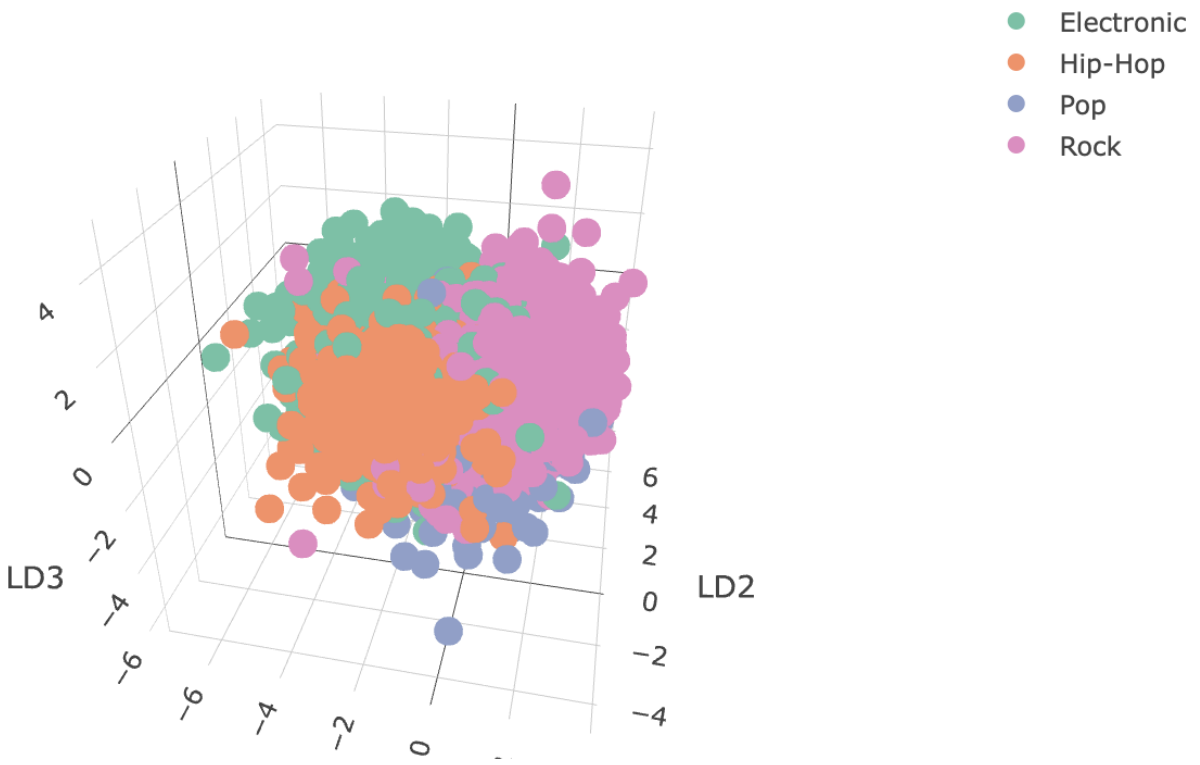
```
Confusion Matrix and Statistics

            Reference
Prediction   Electronic Hip-Hop  Pop Rock
  Electronic       4961     607  407  600
  Hip-Hop           455    1616  125  190
  Pop               212      38  268  236
  Rock              933     226  833 8896

Overall Statistics

               Accuracy : 0.764
                 95% CI : (0.7582, 0.7698)
    No Information Rate : 0.4816
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6236

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: Electronic Class: Hip-Hop Class: Pop Class: Rock
Sensitivity                     0.7561        0.64978    0.16412      0.8966
Specificity                     0.8851        0.95750    0.97438      0.8135
Pos Pred Value                  0.7545        0.67728    0.35544      0.8170
Neg Pred Value                  0.8859        0.95219    0.93123      0.8944
Prevalence                      0.3184        0.12071    0.07926      0.4816
Detection Rate                  0.2408        0.07844    0.01301      0.4318
Detection Prevalence            0.3191        0.11581    0.03660      0.5285
Balanced Accuracy               0.8206        0.80364    0.56925      0.8550
```

Without cross validation, the Accuracy score is 0.764 and the Kappa score is 0.6236. Those are lower than the LDA with all variables, but those reflect a significant improvement. The Sensitivity scores are fine for `Electronic` and `Rock`, but not for `Pop`. The Sensitivity of `Hip Hop` is also a little bit low compared to the other two genres. We mentioned this earlier but haven't used any methods to solve this problem. `Hip Hop` and `Pop` have only 2487 and 1633 observations respectively. We would implement some methods to handle this in the next section.

```
lda_by_MASS2<- lda(track_genre_top ~., data = train_df_most_imp[,1:161])

predict.lda2 <- predict(lda_by_MASS2,train_df_most_imp[,1:161])

lda_plot <- cbind(train_df_most_imp[,1:161],predict.lda2$x)
ggplot(lda_plot, aes(LD1,LD2)) +
```

```
  geom_point(aes(color = track_genre_top)) +
  labs(title = "Biplot with LD1 and LD2") +
  theme_bw() +
  theme(plot.title = element_text(hjust=0.5))

plot_ly(lda_plot, x = ~LD1, y = ~LD2, z = ~LD3, color = ~track_genre_top) %>%
  add_markers() %>%
  layout(scene = list(xaxis = list(title = 'LD1'),
                      yaxis = list(title = 'LD2'),
                      zaxis = list(title = 'LD3')))
```





The plots show that Pop observations don't establish its own area but blend into the areas of other genres. This reflects the misclassification of Pop. Hip Hop observations have its own area but also slightly blend into other genres' areas.

VI. Deal with class imbalance

As we said at the beginning of the report, there is a problem of class imbalance that leads to low Sensitivity for Pop song regardless of the high Accuracy and Kappa score. We would try different methods to deal with class imbalance, including over-sampling, under-sampling and SMOTE.

1. Over-sampling

Over-sampling randomly samples the minority class with some replacement so that its size is roughly the same as other classes

```
ctrl$sampling <- 'up'

lda.fit_upsampling = train(track_genre_top ~ ., data =
train_df_most_imp[,1:161], method = 'lda', trControl = ctrl)

lda.fit_upsampling
```

```
Linear Discriminant Analysis

20603 samples
  160 predictor
    4 classes: 'Electronic', 'Hip-Hop', 'Pop', 'Rock'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 18542, 18543, 18542, 18544, 18543, 18543, ...
Addtional sampling using up-sampling

Resampling results:

  Accuracy  Kappa
  0.686104  0.5478946
```

With cross validation, the Accuracy score is 0.757 and the Kappa score is 0.6135.

2. Under-sampling

Under-sampling randomly subsets all the classes in the data set so that all the class sizes are roughly the same.

```
ctrl$sampling <- 'down'

lda.fit_downsampling = train(track_genre_top ~ ., data =
train_df_most_imp[,1:161], method = 'lda', trControl = ctrl)

lda.fit_downsampling
```

```
Linear Discriminant Analysis

11431 samples
  160 predictor
    4 classes: 'Electronic', 'Hip-Hop', 'Pop', 'Rock'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 10288, 10288, 10288, 10288, 10288, 10288, ...
Addtional sampling using down-sampling

Resampling results:

  Accuracy   Kappa
  0.6350814  0.486548
```

With cross validation, the Accuracy score is 0.68 and the Kappa score is 0.5415.

3. SMOTE

SMOTE stands for Synthetic Minority Oversampling Technique which creates new synthetic cases based on existing cases of the minority class but with more complicated methods other than random sampling to create new data. It is a more complicated version of the normal over-sampling method.

```
smote_data <- smote(track_genre_top ~ ., train_df_most_imp[,1:161])

lda.fit_smotesampling = train(track_genre_top ~ ., data = smote_data, method
= 'lda', trControl = ctrl)
lda.fit_smotesampling
```

```
Linear Discriminant Analysis

11431 samples
  160 predictor
    4 classes: 'Electronic', 'Hip-Hop', 'Pop', 'Rock'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 10288, 10288, 10288, 10288, 10288, 10288, ...
Addtional sampling using down-sampling

Resampling results:

  Accuracy   Kappa
  0.6350814  0.486548
```

With cross validation, the Accuracy score is 0.635 and the Kappa score is 0.4865.

4. More metrics

```
confusionMatrix(predict(lda.fit_upsampling, train_df_most_imp[,1:161]),
train_df_most_imp[,1:161]$track_genre_top)

confusionMatrix(predict(lda.fit_downsampling, train_df_most_imp[,1:161]),
train_df_most_imp[,1:161]$track_genre_top)

confusionMatrix(predict(lda.fit_smotesampling, smote_data),
smote_data$track_genre_top)
```

```
Confusion Matrix and Statistics

           Reference
Prediction   Electronic Hip-Hop  Pop Rock
  Electronic       4137     345  228  487
  Hip-Hop           991    1890  182  331
  Pop              1035     193  906 1775
  Rock              398      59  317 7329

Overall Statistics

               Accuracy : 0.6922
                 95% CI : (0.6859, 0.6985)
    No Information Rate : 0.4816
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5574

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: Electronic Class: Hip-Hop Class: Pop Class: Rock
Sensitivity                     0.6305        0.75995    0.55481      0.7387
Specificity                     0.9245        0.91698    0.84170      0.9275
Pos Pred Value                  0.7960        0.55687    0.23177      0.9045
Neg Pred Value                  0.8427        0.96531    0.95645      0.7926
Prevalence                      0.3184        0.12071    0.07926      0.4816
Detection Rate                  0.2008        0.09173    0.04397      0.3557
Detection Prevalence            0.2522        0.16473    0.18973      0.3933
Balanced Accuracy               0.7775        0.83847    0.69825      0.8331
Confusion Matrix and Statistics




           Reference
Prediction   Electronic Hip-Hop  Pop Rock
  Electronic       4089     348  224  487
  Hip-Hop           954    1882  175  334
  Pop              1073     201  921 1797
  Rock              445      56  313 7304

Overall Statistics

               Accuracy : 0.689
                 95% CI : (0.6827, 0.6953)
    No Information Rate : 0.4816
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.553

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: Electronic Class: Hip-Hop Class: Pop Class: Rock
Sensitivity                     0.6232        0.75674    0.56399      0.7361
Specificity                     0.9246        0.91924    0.83811      0.9238
Pos Pred Value                  0.7943        0.56263    0.23071      0.8997
Neg Pred Value                  0.8401        0.96494    0.95714      0.7903
Prevalence                      0.3184        0.12071    0.07926      0.4816
Detection Rate                  0.1985        0.09135    0.04470      0.3545
Detection Prevalence            0.2499        0.16235    0.19376      0.3940
Balanced Accuracy               0.7739        0.83799    0.70105      0.8300
Confusion Matrix and Statistics
```

```
          Reference
Prediction   Electronic Hip-Hop  Pop Rock
  Electronic       1471     106  621  208
  Hip-Hop           320     681  525  126
  Pop               328      60 2797  651
  Rock              135      19  956 2427

Overall Statistics

               Accuracy : 0.6453
                 95% CI : (0.6364, 0.654)
    No Information Rate : 0.4286
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5013

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: Electronic Class: Hip-Hop Class: Pop Class: Rock
Sensitivity                     0.6526        0.78637     0.5709      0.7113
Specificity                     0.8981        0.90809     0.8409      0.8616
Pos Pred Value                  0.6114        0.41223     0.7291      0.6862
Neg Pred Value                  0.9132        0.98108     0.7232      0.8752
Prevalence                      0.1972        0.07576     0.4286      0.2985
Detection Rate                  0.1287        0.05957     0.2447      0.2123
Detection Prevalence            0.2105        0.14452     0.3356      0.3094
Balanced Accuracy               0.7754        0.84723     0.7059      0.7864
```

The Accuracy and Kappa using over-sampling methods is the highest. The Sensitivity of Pop increases but there is a decrease in other metrics. All should be take into consideration when we choose the final model.

We also fit LDA with the top 300 most important variables using over-sampling method. We would also take this model into consideration.

```
ctrl$sampling <- 'up'

lda.fit_upsampling2 = train(track_genre_top ~ ., data =
train_df_most_imp[,1:301], method = 'lda', trControl = ctrl)

lda.fit_upsampling2
```

```
Linear Discriminant Analysis

20603 samples
  300 predictor
    4 classes: 'Electronic', 'Hip-Hop', 'Pop', 'Rock'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 18543, 18542, 18543, 18541, 18543, 18543, ...
Addtional sampling using up-sampling

Resampling results:

  Accuracy   Kappa
  0.7093044  0.5789688
```

VII. Model consideration

1. Prediction on test set

```
confusionMatrix(predict(lda.fit_important,test_df), test_df$track_genre_top)
```

```
Confusion Matrix and Statistics

            Reference
Prediction   Electronic Hip-Hop  Pop Rock
  Electronic       1777     515  279  617
  Hip-Hop             0       0    0    0
  Pop                 0       0    0    0
  Rock             1034     550  420 3635

Overall Statistics

               Accuracy : 0.6131
                 95% CI : (0.6029, 0.6233)
    No Information Rate : 0.4817
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3298

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Electronic Class: Hip-Hop Class: Pop Class: Rock
Sensitivity                     0.6322         0.0000    0.00000      0.8549
Specificity                     0.7655         1.0000    1.00000      0.5620
Pos Pred Value                  0.5574            NaN        NaN      0.6446
Neg Pred Value                  0.8166         0.8793    0.92081      0.8065
Prevalence                      0.3185         0.1207    0.07919      0.4817
Detection Rate                  0.2013         0.0000    0.00000      0.4118
Detection Prevalence            0.3612         0.0000    0.00000      0.6388
Balanced Accuracy               0.6988         0.5000    0.50000      0.7084
```

With just one variable `mfcc_median_3`, the Accuracy score is 0.6131 and the Kappa score is 0.3298. Just like with the training set, it predicts all as `Electronic` and `Rock`, making the Sensitivity for `Hip Hop` and `Pop` being 0 and the Specificity being 1.

```
confusionMatrix(predict(lda.fit_upsampling, test_df),
test_df$track_genre_top)
confusionMatrix(predict(lda.fit_top160, test_df), test_df$track_genre_top)
confusionMatrix(predict(lda.fit_upsampling2, test_df),
test_df$track_genre_top)
confusionMatrix(predict(lda.fit_top300, test_df), test_df$track_genre_top)
```

```
Confusion Matrix and Statistics

            Reference
Prediction   Electronic Hip-Hop  Pop Rock
  Electronic       1798     127  104  185
  Hip-Hop           411     833   85  145
  Pop               443      90  362  783
  Rock              159      15  148 3139

Overall Statistics

               Accuracy : 0.6947
                 95% CI : (0.685, 0.7043)
    No Information Rate : 0.4817
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5614

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: Electronic Class: Hip-Hop Class: Pop Class: Rock
Sensitivity                     0.6396        0.78216    0.51788      0.7382
Specificity                     0.9309        0.91742    0.83809      0.9296
Pos Pred Value                  0.8121        0.56513    0.21573      0.9070
Neg Pred Value                  0.8468        0.96845    0.95286      0.7926
Prevalence                      0.3185        0.12065    0.07919      0.4817
Detection Rate                  0.2037        0.09437    0.04101      0.3556
Detection Prevalence            0.2508        0.16699    0.19010      0.3921
Balanced Accuracy               0.7852        0.84979    0.67799      0.8339
Confusion Matrix and Statistics


            Reference
Prediction   Electronic Hip-Hop  Pop Rock
  Electronic       2176     246  177  255
  Hip-Hop           183     711   64   83
  Pop                70      13   97  120
  Rock              382      95  361 3794

Overall Statistics

               Accuracy : 0.7679
                 95% CI : (0.7589, 0.7766)
    No Information Rate : 0.4817
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.63

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: Electronic Class: Hip-Hop Class: Pop Class: Rock
Sensitivity                     0.7741        0.66761    0.13877      0.8923
Specificity                     0.8873        0.95749    0.97502      0.8168
Pos Pred Value                  0.7624        0.68300    0.32333      0.8191
Neg Pred Value                  0.8937        0.95453    0.92940      0.8908
Prevalence                      0.3185        0.12065    0.07919      0.4817
Detection Rate                  0.2465        0.08055    0.01099      0.4298
Detection Prevalence            0.3233        0.11793    0.03399      0.5248
Balanced Accuracy               0.8307        0.81255    0.55690      0.8546
Confusion Matrix and Statistics
```

```
           Reference
Prediction   Electronic Hip-Hop  Pop Rock
  Electronic       1905     131  105  188
  Hip-Hop           346     837   85  128
  Pop               407      87  372  730
  Rock              153      10  137 3206

Overall Statistics

               Accuracy : 0.716
                 95% CI : (0.7065, 0.7254)
    No Information Rate : 0.4817
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5891

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: Electronic Class: Hip-Hop Class: Pop Class: Rock
Sensitivity                     0.6777        0.78592    0.53219      0.7540
Specificity                     0.9295        0.92798    0.84941      0.9344
Pos Pred Value                  0.8179        0.59957    0.23308      0.9144
Neg Pred Value                  0.8606        0.96932    0.95478      0.8034
Prevalence                      0.3185        0.12065    0.07919      0.4817
Detection Rate                  0.2158        0.09482    0.04214      0.3632
Detection Prevalence            0.2638        0.15815    0.18081      0.3972
Balanced Accuracy               0.8036        0.85695    0.69080      0.8442
Confusion Matrix and Statistics

           Reference
Prediction   Electronic Hip-Hop  Pop Rock
  Electronic       2206     225  163  233
  Hip-Hop           184     752   62   77
  Pop               106      16  144  127
  Rock              315      72  330 3815

Overall Statistics

               Accuracy : 0.7836
                 95% CI : (0.7749, 0.7922)
    No Information Rate : 0.4817
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6579

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: Electronic Class: Hip-Hop Class: Pop Class: Rock
Sensitivity                     0.7848        0.70610    0.20601      0.8972
Specificity                     0.8968        0.95839    0.96937      0.8433
Pos Pred Value                  0.7803        0.69953    0.36641      0.8418
Neg Pred Value                  0.8992        0.95962    0.93419      0.8983
Prevalence                      0.3185        0.12065    0.07919      0.4817
Detection Rate                  0.2499        0.08519    0.01631      0.4322
Detection Prevalence            0.3203        0.12179    0.04452      0.5134
Balanced Accuracy               0.8408        0.83225    0.58769      0.8703
```

The metrics are slightly different from the training set but they are porprotionally the same between variables

2. Choosing models

There are 4 models we need to consider:

- LDA with the top 160 most important variables
- LDA with the top 160 most important variables using over-sampling method
- LDA with the top 300 most important variables
- LDA with the top 300 most important variables using over-sampling method

There may be some limitations on time and computational resources if we use 300 variables on a large scale. However, if we have enough time and computational resources, we should use 300 variables for sure. This depends more on the case-by-case context.

The only thing we that we should concern about is should we use the models with over-sampling method or not. The Sensitivity of Pop does increase but other metrics decrease. We have to think through the context carefully.

In this context, a False Positive (FP) is definitely better than a False Negative (FN). To be more specific, it is generally better to put a song that is not Pop into a Pop playlist or recommend a non-pop song to the listeners when they want a pop song than missing a Pop song in a Pop playlist. A song usually doesn't have just one genre but secondary or even tertiary genres so it's find to classify a song that doesn't have pop as the primary genre into pop songs. But think about the situation when a Pop song is missed out of the Pop playlist. Listeners are more open to trying out new songs than missing out songs that may be later popular. In a real life situation, we also have the artists as a predictor variable that may reduce the number of FNs.

In this context, a True Positive (TP) is also better than a True Negative (TN). The main goal is to identify songs that listeners likes or identify songs to be put into the right playlists. The main goal is not to avoid songs that listeners may not like or avoid putting songs into the wrong playlists. Even if we choose the wrong songs to put into a playlist, it's definitely better to have some songs in the playlists than to have nothing in them. A small number of songs wouldn't encourage listeners to listen more on our platform.

As a whole, FP and TP are more important. Therefore, the models that have a more balanced Sensitivity would be better. This indicates that models using over-sampling method are better. Our final model is the LDA fit with the top 160 most important features with over-sampling method. Below are the top 160 features.

```
colnames(train_df_most_imp[,2:161])
```

```
  [1] "mfcc_median_20"                "mfcc_median_3"                 "mfcc_mean_20"
  [4] "spectral_centroid_std_1"       "mfcc_mean_3"                   "mfcc_median_18"
  [7] "mfcc_mean_18"                  "mfcc_max_16"                   "spectral_rolloff_std_1"
 [10] "mfcc_max_10"                   "mfcc_mean_2"                   "mfcc_median_16"
 [13] "mfcc_max_14"                   "mfcc_mean_16"                  "mfcc_std_13"
 [16] "mfcc_std_11"                   "mfcc_median_2"                 "mfcc_max_18"
 [19] "mfcc_std_9"                    "mfcc_skew_3"
"spectral_contrast_min_5"
 [22] "spectral_contrast_min_4"       "mfcc_max_12"                   "mfcc_std_14"
 [25] "rmse_std_1"                    "mfcc_std_12"                   "mfcc_std_15"
 [28] "mfcc_std_8"                    "mfcc_std_16"                   "zcr_std_1"
 [31] "mfcc_max_3"                    "mfcc_std_10"                   "mfcc_max_4"
 [34] "mfcc_median_14"                "mfcc_max_20"
"spectral_contrast_min_3"
 [37] "mfcc_std_17"                   "mfcc_max_17"                   "mfcc_max_7"
 [40] "mfcc_std_6"                    "mfcc_max_15"
"spectral_rolloff_kurtosis_1"
 [43] "mfcc_max_6"                    "spectral_contrast_std_4"       "mfcc_mean_14"
 [46] "mfcc_std_7"                    "mfcc_kurtosis_2"               "mfcc_skew_1"
 [49] "spectral_contrast_min_2"       "chroma_stft_skew_1"            "chroma_stft_skew_11"
 [52] "mfcc_max_9"                    "spectral_contrast_min_6"       "mfcc_max_8"
 [55] "chroma_stft_skew_2"            "spectral_bandwidth_std_1"      "mfcc_std_18"
 [58] "spectral_contrast_median_5"    "spectral_centroid_kurtosis_1"  "mfcc_min_2"
 [61] "chroma_stft_skew_4"            "mfcc_min_11"
"spectral_bandwidth_mean_1"
 [64] "mfcc_std_4"                    "spectral_rolloff_skew_1"
"spectral_bandwidth_median_1"
 [67] "mfcc_median_12"                "mfcc_max_19"                   "chroma_stft_median_1"
 [70] "spectral_contrast_max_4"       "chroma_stft_mean_1"            "mfcc_median_10"
 [73] "chroma_stft_skew_3"            "mfcc_std_19"                   "mfcc_max_11"
 [76] "mfcc_max_1"                    "chroma_stft_skew_6"
"spectral_contrast_std_3"
 [79] "mfcc_mean_10"                  "mfcc_max_13"                   "mfcc_mean_4"
 [82] "spectral_contrast_max_7"       "mfcc_skew_2"                   "chroma_stft_skew_12"
 [85] "mfcc_mean_12"                  "mfcc_std_2"                    "tonnetz_std_2"
 [88] "mfcc_kurtosis_1"               "chroma_stft_median_2"
```

```
 [91] "chroma_stft_median_11"          "spectral_bandwidth_skew_1"     "chroma_stft_mean_2"
 [94] "spectral_contrast_min_1"        "mfcc_min_1"
"spectral_contrast_std_6"
 [97] "mfcc_std_20"                    "chroma_stft_skew_5"            "mfcc_std_5"
[100] "spectral_contrast_mean_5"       "spectral_rolloff_mean_1"       "chroma_stft_median_4"
[103] "tonnetz_mean_2"                 "mfcc_median_4"                 "mfcc_min_7"
[106] "chroma_stft_mean_11"            "rmse_max_1"
"chroma_stft_kurtosis_11"
[109] "tonnetz_median_2"               "chroma_stft_median_3"          "chroma_stft_skew_8"
[112] "mfcc_min_5"                     "mfcc_median_7"
"spectral_contrast_median_4"
[115] "chroma_stft_median_6"           "mfcc_mean_7"                   "chroma_stft_mean_4"
[118] "mfcc_min_13"                    "mfcc_mean_6"                   "chroma_stft_skew_10"
[121] "spectral_contrast_std_2"        "chroma_stft_skew_7"            "chroma_stft_median_12"
[124] "mfcc_min_9"                     "chroma_stft_mean_3"
"spectral_contrast_skew_1"
[127] "chroma_stft_skew_9"             "spectral_contrast_mean_4"      "zcr_median_1"
[130] "spectral_contrast_max_3"        "spectral_contrast_kurtosis_2"  "chroma_stft_kurtosis_1"
[133] "mfcc_mean_19"                   "mfcc_median_6"                 "rmse_kurtosis_1"
[136] "chroma_stft_mean_6"             "spectral_bandwidth_kurtosis_1" "zcr_max_1"
[139] "spectral_centroid_mean_1"       "chroma_cqt_std_9"
"spectral_contrast_skew_2"
[142] "mfcc_median_19"                 "chroma_stft_median_5"
"spectral_contrast_std_7"
[145] "mfcc_median_17"                 "chroma_stft_std_11"            "mfcc_kurtosis_3"
[148] "mfcc_mean_1"                    "spectral_contrast_std_5"
"spectral_contrast_mean_3"
[151] "spectral_contrast_median_3"     "mfcc_mean_17"
"spectral_centroid_skew_1"
[154] "tonnetz_std_5"                  "chroma_cqt_std_11"             "tonnetz_std_6"
[157] "mfcc_median_1"                  "chroma_stft_kurtosis_2"        "mfcc_min_4"
[160] "mfcc_std_1"
```

VIII. Conclusions

For the two questions we aim to answer at the beginning,

1. If we are limited to a single feature to decide the genre, we will choose `mfcc_median_3`.
2. If we can consider multiple features, we will choose the first top 160 most important variable generated by `varImp` and use over-sampling method on it.