STOR 565 Final: Group 8

Daniel Zhang, Jake Mareno, Alex McGarrigle

2023-12-06

I. Project Summary

Through the use of classification modeling methods, our project aimed to create a model which can accurately predict the primary genre of a track uploaded to Spotify. To do this, we used a dataset of tracks listed on Spotify which included their listed genre and various machine-generated metrics describing the audio. We ultimately attempted five different modeling methods: Naive Bayes, Linear Discriminant Analysis, Quadratic Discriminant Analysis, K-Nearest Neighbors, and Random Forest Classification. By comparing the viability of each model, we explored the potential strengths and limitations of each approach in correctly classifying the genre of tracks on Spotify. We believe that a quality classification model could assist in building a streamlined genre assignment process on Spotify, increasing the accuracy of genre labels on the platform and potentially allowing for the automation of genre assignment.

II. AI Disclaimer and Member Contribution Statement

All members attest that no AI was used at any stage of this project.

III. Data

Source

Our dataset, taken from Kaggle, consisted of 50,000 observations, each representing an audio track that had been uploaded to Spotify. These observations were divided evenly into ten groups of 5,000 for each response class, recorded in the music_genre variable. The ten Spotify-defined genres represented in the response were Alternative, Anime, Blues, Classical, Country, Electronic, Jazz, Hip-Hop, Rap, and Rock. Each observation included several "audio features," referring to various descriptive metrics automatically calculated and recorded by Spotify for tracks uploaded to their database. These audio feature variables were used in our models to predict music_genre; the remaining variables, which included track name, artist name, numerical Spotify ID, and the date on which each observation was obtained from the Spotify API, were not considered valuable for classification and therefore removed from the dataset prior to analysis.

Explanation of Variables

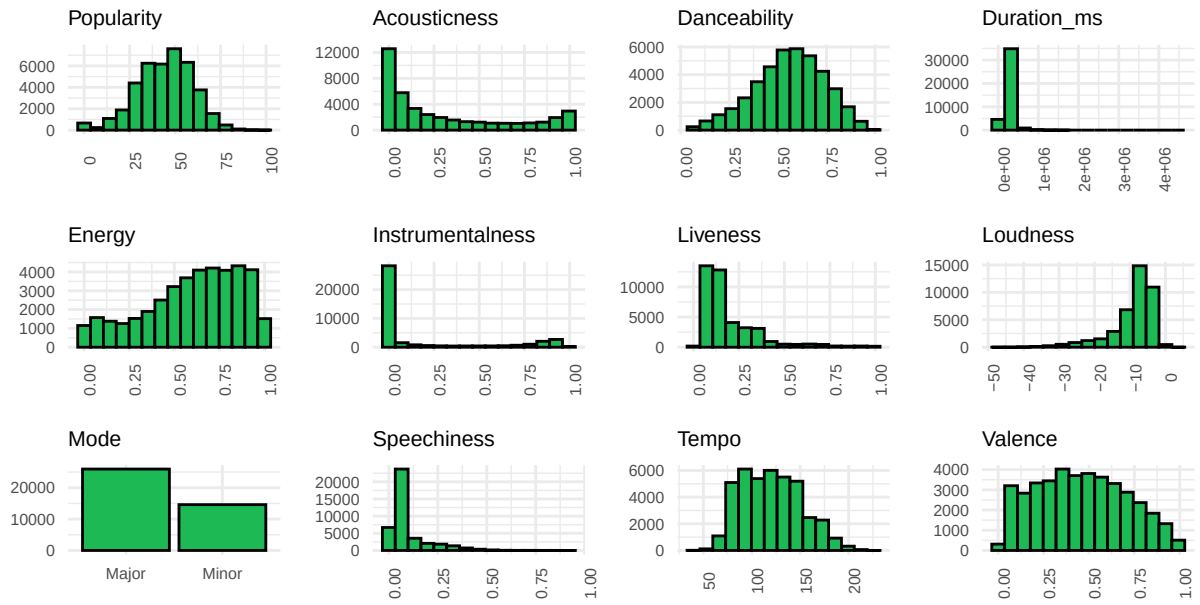
- `acousticness`: Confidence in a track being produced with acoustic instruments. Measured on a continuous zero-to-one scale, with 0.0 representing fully electronic tracks and 1.0 representing fully acoustic tracks.
- `danceability`: A "dancing suitability" rating which considers factors such as tempo. Measured on a continuous zero-to-one scale, with 0.0 representing the least "danceable" tracks and 1.0 representing the most.
- `duration_ms`: Track duration measured in milliseconds.
- `energy`: Indicates the perceived intensity and activity of a track using factors such as loudness and timbre. Measured on a continuous zero-to-one scale, with 0.0 representing the least energetic tracks and 1.0 representing the most.
- `instrumentalness`: Confidence in the lack of vocals in a track. Singing which does not contain words (referred to as "'ooh' and 'aah' sounds" by Spotify documentation) is not considered to be vocal. Measured on a continuous zero-to-one scale, with values above 0.5 generally * indicating instrumental tracks and perfect scores of 1.0 indicating full confidence in a lack of vocals.
- `key`: Musical key of a track.
- `liveness`: Confidence in a track being a recording of a live performance. Measured on a continuous zero-to-one scale, with values above 0.8 indicating high confidence in a track having been performed live.
- `loudness`: Average loudness of a track, measured in decibels.
- `mode`: Modality of a track; whether a track is in a major or minor key.
- `popularity`: The popularity of a track on Spotify. Measured on an integer scale from zero to one hundred, with values of 0 representing the least popular tracks and 100 representing the most popular.
- `speechiness`: Confidence in the presence of spoken word at the exclusion of instrumentals. Measured on a continuous zero-to-one scale, with values above 0.66 indicating tracks that are not musical and values below 0.66 but above 0.33 indicating a likely break in music for a * spoken section or intermission.
- `tempo`: Speed of a track, measured in beats per minute.
- `valence`: Musical "positiveness" of a track, which is given no clear basis for measurement in the Spotify documentation. Measured on a continuous zero-to-one scale, with 0.0 representing tracks which inspire negative emotions and 1.0 representing more upbeat tracks.

Cleaning

Fortunately, not very much cleaning was needed. We simply needed to reformat the variables `mode`, `key`, and `genre` as factors instead of pure character vectors and make sure that R recognized `tempo` as a numeric variable. We also had to filter out entries that had missing data for tempo and/or duration, which ended up cutting the size of our data down to 40,560 observations instead of 50,000. For all of our models, we use a 9:1 split of training data to test data, resulting in **32,850** training observations an **3,650** test observations.

IV. Exploratory Data Analysis

Distributions of Features



First, we did a brief review of the distribution of all predicting features using histograms (we also included a basic count barplot for the mode variable). We can group the features like so:

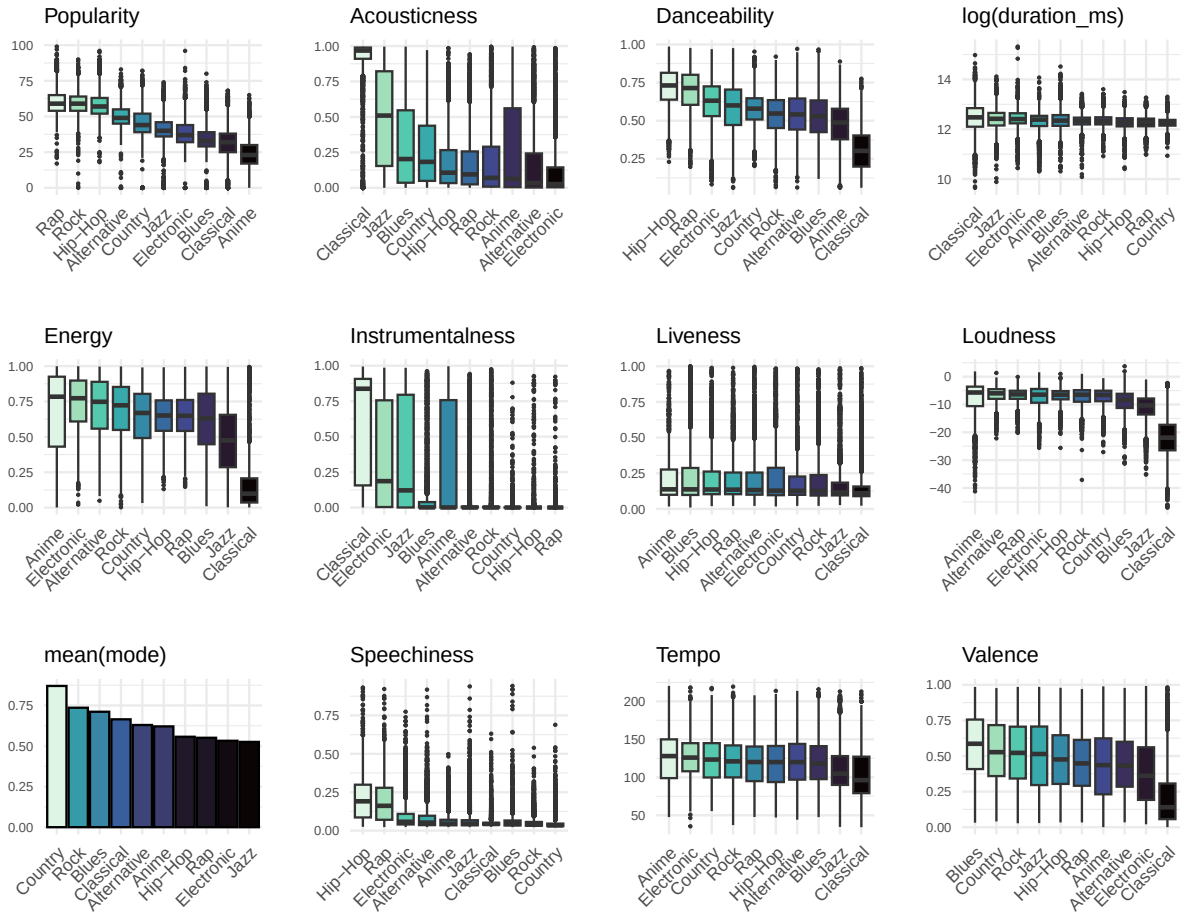
- (Roughly) Gaussian: `popularity`, `danceability`, `tempo`, `valence`
- Skewed Right: `acousticness`, `duration_ms`, `instrumentalness`, `liveness`, `speechiness`
- Skewed Left: `energy`, `loudness`

Note that variables `instrumentalness` and `acousticness` also show a slight bimodal peak at the right end of their distribution. This makes sense as the Spotify algorithm is likely quite sure that a song is instrumental/acoustic or not, with few cases of it being unsure and assigining a value in the 0.1-0.9 range. We also see a very sharp right skew in `duration`, which suggests that while most songs may be in the 2-5 minute range, there are a few extreme outliers that are closer to the 10-60 minute range. Overall, we note that a significant portion of our features are not very close to being Gaussian/normal, which may cause issues with models such as LDA & QDA which implicitly assume a Gaussian distribution.

Feature Correlation

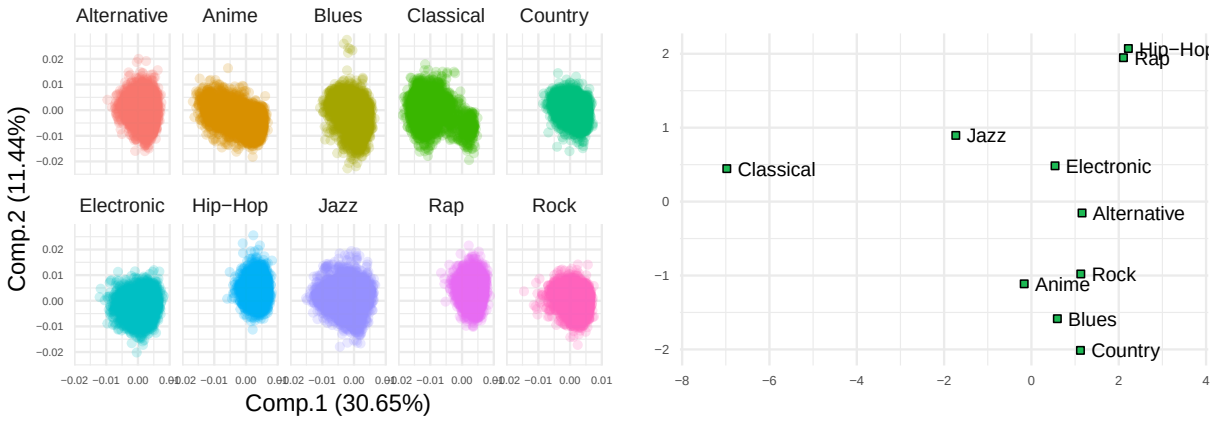
Next we check the correlation between all of our features. We find that the most strongly correlated variables are `acousticness` & `energy`, `acousticness` & `loudness`, and `energy` and `loudness`, which makes sense as acoustic songs are likely to have lower energy and loudness, while energetic songs are like to be more loud. Besides these 3 strong correlations, from the color density of the plot we can still see there is a handful of instances of moderate correlation between nearly all of the variables apart from `duration_ms`, `liveness` and `mode`, which appear to be mostly independent. Therefore, we must consider the implications of these correlations when using a method that assumes independence (i.e. Naive Bayes).

Distributions by Genre



We now look at the distribution of each feature conditional on a track's genre using boxplots. Note that we took a log transform to visualize `duration_ms`, as otherwise the majority of the distribution would be shrunk to the bottom and the plot would overemphasize outliers. Overall, it appears the the features with the most consistent variability between genres are `popularity`, `acousticness`, `danceability`, and `instrumentalness`, so we expect these variables to potentially be more useful in making predictions. Meanwhile, the variables `duration_ms`, `liveness`, and `tempo` appear to have very similar distributions regardless of the genre, so these variables may end up being not as useful. We can also observe that some genres have a quite distinct different distribution in certain categories, such as Classical in regards to `acousticness`, `energy`, and `loudness`, or Hip-Hop and Rap with regards to `speechiness`. These specialized features may be quite useful/proficient at predicting these specific genres, even if they are less useful in distinguishing the remaining more similar genres. We can see that Classical tends to have the most unique distributions overall, so it will likely end up being the easiest genre to classify. Meanwhile, genres like Rap & Hip-Hop or Rock & Country tend to have similar distributions, meaning that it may be difficult to distinguish between them.

PCA & MDS of Genres



	popula..	acoust..	dancea..	durati..	energy	instru..	liveness	loudness	mode	speech..	tempo	valence
Comp.1	0.25	-0.43	0.30	-0.06	0.44	-0.34	0.09	0.46	-0.01	0.17	0.16	0.28
Comp.2	0.34	0.22	0.46	-0.40	-0.30	-0.09	-0.29	-0.17	-0.10	0.33	-0.33	0.16

Finally, we applied the dimension reduction techniques of Principal Components Analysis (PCA) and Multidimensional Scaling (MDS) to visualize the similarity or dissimilarity between genres.

Looking at the general shape/distribution of each genre based on the first two PCs of the data, we see a large amount of overlap but some small trends are evident. Notice that Rap and Hip-Hop have very similar distributions that are clustered to the right and slightly higher than other genres. This makes sense because for example the positive X component is positively associated with `popularity` & negatively associated with `acousticness`, and the positive Y component is positively associated with `speechiness` and `danceability`, and both of these genres tend to score higher (lower for acousticness) on all of these categories. In contrast, Classical is shifted to the left because it tends to have a higher `acousticness` score with a lower `energy` and `loudness` score. Some other evident trends are that the overall distributions of Country, Blues, and Rock are also quite similar, or that Anime and Jazz are the only distributions besides Classical that have a strong present in the left half of their plots.

MDS yields extremely similar findings. For MDS, to generate a distance matrix, we normalized all features (to have a mean of 0, standard deviation of 1), and then calculated the conditional mean of each scaled feature dependent on the genre, and simply used the pairwise Euclidean distance between the 12-dimensional vectors representing each set of genre means. The results are in line with our intuitions and observations from the rest of our analysis in that we again see that Hip-Hop and Rap are almost on top of each other, while Classical is completely isolated (Jazz is also relatively isolated). We can also note how Alternative is "in between" Electronic and Rock or how Blues is "in between" Rock and Country, which aligns with our intuitions about the styles of these genres and again demonstrates that these genres may be harder to distinguish.

V. Modeling

Naive Bayes

Naive Bayes is based on Bayes Theorem to determine class probabilities. It works on the assumption that features are conditionally independent from each other which is somewhat violated in our dataset. For example, Energy and Loudness have a strong positive correlation, providing evidence that they are not conditionally independent. In addition, the R function assumes that the quantitative features are normally

distributed but not all of our features are. As such, we do not expect Naive Bayes to provide very accurate predictions. The test error rate and confusion matrix are shown below:

```
## [1] "Test Error Rate: 0.56274"
```

```
##              truth
## predictions   Electronic Anime Jazz Alternative Country  Rap Blues Rock Classical Hip-Hop  Sum
##   Electronic         157    30   58          12       1    3    21    5         9       3  299
##   Anime               15   156    6           2       3    0    21    0         7       0  210
##   Jazz                35     8  123          17      12    6    32   15         9       5  262
##   Alternative         21     9    7          63       3   11    11   22         2       5  154
##   Country             62   117   83         155     294   35   172  131        11      35 1095
##   Rap                 12     1    2          29       8  180     2   29         0     175  438
##   Blues               14    12   25           8      13    3    77    5         4       1  162
##   Rock                 9     1    5          38      25   25    11  112         0      10  236
##   Classical            7    59   55           1       5    0    16    9       287       0  439
##   Hip-Hop             17     0   25          37      14  105     2    8         0     147  355
##   Sum                349   393  389         362     378  368   365  336       329     381 3650
```

Naive Bayes gives us the worst test error of all the models we tested and the confusion matrix shows some clear problems. The size of each prediction is unbalanced. For example, from out test set Naive Bayes predicts that around 1000 of them have genre "Country" However, in the truth we can see that it should predict an even distribution of classes of values around 350. We believe that this is because Country's features have a distribution that is very similar to other genres but with a mean that is slightly shifted. Because of this when those other classes have their features farther from their mean but closer to Country's mean Naive Bayes will predict a higher probability as Country compared to those other classes. Overall, we can tell from this model that the features values are not very well separated for most of the classes.

Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) assumes that the features for each class are of Gaussian distribution and also that each distribution for each class has the same covariance matrix, leading to linear decision boundaries. In our dataset some of our classes don't follow Gaussian distribution. For example, looking at the PCA plots by Genre, Anime and Classical obviously do not have eliptical shapes which means that they likely do not follow Gaussian distribution. In addition even if each class followed a Gaussian Distribution, it is clear from the PCA plots that the covariance matricies for each class distribution would not be the same. The test error rate and confusion matrix for LDA are shown below:

```
## [1] "Test Error Rate: 0.47726"
```

```
##              truth
## predictions   Electronic Anime Jazz Alternative Country  Rap Blues Rock Classical Hip-Hop  Sum
##   Electronic         195    26   70          20       6    3    11    3        10       9  353
##   Anime               23   240    5           2       5    0    46    0        15       0  336
##   Jazz                27    10  126          15       5    0    33   10        13       1  240
##   Alternative         24    14   12         133      22   30    17   61         5      24  342
##   Country             20    14   46          76     211   18    33   29         3      11  461
##   Rap                  5     0    4          12       2  118     1   10         0     107  259
##   Blues               26    55   57           8      47    1   192    0        10       0  396
##   Rock                 7     2   14          61      70   51    20  217         0      26  468
##   Classical            3    32   25           0       1    0     9    1       273       0  344
##   Hip-Hop             19     0   30          35       9  147     3    5         0     203  451
##   Sum                349   393  389         362     378  368   365  336       329     381 3650
```

LDA results in our 3rd best model and provides class prediction sizes that are a little more even although some classes still have larger prediction sizes than others (Country and Hip-Hop). Overall, we believe this is because when LDA fits normal distributions for each class, they all have the same covariance matrix which means that each area that we predict a certain class will have a more similar size leading to more uniform class prediction sizes. However, this is still a little unbalanced because the means of classes like Country, Rock, and Hip-Hop have similar means to other classes which reduces the area that we predict those other

classes since instead we would predict Country, Rock, or Hip-Hop. Overall, LDA performs better than Naive Bayes in terms of accuracy of almost all classes.

Quadratic Discriminant Analysis

Quadratic Discriminant Analysis (QDA) is similar to LDA in that assumes the features for each class are of Gaussian distribution, but instead it does not assume that the covariance matrix are the same between classes. This allows it a greater deal of flexibility than LDA. The assumption that the class features are of Gaussian distribution is violated in our dataset as explained in the previous section. The test error rate and confusion matrix are shown below:

```
## [1] "Test Error Rate: 0.53945"
```

```
##              predicted
## truth         Electronic Anime Jazz Alternative Country  Rap Blues Rock Classical Hip-Hop  Sum
##   Electronic         172    28   56          19       2    5    15    3         7       6  313
##   Anime               22   188    8           3       5    0    13    0        12       0  251
##   Jazz                27    16  128          14       4    3    29   13        11       1  246
##   Alternative         13     7   14          59       3   19    14   21         2      10  162
##   Country             40    73   57         119     275   23   116  102         7      19  831
##   Rap                 13     0    2          31      15  133     3   51         2     137  387
##   Blues               30    45   55          15      21    2   146    9        11       1  335
##   Rock                 9     0    7          51      36   36    13  119         0      23  294
##   Classical            3    33   38           1       2    0    14    6       277       0  374
##   Hip-Hop             20     3   24          50      15  147     2   12         0     184  457
##   Sum                349   393  389         362     378  368   365  336       329     381 3650
```

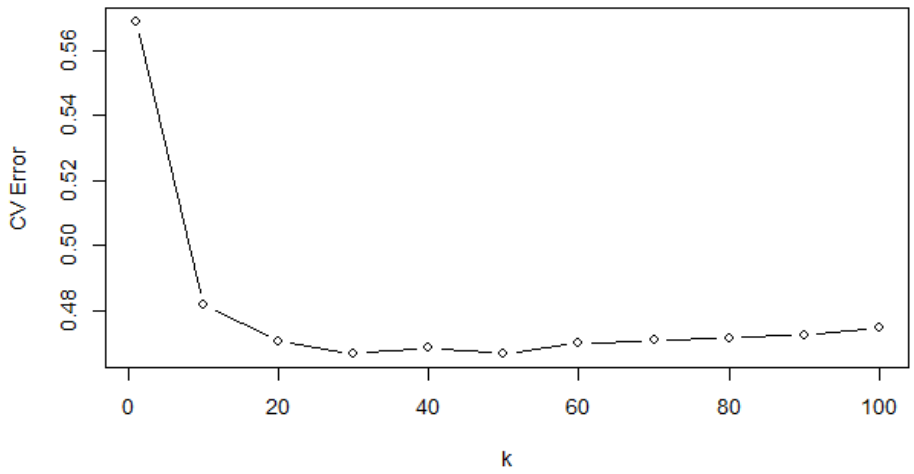
Surprisingly, the Test Error Rate is a lot higher for QDA. We also tried training error and it was also higher. We believe that this is because the covariance estimate for our dataset couldn't account the distribution of features for each class since it was non normal and weirdly spread out. Looking at the confusion matrix, QDA also leads to inbalanced class prediction sizes. For example, Country was predicted around 800 times which is more than twice the true amounts of Country in the test set. Again we believe that the reason this occurs is that there is a lot of overlap in the area where other classes have lower probabilities and where Country has high probabilities which causes Country to be predicted there, leading to more Country predictions. Overall QDA performs better than Naive Bayes but worse than LDA.

K-Nearest Neighbors

K-Nearest Neighbors (KNN) uses a distance metric to find the k nearest observations to the observation that we want to predict and uses their class to predict the value of this new observation. In our analysis we use Euclidean distance and decide the classification using majority vote with ties broken at random. We can adjust k to change the complexity of our model with higher k's leading to a less complex model and lower k's leading to a more complex model. We use 6 fold cross validation to estimate test errors at different values of k to choose the best k for our dataset. The code below tests k values of 1, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100. Choosing higher values of k results in computational problems in R so we opt to stop at k = 100.

```
errors = c()
ks = c(1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100) # Ranges of Ks to plot
for (k in ks) {
  err = c()
  for (i in 1:6) {
    test = music_train[(1 + (5475 * (i - 1))):(5475 * i),]
    train = music_train[-(1 + (5475 * (i - 1))):(5475 * i)),]
    mod_knn = knn(scale(train[,c(-7, -10, -14)]), scale(test[,c(-7, -10, -14)]),
                  cl=train$music_genre, k=k)
    err = c(err, mean(mod_knn != test$music_genre))
  }
  errors = c(errors, mean(err))
```

```
}
plot(ks, errors, xlab = "k", ylab = "CV Error", type="b")
```



We can see from the plot that the lowest error occurs with k = 50, which is a pretty complex decision boundary considering that the dataset has around 40,000 observations. Next we use k = 50 as our model and get our test error rate and confusion matrix.

```
## [1] "Test Error Rate: 0.46795"
```

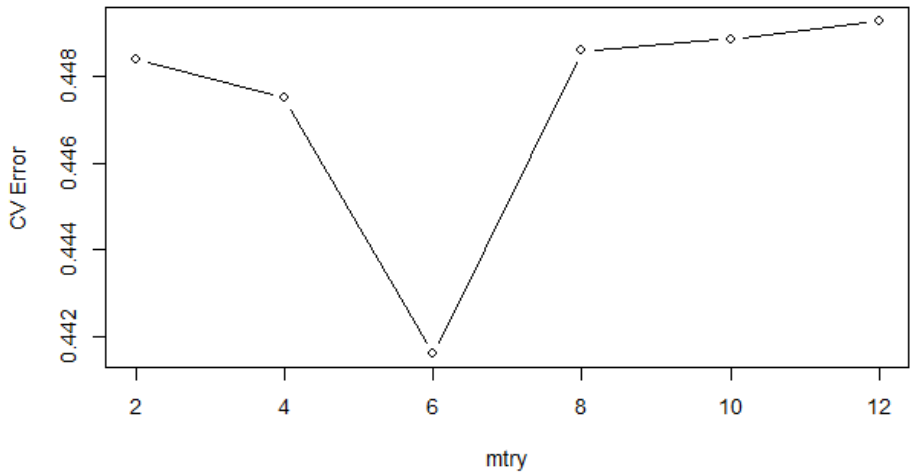
```
##              truth
## predicted     Electronic Anime Jazz Alternative Country  Rap Blues Rock Classical Hip-Hop  Sum
##   Electronic         180    12   44           9       6    4     9    3         8       4  279
##   Anime               33   278    6           7       7    0    36    0         7       0  374
##   Jazz                30    10  154          15      20    2    29   14        15       1  290
##   Alternative         15    12   16         111      24   17    13   48         6       7  269
##   Country             33    22   42          90     221    9    63   26         2      12  520
##   Rap                  4     0    4          17       5  138     2   11         0     157  338
##   Blues               16    26   51           6      17    1   182    3         4       2  308
##   Rock                13     2   16          64      66   56    19  223         0      30  489
##   Classical            7    30   32           1       1    0     9    2       287       0  369
##   Hip-Hop             18     1   24          42      11  141     3    6         0     168  414
##   Sum                349   393  389         362     378  368   365  336       329     381 3650
```

KNN performs well as our second best model. KNN performs worse at predicting some classes like Hip-Hop and Alternative but better on Anime and Classical when compared to LDA. We believe that this is because the distribution of features from Anime and Classical are more non-normal and have some more spaced out values. KNN performs better on these because LDA doesn't capture these very well by fitting a normal distribution on it while KNN avoids this problem since it only uses its neighbors classes for classification. Overall, it performs better than LDA on select areas while also performing worse in some areas.

Random Forest

Random Forest Classifier (RFC) combines the output of multiple decision trees that are built on a variable number of predictors and also on a subset of the dataset. One potential worry is that since it relies on decision trees which make splits depending on features, it may not be able to completely separate some of the classes from the others since there is so much overlap between the distribution of classes. In addition, training a random forest model can be very expensive. As such we choose the number of trees to be used as 100 since increasing it did not result in much difference in training error. Then we must choose the number of predictors to be used in each tree which we do with 6-fold cross validation below:

```
errors = c()
mtrys = c(2, 4, 6, 8, 10, 12)
for (m in mtrys) {
  err = c()
  for (i in 1:6) {
    test = music_train[(1 + (5475 * (i - 1))):(5475 * i),]
    train = music_train[-(1 + (5475 * (i - 1))):(5475 * i)),]
    rfc = randomForest(music_genre~., data = train, mtry = m, ntree = 100)
    error = mean(music_test$music_genre != predict(rfc, music_test))
    err = c(err, error)
  }
  errors = c(errors, mean(err))
}
plot(mtrys, errors, ylab = "CV Error", xlab = "mtry", type="b")
```



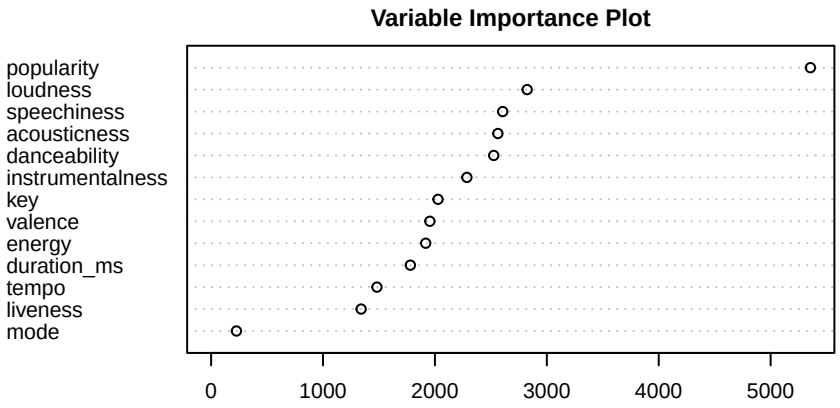
Looking at the RFC cross validation plot above, we can see that the lowest cross validation error occurs at mtry=6. This means that each tree using 6 predictors provides the lowest cross validation error. In addition, we can conclude that there are a few strong features in our dataset. Next we create a model with mtry=6 below:

```
## [1] "Test Error Rate: 0.45534"
```

```
##              truth
## prediction    Electronic Anime Jazz Alternative Country  Rap Blues Rock Classical Hip-Hop  Sum
##   Electronic         198    17   50          20       3    0    17    1         5       3  314
##   Anime               17   293    3           6       6    0    23    1         8       0  357
##   Jazz                52     6  201          22      12    0    43    8        16       3  363
##   Alternative         15    10   20         125      34   16    17   47         6      21  311
##   Country              9    13   25          53     214    5    30   31         0       4  384
##   Rap                  9     1    4          16       4  112     0   23         0     197  366
##   Blues               28    29   45           7      19    2   209    6        11       1  357
##   Rock                14     2   13          77      77   35    21  212         1      13  465
##   Classical            4    22   16           2       2    0     5    0       282       0  333
##   Hip-Hop              3     0   12          34       7  198     0    7         0     139  400
##   Sum                349   393  389         362     378  368   365  336       329     381 3650
```

RFC gives us the lowest test error rate. It provides the most balanced prediction classes of all the other models. We believe this is because of RFC's ability to use a lot of decision trees which should split the data pretty evenly. In addition, it outperforms almost every other model in predicting almost all the classes except for Rap and Hip-Hop which it performs the worst. We believe this is because the distribution of Rap and Hip-Hop are so similar that by using decision trees they are not separable. They most likely need a

combination of factors in order to be separated. Overall, RFC provides the best model overall in terms of classification accuracy although a little worse at predicting Rap and Hip-Hop.



Random Forest also provides a way to observe the predictive power of each feature in the data through the variable importance plot above. We can see that popularity is the strongest by far. This makes sense because looking at the distributions by genre we can clearly see that the difference in means by popularity varies the most for each genre. Thus the decision trees can easily use this to classify each class. Meanwhile mode is the weakest which makes sense since it only takes 2 values and each genre has modes of both values which means that it cannot really be used to split the data very well.

Performance Summary: Misclassification Rate of Each Model

NB	LDA	QDA	KNN	RFC
0.563	0.477	0.539	0.468	0.456

VI. Bonus: Applying Our Model to Famous Works

We were curious to see the results from applying our best (RFC) model to a few famous albums from our favorite genres to see how accurate it was and which tracks it would correctly or incorrectly classify. To do this, we needed to use the Spotify API together with the packages `spotipy` and `pandas` for Python to create CSVs containing track metrics, which we could then import into R and apply our model to. The first table shows an example applying the model to the Alternative album *Achtung Baby* by Irish band U2, while the second shows a summary of our results. Note that we considered Rap and Hip-Hop together for this instance (we considered the model correct if it predicted either genre). We see that the predictions for Classical, Jazz, Hip-Hop/Rap and Electronic were relatively accurate, while Alternative, Country, and Rock struggled.

Song Title	Prediction
Zoo Station	Electronic
Even Better Than The Real Thing	Blues
One	Rock
Until The End Of The World	Blues
Who's Gonna Ride Your Wild Horses	Rock
So Cruel	Electronic
The Fly	Alternative
Mysterious Ways	Rock
Tryin' To Throw Your Arms Around The World	Alternative
Ultra Violet (Light My Way)	Electronic
Acrobat	Alternative
Love Is Blindness	Electronic

Artist	Album	Genre	Accuracy	Fraction
U2	Achtung Baby	Alternative	0.25	3/12
Radiohead	OK Computer	Alternative	0.08	1/12
Antonio Vivaldi	Seasons	Classical	0.83	10/12
Frederic Chopin	Nocturnes	Classical	1.00	19/19
Johnny Cash	At Folsom Prison	Country	0.12	2/16
Dolly Parton	Jolene	Country	0.50	5/10
Daft Punk	Homework	Electronic	0.81	13/16
Aphex Twin	. . . I Care Because You Do	Electronic	0.92	11/12
Kendrick Lamar	good kid, m.A.A.d city	Hip-Hop/Rap	1.00	17/17
Kanye West	My Beautiful Dark Twisted Fantasy	Hip-Hop/Rap	0.77	10/13
Miles Davis	Kind of Blue	Jazz	0.60	3/5
Stan Getz, Joao Gilberto	Getz/Gilberto	Jazz	0.80	8/10
Led Zeppelin	IV	Rock	0.62	5/8
Pink Floyd	The Dark Side of the Moon	Rock	0.40	4/10

VII. Conclusion

In the end, our exploratory analysis and modeling efforts revealed many trends in the data which would need to be accounted for if Spotify were to implement some form of automated genre classification. Parametric models provided a relatively weak performance; Linear and Quadratic Discriminant Analysis assumptions are violated by the majority of features not following Gaussian distributions, while Naive Bayes falls victim to strong correlations between certain features as well as the aforementioned non-Gaussian distributions. K-Nearest Neighbors seems to outperform parameter-based models, but we still found it to be weaker than using tree-based methods with Random Forest modeling. Despite being our strongest model, using a Random Forest Classifier was still far from perfect; it still resulted in a test error rate of 0.45534 and specifically struggled to separate Hip-Hop and Rap from one another.

Notably, our Random Forest model found track popularity to be far and away the most important variable for classifying genre. This is concerning for the viability of automated track classification immediately upon upload, as it implies that the genres listed on Spotify are not defined strictly by the contents of the track itself. Popularity is not a metric which is inherent to a track or able to be generated by an algorithmic confidence measure; engagement measures are recorded after a track has been uploaded and viewed, at which point the track has likely been assigned a genre already. This also implies that certain Spotify genres with consistent popularity but large variance in their audio features, such as Anime, may be extremely difficult to separate from other tracks on the basis of audio features alone.

It is also worth considering the viability of genre classification focused on associating a track with a single genre of music. The data we worked with only included ten genres and assigned a single genre to each track. Spotify, however, contains far more than ten genre labels and allows tracks to be assigned to multiple genres. Implementing a model which can only assign a single genre to a single track may still be helpful for ensuring every track has at least one label, but it is very limited in scope compared to the robust selection of genre combinations which exist and may still fall short if too many single genres exist.

On a final practical note, Spotify also features audio tracks which are not music, such as podcasts and audio books. As it stands, classification of non-music into genres would be impossible; Spotify is able to distinguish music from non-music through the speechiness variable, but has no other generated metrics designed to distinguish between types of non-musical tracks. Spotify would likely need to create new audio features to address this gap in audio classification potential.

In conclusion, we found music genre classification to be a challenging task with a surprising level of depth. Future classification attempts, especially those experimenting with additional variables and genre labels, would likely find the most success following methods which do not rely on parameters to do so.

References

https://www.kaggle.com/datasets/vicsuperman/prediction-of-music-genre

https://developer.spotify.com/documentation/web-api/reference/get-audio-features

https://spotipy.readthedocs.io

https://github.com/simon-th/spotify-data-project

Appendix: Additional Code Used

Part III

Import and Cleaning

```
library(tidyverse)
library(randomForest)
library(e1071)
library(MASS)
library("class")

music <- read_csv("music_genre.csv", show_col_types = F) |>
  filter(tempo != "?", duration_ms != -1) |>
  dplyr::select(-c("instance_id", "artist_name", "track_name", "obtained_date")) |>
  mutate(tempo = as.numeric(tempo),
         mode = as.logical(ifelse(mode == "Minor", F, T)),
         key = fct(key),
         music_genre = fct(music_genre, levels = c("Alternative", "Anime", "Blues",
                                    "Classical", "Country", "Electronic",
                                    "Hip-Hop", "Jazz", "Rap", "Rock")))

music <- music[sample(nrow(music)),]
music_train <- music[1:32850,]
music_test <- music[32851:36500,]
```

Part IV

Histograms

```
library(patchwork)

music_hist <- function(feature) {
  p <- music |>
    ggplot(aes(x = as.numeric(unlist(feature)))) +
    geom_histogram(color = "black", fill = "#1DB954", bins = 15) +
    theme_minimal() +
    xlab("") +
    ylab("") +
    ggtitle(str_to_title(colnames(feature))) +
    theme(plot.title = element_text(size = 8), axis.text.y = element_text(size = 6),
          axis.text.x = element_text(size = 6, angle = 90),
          plot.margin = unit(c(0,0,0,0), "cm"))
}

hists <- list()
```

```
for(i in 1:(ncol(music)-1)) {
  if((i != 7)) {
    hists[[i]] <- music_hist(music[,i])
  }
}

hists[[10]] <- music |>
  ggplot(aes(x = fct_infreq(ifelse(mode, "Major", "Minor")))) +
  geom_bar(color = "black", fill = "#1DB954") +
  theme_minimal() +
  xlab("") +
  ylab("") +
  ggtitle("Mode")  +
  theme(plot.title = element_text(size = 8), axis.text.y = element_text(size = 6),
        axis.text.x = element_text(size = 6, angle = 0),
        plot.margin = unit(c(0,0,0,0), "cm"))

hists[[1]] + hists[[2]] + hists[[3]] + hists[[4]] + hists[[5]] + hists[[6]] +
  hists[[8]] + hists[[9]] + hists[[10]] + hists[[11]] + hists[[12]] + hists[[13]] +
  plot_layout(ncol = 4)
```

Correlation

```
library(corrplot)

corrplot(cor(music_numerical), method = "color", col = COL2('PRGn'),
         tl.cex = 0.7, tl.srt = 45, tl.offset = 1, tl.col = "black",
         cl.cex=0.5)
```

Boxplots

```
music_boxplot <- function(feature) {
  y <- as.numeric(unlist(feature))
  p <- music |>
    mutate(genre = reorder(genre, y, median, decreasing = TRUE)) |>
    ggplot(aes(x = genre, y = y, fill = genre)) +
    geom_boxplot(show.legend = FALSE, outlier.size=0.3) +
    theme_minimal() +
    theme(plot.title = element_text(size = 10),
          axis.text.x = element_text(angle = 45, vjust = 1, hjust=1, size = 8)) +
    scale_fill_viridis_d(option = "G", direction = -1) +
    ylab("") +
    xlab("") +
    ggtitle(str_to_title(colnames(feature)))

  return(p)
}

bp <- list()

for(i in 1:(ncol(music)-1)) {
  if(i == 4) {
    bp[[i]] <- (music_boxplot(log(music[,i])) + ggtitle("log(duration_ms)"))
  }
```

```
  else if((i != 7) && (i != 10)) {
    bp[[i]] <- music_boxplot(music[,i])
  }
}

bp[[10]] <- music |>
  group_by(genre) |>
  summarize(mean_mode = mean(mode)) |>
  mutate(genre = fct_infreq(genre, mean_mode)) |>
  ggplot(aes(x = genre, y = mean_mode, fill = mean_mode)) +
  geom_col(color = "black", show.legend = FALSE) +
  theme_minimal() +
  theme(plot.title = element_text(size = 10),
        axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1, size = 8)) +
  scale_fill_viridis_c(option = "G") +
  ylab("") +
  xlab("") +
  ggtitle("mean(mode)")

bp[[1]] + bp[[2]] + bp[[3]] + bp[[4]] + bp[[5]] + bp[[6]] +
  bp[[8]] + bp[[9]] + bp[[10]] + bp[[11]] + bp[[12]] + bp[[13]] +
  plot_layout(ncol = 4)
```

PCA & MDS

```
library(ggfortify)

log_dur_music <- music |>
  mutate(duration_ms = log(duration_ms)) |>
  mutate(genre = factor(genre, levels = c("Alternative", "Anime", "Blues",
                                          "Classical", "Country", "Electronic",
                                          "Hip-Hop", "Jazz", "Rap", "Rock")))

music_pc <- princomp(scale(log_dur_music[-c(7, 14)]))

pca_plot <- autoplot(music_pc, data = log_dur_music, color = "genre", alpha = 0.2) +
  facet_wrap(~genre, ncol = 5) +
  theme_minimal() +
  theme(legend.position = "none", axis.text = element_text(size = 5))

music_means <- music |>
  group_by(genre) |>
  summarize(across(everything(), mean)) |>
  dplyr::select(-c("key")) |>
  arrange(genre)

music_mds <- data.frame(cmdscale(dist(scale(music_means[,-1]))))

mds_plot <- ggplot(music_mds, aes(x = -X1, y = X2)) +
  geom_point(size = 1.5, fill = "#1DB954", shape = 22) +
  geom_text(label=music_means$genre, size = 3, hjust = 0, nudge_x = .25) +
  theme_minimal() +
  theme(axis.text = element_text(size = 6)) +
  xlab("") +
```

```
  ylab("") +
  xlim(-7.5, 3.5)
pca_plot + mds_plot


library(knitr)
loadings <- t(music_pc$loadings[1:12,1:2])
colnames(loadings) <- str_trunc(colnames(loadings), width = 8, ellipsis = "..")
kable(loadings, digits = 2)
```

Part V

Test Error and Confusion Matrices

```
nb <- naiveBayes(music_genre~., data = music_train)
sprintf("Test Error Rate: %.5f",mean(predict(nb, newdata=music_test) != music_test$music_genre))
addmargins(table(predictions = predict(nb, music_test), truth = music_test$music_genre))

lda_mod <- lda(music_genre ~ ., data = music_train)
sprintf("Test Error Rate: %.5f",mean(predict(lda_mod, music_test)[[1]] != music_test$music_genre))
addmargins(table(predictions = predict(lda_mod, music_test)[[1]], truth = music_test$music_genre))


qda_mod <- qda(music_genre ~ ., data = music_train)
sprintf("Test Error Rate: %.5f",mean(predict(qda_mod, newdata=music_test)[[1]] != music_test$music_genre
addmargins(table(truth = predict(qda_mod, music_test)[[1]], predicted = music_test$music_genre))

mod_knn <- knn(scale(music_train[,c(-7, -10, -14)]),
               scale(music_test[,c(-7, -10, -14)]), cl=music_train$music_genre, k=50)
sprintf("Test Error Rate: %.5f", mean(mod_knn != music_test$music_genre))
addmargins(table(predicted = mod_knn, truth = music_test$music_genre))

rfc <- randomForest(music_genre~., data = music_train, mtry = 6, ntree = 100)
sprintf("Test Error Rate: %.5f",mean(music_test$music_genre != predict(rfc, music_test)))
addmargins(table(prediction = predict(rfc, music_test), truth = music_test$music_genre))
```

Variable Importance Plot

```
par(mar=rep(0,4) + 1.2)
varImpPlot(rfc, sort=TRUE, main="Variable Importance Plot", cex = 0.65)
```

Summary Table

```
errors <- tibble(
  NB = mean(predict(nb, newdata = music_test) != music_test$music_genre),
  LDA = mean(predict(lda_mod, music_test)[[1]] != music_test$music_genre),
  QDA = mean(predict(qda_mod, music_test)[[1]] != music_test$music_genre),
  KNN = mean(mod_knn != music_test$music_genre),
  RFC = mean(music_test$music_genre != predict(rfc, music_test))
)

kable(errors, digits=3)
```

Part VI

Python Script for album data

```
import spotipy
import pandas as pd
from spotipy.oauth2 import SpotifyClientCredentials

client_id = # censored
client_secret = # censored

client_credentials_manager = SpotifyClientCredentials(client_id=client_id, client_secret=client_secret)
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)


# ALBUMS:
album_ids = ["4eEJooUkzenp7lZTtfbIP7", # Vivaldi's Seasons
             "5Q5m22P7jIwdMSKXqhrpaN", # Chopin's Nocturnes
             "3DGQ1iZ9XKUQxAUWjfC34w", # Good Kid Maad City
             "20r762YmB5HeofjMCiPMLv", # MBDTF
             "5n52kyQKeUZs5ObZJejLQd", # Achtung Baby
             "6dVIqQ8qmQ5GBnJ9shOYGE", # OK Computer
             "5EyIDBAqhnlkAHqvPRwdbX", # Led Zeppelin IV
             "4LH4d3cOWNNsVw41Gqt2kv", # DSOTM
             "1weenld61qoidwYuZ1GESA", # Kind of Blue
             "73ZRKdD3Ds43IjHrhKgucY", # Getz Gilberto
             "4TJIdlY9hGSSTO1kUs1neh", # At Folsom Prison
             "5DyOxuvdSmTSNAmkfcsBsj", # Jolene
             "5uRdvUR7xCnHmUW8n64n9y", # "Homework"
             "6TmEZKJtPJ9mPsMBmyteCE"] # I Care Because You Do

n = 1
for album in album_ids:
    track_ids = []
    track_titles = []

    tracks = sp.album_tracks(album_id = album)
    for i in range(len(tracks['items'])):
        track_ids.append(tracks['items'][i]['id'])
        track_titles.append(tracks['items'][i]['name'])

    features = sp.audio_features(track_ids)

    features_df = pd.DataFrame(data=features, columns=features[0].keys())

    features_df['title'] = track_titles
    features_df.to_csv('features' + str(n) + '.csv')
    n += 1
```

Accuracy Table

```
artists = c("Antoni Vivaldi", "Frederic Chopin", "Kendrick Lamar", "Kanye West", "U2", "Radiohead",
            "Led Zeppelin", "Pink Floyd", "Miles Davis", "Stan Getz, Joao Gilberto", "Johnny Cash",
            "Dolly Parton", "Daft Punk", "Aphex Twin")
albums = c("Seasons", "Nocturnes", "good kid, m.A.A.d city",
           "My Beautiful Dark Twisted Fantasy", "Achtung Baby", "OK Computer", "IV",
           "The Dark Side of the Moon","Kind of Blue", "Getz/Gilberto", "At Folsom Prison",
           "Jolene", "Homework", "...I Care Because You Do")
```

```r
genres = c("Classical", "Classical", "Hip-Hop/Rap", "Hip-Hop/Rap",
           "Alternative", "Alternative", "Rock", "Rock", "Jazz", "Jazz",
           "Country", "Country", "Electronic", "Electronic")

u2tracks <- read_csv(paste0("features5.csv")) |>
  mutate(mode = fct(ifelse(mode == 1, "Major", "Minor")),
         popularity = 0,
         key = fct(levels(music$key)[key+1]))

levels(u2tracks$key) <- levels(music$key)

u2predictions <- predict(rf, newdata = u2tracks)

u2results <- u2tracks |>
  dplyr::select(title) |>
  mutate(Prediction = u2predictions) |>
  rename(`Song Title` = title)

predictions <- c()
accuracy <- c()
fraction <- c()

for(i in 1:14) {
  test_tracks <- read_csv(paste0("features", i, ".csv")) |>
    mutate(mode = fct(ifelse(mode == 1, "Major", "Minor")),
           key = fct(levels(music$key)[key+1]),
           popularity = 0)

  levels(test_tracks$key) <- levels(music$key)

  if(i %in% c(3, 4)) {
    predictions[[i]] <- predict(rf, newdata = test_tracks)
    accuracy[i] <- mean(predictions[[i]] == "Rap" | predictions[[i]] == "Hip-Hop")
    fraction[i] <- paste0(sum(predictions[[i]] == "Rap" | predictions[[i]] == "Hip-Hop"),
                          "/", nrow(test_tracks))
  } else {
    predictions[[i]] <- predict(rf, newdata = test_tracks)
    accuracy[i] <- mean(predictions[[i]] == genres[i])
    fraction[i] <- paste0(sum(predictions[[i]] == genres[i]), "/", nrow(test_tracks))
  }
}

album_results <- tibble(Artist = artists, Album = albums, Genre = genres,
                        Accuracy = round(accuracy, 2), Fraction = fraction) |>
  arrange(Genre)

kable(u2results)
kable(album_results)
```