Final Project PSTAT131

Spotify songs as of 2023

Parker Reedy and Chris Zhao

December 11, 2023

Data description

This dataset contains a list of the most famous songs as listed on Spotify in 2023 found on the website Kaggle.com. The dataset offers a wealth of features beyond what is typically available in similar datasets. It provides insights into each song's attributes, popularity, and presence on various music platforms. The dataset includes information such as track name, artist(s) name, release date, Spotify playlists and charts, streaming statistics, Apple Music presence, Deezer presence, Shazam charts, and various audio features. Look at the following infomation for details on the 24 total variables in the data set.

- track_name: Name of the song
- artist_name: Name of the artist(s) of the song
- artist_count: Number of artists contributing to the song
- released_year: Year when the song was released
- released_month: Month when the song was released
- released_day: Day of the month when the song was released
- in_spotify_playlists: Number of Spotify playlists the song is included in
- in_spotify_charts: Presence and rank of the song on Spotify charts
- streams: Total number of streams on Spotify
- in_apple_playlists: Number of Apple Music playlists the song is included in
- in_apple_charts: Presence and rank of the song on Apple Music charts
- in_deezer_playlists: Number of Deezer playlists the song is included in
- in_deezer_charts: Presence and rank of the song on Deezer charts
- in_shazam_charts: Presence and rank of the song on Shazam charts
- bpm: Beats per minute, a measure of song tempo

- key: Key of the song
- mode: Mode of the song (major or minor)
- danceability_%: Percentage indicating how suitable the song is for dancing
- valence_%: Positivity of the song's musical content
- energy_%: Perceived energy level of the song
- acousticness_%: Amount of acoustic sound in the song
- instrumentalness_%: Amount of instrumental content in the song
- liveness_%: Presence of live performance elements
- speechiness_%: Amount of spoken words in the song

Question of Interest

One of the most important parts of making music is for it to be heard by everyone. In this analysis, we attempt to answer the question of what variables have the most impact in predicting the number of streams (or listens) a Spotify song has. Because we are working with a numeric response variable, we will be using regression to understand these relationships.

Data Cleaning

First, read in our data from csv file.

```
dataset <- read_csv('spotify-2023.csv', show_col_types = FALSE)
head(dataset)
```

```
## # A tibble: 6 x 24
##   track_name          `artist(s)_name` artist_count released_year released_month
##   <chr>               <chr>                  <dbl>         <dbl>          <dbl>
## 1 Seven (feat. Latto~ Latto, Jung Kook           2          2023              7
## 2 LALA                Myke Towers                1          2023              3
## 3 vampire             Olivia Rodrigo             1          2023              6
## 4 Cruel Summer        Taylor Swift               1          2019              8
## 5 WHERE SHE GOES      Bad Bunny                  1          2023              5
## 6 Sprinter            Dave, Central C~           2          2023              6
## # i 19 more variables: released_day <dbl>, in_spotify_playlists <dbl>,
## #   in_spotify_charts <dbl>, streams <chr>, in_apple_playlists <dbl>,
## #   in_apple_charts <dbl>, in_deezer_playlists <dbl>, in_deezer_charts <dbl>,
## #   in_shazam_charts <dbl>, bpm <dbl>, key <chr>, mode <chr>,
```

```
## #   `danceability_%` <dbl>, `valence_%` <dbl>, `energy_%` <dbl>,
## #   `acousticness_%` <dbl>, `instrumentalness_%` <dbl>, `liveness_%` <dbl>,
## #   `speechiness_%` <dbl>
```

```
spotify <- data.frame(dataset)

missing_columns <- spotify %>% is.na() %>% colSums()
names(missing_columns[missing_columns != 0])
```

```
## [1] "in_shazam_charts" "key"
```

As we can see, there are only missing values in the variables `in_shazam_chart` and `key`. For the Shazam charts, the missing values represent the song not being on the chart so we will replace them with zeros. There are also commas in this variable so we want to remove them.

```
spotify$in_shazam_charts[is.na(spotify$in_shazam_charts)] <- 0

spotify$in_shazam_charts <- gsub(',', '', spotify$in_shazam_charts)
spotify$in_shazam_charts <- as.numeric(spotify$in_shazam_charts)

missing_columns2 <- spotify %>% is.na() %>% colSums()
names(missing_columns2[missing_columns2 != 0])
```

```
## [1] "key"
```

For the missing `key` values, since `key` might be an important part of song analysis, we will remove the observations with missing values and because it is the only variable with missing values, we can just use `na.omit()`.

```
spotify <- na.omit(spotify)
```

Let's Check the type of variables before analysis.

```
str(spotify)
```

```
## 'data.frame':    858 obs. of  24 variables:
##  $ track_name          : chr  "Seven (feat. Latto) (Explicit Ver.)" "LALA" "vampire" "Cruel Summer"
##  $ artist.s._name      : chr  "Latto, Jung Kook" "Myke Towers" "Olivia Rodrigo" "Taylor Swift" ...
##  $ artist_count        : num  2 1 1 1 1 2 2 1 1 2 ...
##  $ released_year       : num  2023 2023 2023 2019 2023 ...
##  $ released_month      : num  7 3 6 8 5 6 3 7 5 3 ...
##  $ released_day        : num  14 23 30 23 18 1 16 7 15 17 ...
##  $ in_spotify_playlists: num  553 1474 1397 7858 3133 ...
##  $ in_spotify_charts   : num  147 48 113 100 50 91 50 43 83 44 ...
##  $ streams             : chr  "141381703" "133716286" "140003974" "800840817" ...
##  $ in_apple_playlists  : num  43 48 94 116 84 67 34 25 60 49 ...
##  $ in_apple_charts     : num  263 126 207 207 133 213 222 89 210 110 ...
##  $ in_deezer_playlists : num  45 58 91 125 87 88 43 30 48 66 ...
##  $ in_deezer_charts    : num  10 14 14 12 15 17 13 13 11 13 ...
##  $ in_shazam_charts    : num  826 382 949 548 425 946 418 194 953 339 ...
##  $ bpm                 : num  125 92 138 170 144 141 148 100 130 170 ...
```

```
##  $ key                : chr  "B" "C#" "F" "A" ...
##  $ mode               : chr  "Major" "Major" "Major" "Major" ...
##  $ danceability_.     : num  80 71 51 55 65 92 67 67 85 81 ...
##  $ valence_.          : num  89 61 32 58 23 66 83 26 22 56 ...
##  $ energy_.           : num  83 74 53 72 80 58 76 71 62 48 ...
##  $ acousticness_.     : num  31 7 17 11 14 19 48 37 12 21 ...
##  $ instrumentalness_. : num  0 0 0 0 63 0 0 0 0 0 ...
##  $ liveness_.         : num  8 10 31 11 11 8 8 11 28 8 ...
##  $ speechiness_.      : num  4 4 6 15 6 24 3 4 9 33 ...
##  - attr(*, "na.action")= 'omit' Named int [1:95] 13 18 23 36 45 47 59 60 125 128 ...
##   ..- attr(*, "names")= chr [1:95] "13" "18" "23" "36" ...
```

We can see that `streams` is a character variable so we need to change that to numeric.

```
spotify$streams <- as.numeric(spotify$streams)
```

```
## Warning: NAs introduced by coercion
```

It seems that we get an error saying that NAs were introduced when setting streams to numeric.

```
dataset[575, 9]
```

```
## # A tibble: 1 x 1
##   streams
##   <chr>
## 1 BPM110KeyAModeMajorDanceability53Valence75Energy69Acousticness7Instrumentalne~
```

By checking our row data, it seems like an input error, so we will just remove the observation.

```
spotify <- na.omit(spotify)

#sum of the number of missing values from each column
sum(spotify %>% is.na() %>% colSums())
```

```
## [1] 0
```

Now we have no missing values and all the categories of our data are correct. 857 total observations left. For the sake of simplicity, we are going to log the streams values to make analysis easier.

```
spotify$streams <- log(spotify$streams) # apply log
```

```
spotify_num <- spotify[-1]

spotify_num$mode <- as.numeric(as.factor(spotify_num$mode))
spotify_num$key <- as.numeric(as.factor(spotify_num$key))
spotify_num$artist.s._name <- as.numeric(as.factor(spotify_num$artist.s._name))
```

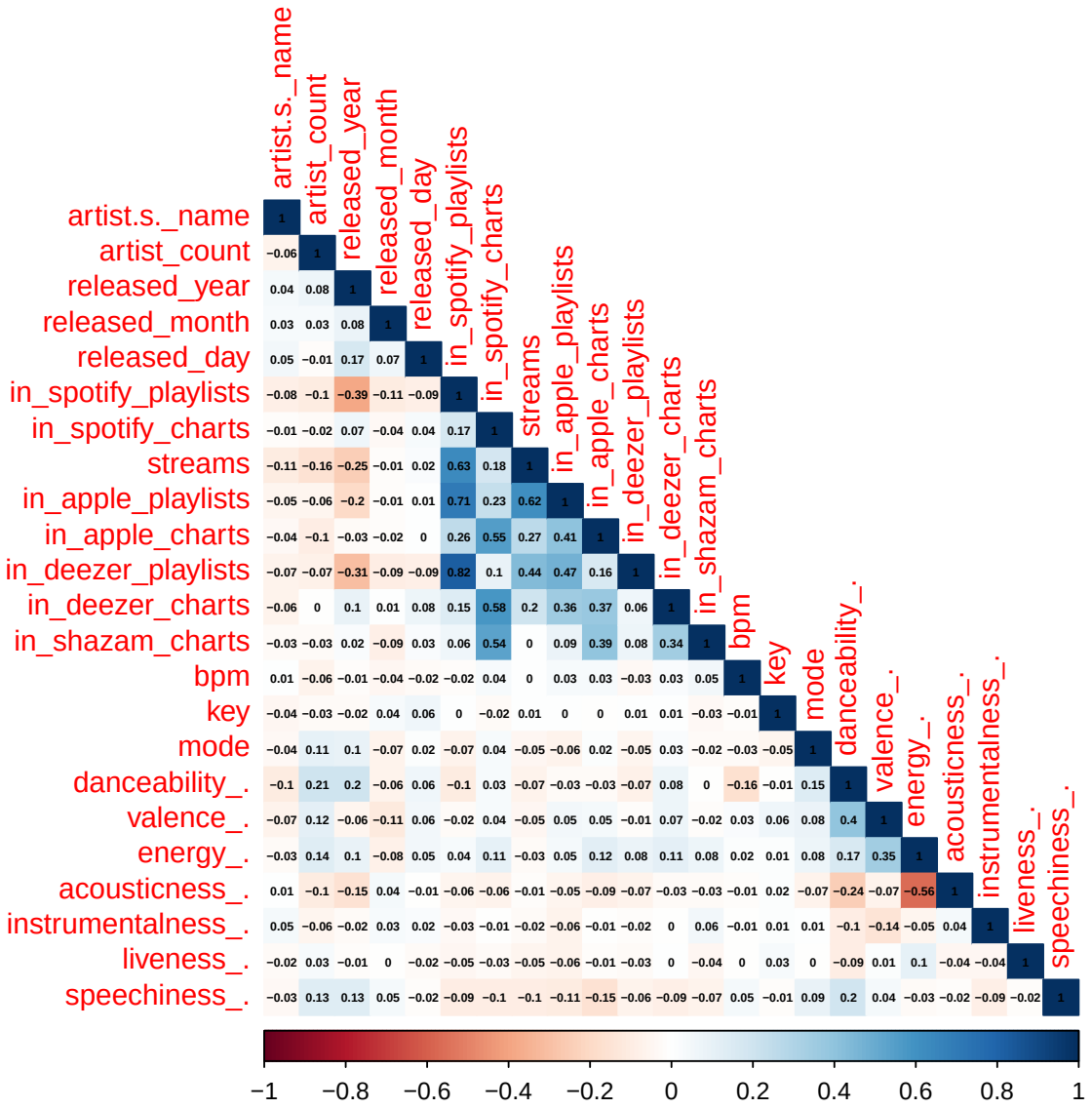
Encode `mode`, `key` and etc to factor so that we can use it as predictor.

Now that the data is tidy, it is ready to be analyzed.

Data Analysis

First, we are going to start with a correlation matrix to plot the correlations between each variable so we can better understand our data.

```
#create the correlation matrix and heatmap
cor_matrix <- cor(spotify_num)
corrplot(cor_matrix, method = "color", type = 'lower',
         number.cex = 0.45, addCoef.col = 'black')
```



The correlation plot use color and value to show the strength of the correlation between all variables except the name of songs. Values closer to 1 appear as a darker blue while values closer to -1 appear as a darker orange. Now, we want to focus on our most interested variable, streams.

```
streams_cor <- cor_matrix["streams",]
streams_cor
```

```
##       artist.s._name         artist_count        released_year
##            -0.107044            -0.158497            -0.247515
##       released_month         released_day in_spotify_playlists
##            -0.014171             0.020266             0.631340
##    in_spotify_charts              streams   in_apple_playlists
##             0.181082             1.000000             0.616928
##      in_apple_charts  in_deezer_playlists     in_deezer_charts
##             0.266153             0.437952             0.203632
##     in_shazam_charts                  bpm                  key
##             0.001252             0.004482             0.009051
##                 mode       danceability_.            valence_.
##            -0.048689            -0.067557            -0.048348
##             energy_.       acousticness_.   instrumentalness_.
##            -0.027329            -0.012694            -0.020787
##           liveness_.        speechiness_.
##            -0.054727            -0.097324
```

Filter out those that are too weakly related.

```
streams_cor[abs(streams_cor) > 0.2 & streams_cor != 1]
```

```
##        released_year in_spotify_playlists   in_apple_playlists
##              -0.2475               0.6313               0.6169
##      in_apple_charts  in_deezer_playlists     in_deezer_charts
##               0.2662               0.4380               0.2036
```

Here are all variable whose absolute values of correlation coefficient with `streams` are greater than 0.2. For these values great than 0.2, the strength of their association are not regarded as very weak. So, let's prioritize the few parameters shown above.

Principal Components Analysis

```
pc.out <- prcomp(spotify_num, scale=TRUE)
```

```
summary(pc.out)
```

```
## Importance of components:
##                           PC1   PC2    PC3    PC4    PC5    PC6   PC7    PC8
## Standard deviation     1.890 1.542 1.3855 1.1410 1.1032 1.0677 1.040 1.0298
## Proportion of Variance 0.155 0.103 0.0835 0.0566 0.0529 0.0496 0.047 0.0461
## Cumulative Proportion  0.155 0.259 0.3421 0.3987 0.4516 0.5011 0.548 0.5942
##                           PC9   PC10   PC11   PC12   PC13   PC14   PC15  PC16
## Standard deviation     0.9912 0.9789 0.9531 0.9416 0.9207 0.8935 0.8608 0.774
## Proportion of Variance 0.0427 0.0417 0.0395 0.0386 0.0369 0.0347 0.0322 0.026
## Cumulative Proportion  0.6369 0.6786 0.7181 0.7566 0.7935 0.8282 0.8604 0.886
```

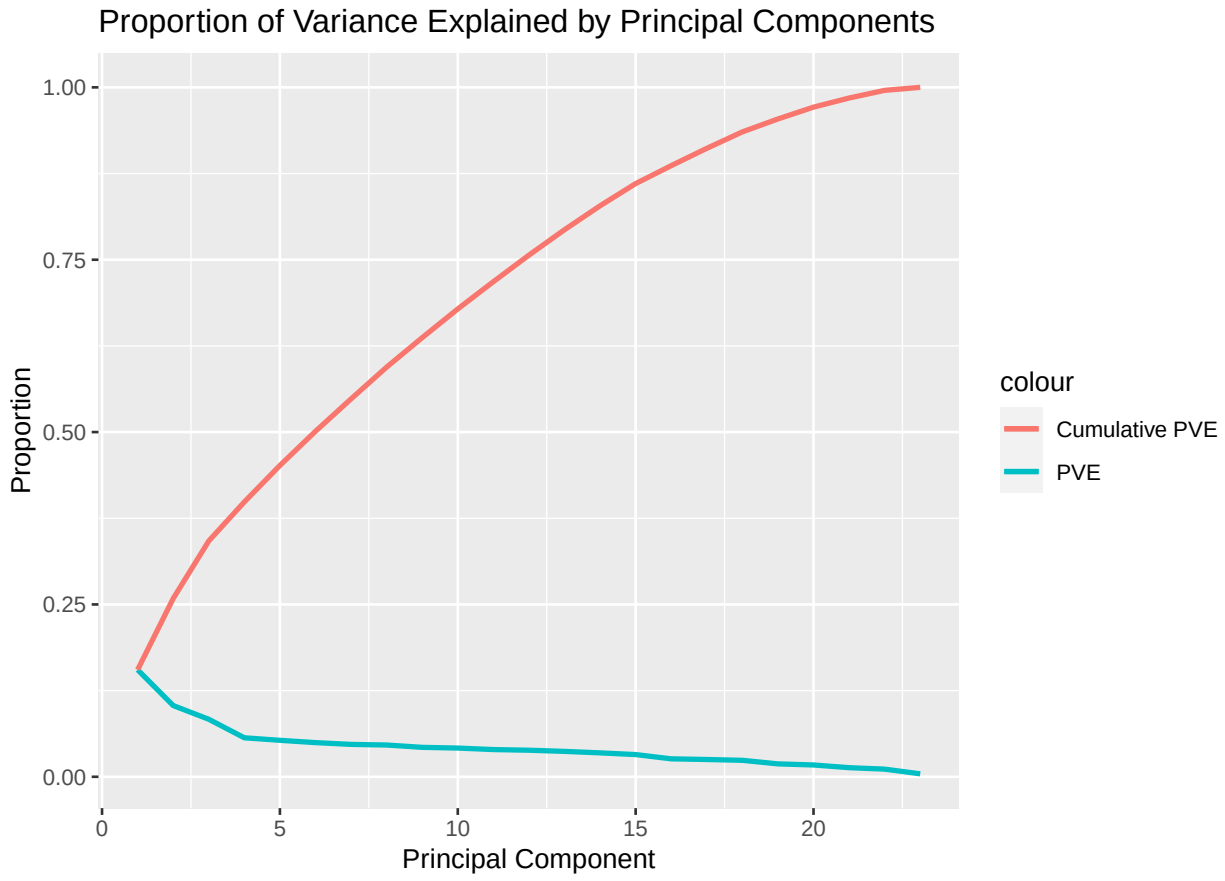
```
##                           PC17   PC18   PC19   PC20   PC21   PC22    PC23
## Standard deviation     0.7595 0.7413 0.6562 0.6282 0.5517 0.5059 0.31545
## Proportion of Variance 0.0251 0.0239 0.0187 0.0172 0.0132 0.0111 0.00433
## Cumulative Proportion  0.9115 0.9354 0.9542 0.9713 0.9846 0.9957 1.00000
```

We see that the first principal component explains 15.5% of the variance in the data, the next principal component explains 10.3% of the variance, and so forth.

We can plot the PVE explained by each component, as well as the cumulative PVE, as follow

```
pc.summary <- summary(pc.out)$importance %>%
  t() %>%
  as.data.frame()
```

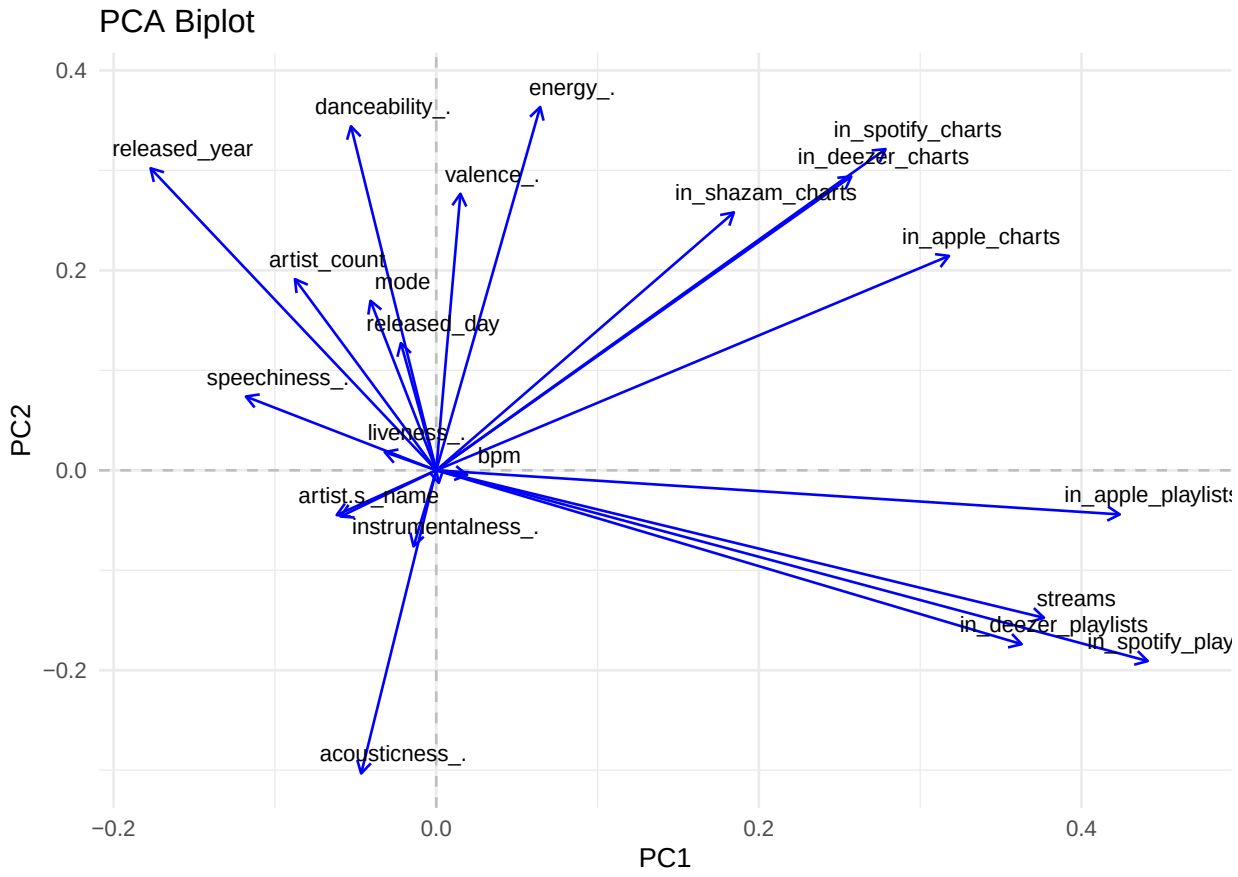
```
ggplot(pc.summary, aes(x = 1:nrow(pc.summary))) +
  geom_line(aes(y = `Proportion of Variance`, color = 'PVE'), linewidth = 1) +
  geom_line(aes(y = `Cumulative Proportion`, color = 'Cumulative PVE'), linewidth = 1) +
  labs(x = 'Principal Component', y = 'Proportion',
       title = 'Proportion of Variance Explained by Principal Components')
```




```
pc.biplot <- as.data.frame(pc.out$rotation[, 1:2])  # Extract the first two principal components
pc.biplot$variable <- rownames(pc.biplot)  # Add variable names as a new column

#create the plot
```

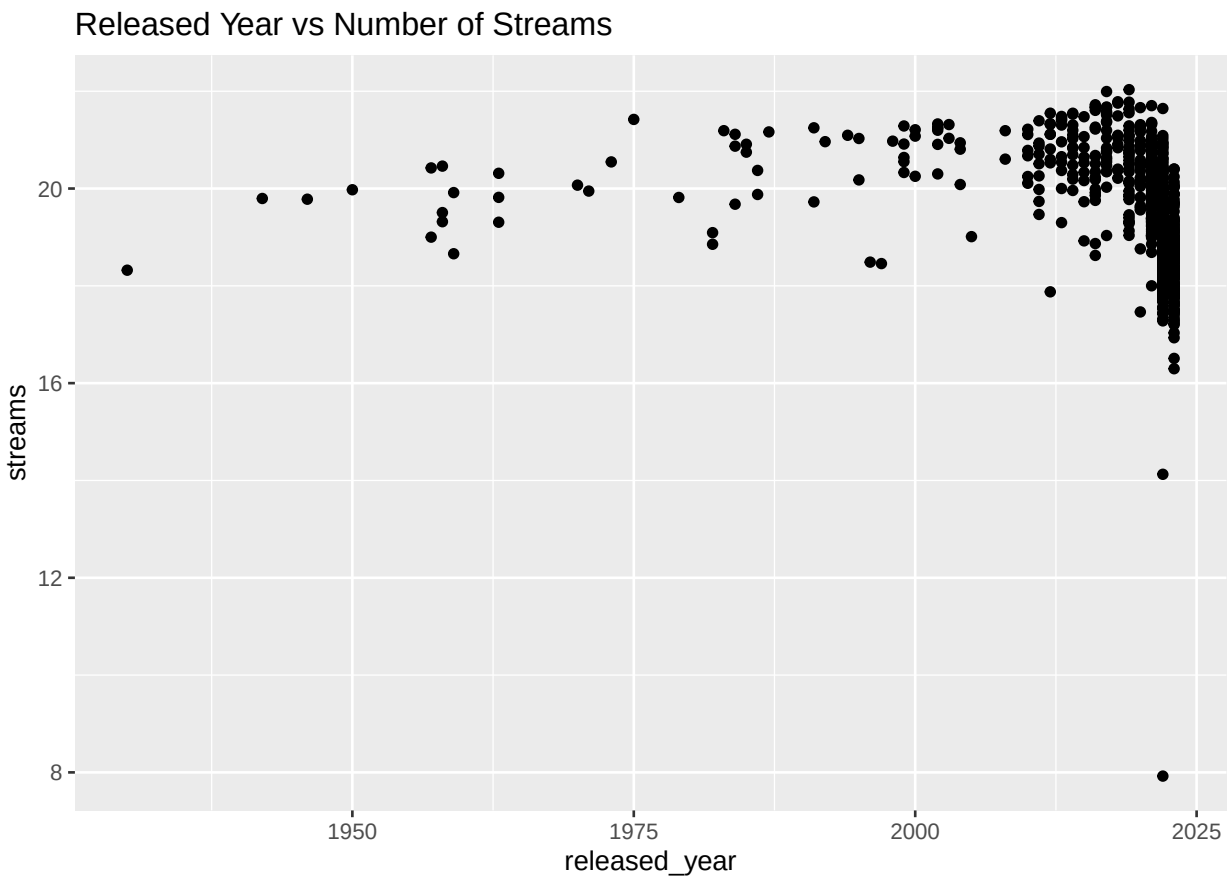
```
ggplot(pc.biplot, aes(x = PC1, y = PC2, label = variable)) +
  geom_hline(yintercept = 0, linetype = "dashed", color = "gray") +
  geom_vline(xintercept = 0, linetype = "dashed", color = "gray") +
  geom_segment(aes(x = 0, y = 0, xend = PC1, yend = PC2), arrow = arrow(length = unit(0.2, "cm")), colo
  geom_text(nudge_x = 0.02, nudge_y = 0.02, check_overlap = TRUE, size = 3) +
  labs(title = "PCA Biplot") +
  theme_minimal()
```



From this plot of PCA1 vs PCA2 above, we can see that many of out variables of importance are have vectors pointing towards a similar direction and have a similar length which suggests that these variables are positively correlated with each other and contribute similarly to the Principal Components. If we look at the streams vector pointing to the bottom right, we notice that the vectors pointing in the opposite direction have a negative correlation with streams as seen in the correlation plot earlier. Now, lets make some models to help us predict the amount of streams a spotify song has.

From the PCA plot and the correlation heatmap, Released_year is an interesting variable because it seems to have a negative correlation with Streams meaning that an increase in year indicates a decrease in the number of streams.

```
ggplot(spotify, aes(x = released_year, y = streams)) + geom_point() + labs(
  title = 'Released Year vs Number of Streams'
)
```



From this plot, we can see that The number of streams of a song that came out very recently is not very high which intuitively makes sense. But not including the most recent years, the plot seems to be slightly positively correlated. The amount of streams increases as the year increases. The reason for the negative correlation is that the most recent songs have not had enough time to obtain the true number of streams. This is just something to keep in mind for future predictions. It is important to note that the number of streams for songs in current years will go up in future updated datasets but similar observations will be seen for newer songs.

Simple Linear Regression Model

To start off, lets fit a simple Linear Regression model to a training set and use it to predict a test set.

```
#set seed for reproducability
set.seed(123)

songs <- spotify %>% select(-track_name, -artist.s._name)

#training with 500, test with the rest
training = sample(1:nrow(songs), 500)
songs.train = songs[training,]
songs.test = songs[-training,]

SpotYTrain = songs.train$streams
SpotXTrain = songs.train %>% select(-streams)
```

```r
SpotYTest = songs.test$streams
SpotXTest = songs.test %>% select(-streams)


#creating the model will all predictors
mod1 <- lm(streams~., data=songs.train)

#predict on test set
predictions <- predict(mod1, songs.test)

#find Test MSE for this model
mse <- mean((predictions - SpotYTest)^2)
mse
```

```
## [1] 0.635
```

Now we have a baseline MSE for a model with all predictors. For the last Linear model, lets only do the predictors that we found earlier to be useful.

```r
#set seed for reproducability
set.seed(123)

#create a linear model with less predictors
mod2 <- lm(streams~ in_spotify_playlists + in_apple_playlists + in_deezer_playlists + in_apple_charts +

#predict on test set
predictions2 <- predict(mod2, songs.test)

#find Test MSE for this model
mse2 <- mean((predictions - SpotYTest)^2)
mse2
```

```
## [1] 0.635
```

We can see that the MSE of the 2 models are very similar so lets take a look at the anova table to measure the usefulness of the predictors in the models.

```r
anova(mod2, mod1)
```

```
## Analysis of Variance Table
##
## Model 1: streams ~ in_spotify_playlists + in_apple_playlists + in_deezer_playlists +
##     in_apple_charts + in_deezer_charts + in_shazam_charts + in_spotify_charts
## Model 2: streams ~ artist_count + released_year + released_month + released_day +
##     in_spotify_playlists + in_spotify_charts + in_apple_playlists +
##     in_apple_charts + in_deezer_playlists + in_deezer_charts +
##     in_shazam_charts + bpm + key + mode + danceability_. + valence_. +
##     energy_. + acousticness_. + instrumentalness_. + liveness_. +
##     speechiness_.
##   Res.Df RSS Df Sum of Sq    F Pr(>F)
## 1    492 383
## 2    469 365 23      17.6 0.98   0.49
```

After looking at the anova table between the two models, it is clear that there is no significant difference between the predicted values because the Pr(>F) is .49 meaning that we fail to reject the null hypothesis that there is a difference between the models. The model with less predictors is just as important and by the Occams Razor principle, the model with less predictors is better. We will use this model as a baseline for testing MSE.

K-Nearest Neighbors Model

First, Lets make a subset with the variables that we are interested in and then find the best k value for K-Nearest neighbors.

```
playlist = spotify %>%
  select(streams, in_spotify_playlists, in_apple_playlists, in_deezer_playlists)

data <- playlist
target <- data$streams

k_folds <- 10

# Perform K-fold cross-validation with different K values
k_range <- 1:100
best_k <- 0
lowest_mse <- Inf

for (k in k_range) {
  # Train the KNN model for the current K value
  knn_model <- knn.reg(train = data, y = target, k = k)

  # Calculate the average MSE across folds
  mse_per_fold <- sapply(1:k_folds, function(fold) {
    # Extract training and testing data for the current fold
    train_index <- (1:(k_folds * fold)) %% k_folds + 1 != fold
    test_index <- (1:(k_folds * fold)) %% k_folds + 1 == fold

    # Predict on the testing data
    predictions <- knn_model$pred[-train_index]

    # Calculate the MSE for the current fold
    mean((predictions - target[-train_index])^2)
  })

  # Calculate the average MSE across all folds
  average_mse <- mean(mse_per_fold)

  # Check if the current K value has the lowest average MSE
  if (average_mse < lowest_mse) {
    lowest_mse <- average_mse
    best_k <- k
  }
}

cat("Best k:", best_k)
```

```
## Best k: 34
```

As we can see, the best K value seems to be 34. Lets use this to train a K-Nearest Neighbors model looking at just the playlists variables.

```
#set seed for reproducability
set.seed(123)

#Splitting data into train and test
train = sample(1:nrow(playlist), 500)
playlist.train = playlist[train,]
playlist.test = playlist[-train,]

#splitting data into X and Y / Train and Test
YTrain = playlist.train$streams
XTrain = playlist.train %>% select(-streams) %>% scale(center = TRUE, scale = TRUE)
YTest = playlist.test$streams
XTest = playlist.test %>% select(-streams) %>% scale(center = TRUE, scale = TRUE)


#Make model for training and test data using TRAINING data

pred.YTtrain = knn.reg(train=XTrain, test=XTrain, y=YTrain, k=best_k)
pred.YTest = knn.reg(train=XTrain, test=XTest, y=YTrain, k=best_k)

#Print results
print(paste(round(mean((pred.YTtrain$pred - YTrain)^2),4) , ': Training MSE'))
```

```
## [1] "0.5404 : Training MSE"
```

```
print(paste(round(mean((pred.YTest$pred - YTest)^2), 4), ': Test MSE'))
```

```
## [1] "0.4426 : Test MSE"
```

The Test MSE for this model is 0.4426. This model works better than both of the linear regression models we made previously when looking at the MSE.

Next, we want to fit a model that will allow us to test if more variables can help the Test MSE. The reason we want to do this is because K-nearest Neighbors works better with a small amount of predictors.

Bagging model and Random Forest model

Now, lets try to fit a bagging model and a random forest model

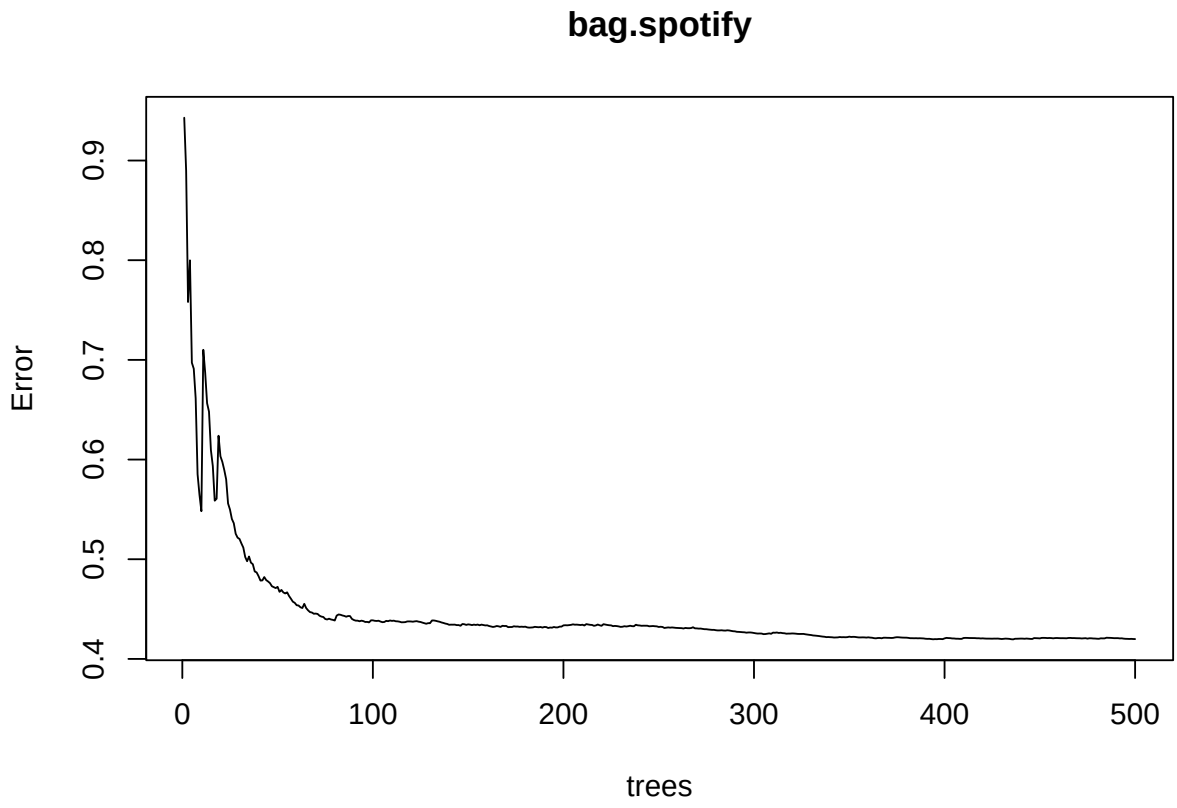
```
#Splitting data into test and train
set.seed(123)

#removing artist name and track name
subset2 <- spotify[, -c(1,2)]
#2nd training set
```

```
train = sample(nrow(subset2), 0.75*nrow(subset2))
train.spotify = subset2[train,]
test.spotify = subset2[-train,]
```

This is the bagged model because mtry = 21 which is the number of predictors in the model (m = p). This determines the number of predictors that should be considered for each split of the tree.

```
#fit a random forest model with m = p for a bagged model
bag.spotify = randomForest(streams ~ ., data=train.spotify,
                           mtry=21, importance=TRUE)
plot(bag.spotify)
```



This plot shows how the Error changes with the number of trees it makes. Now lets use the model to predict the MSE of this model.

```
yhat.bag = predict(bag.spotify, newdata = test.spotify)
mean((yhat.bag - test.spotify$streams)^2)
```

```
## [1] 0.2847
```

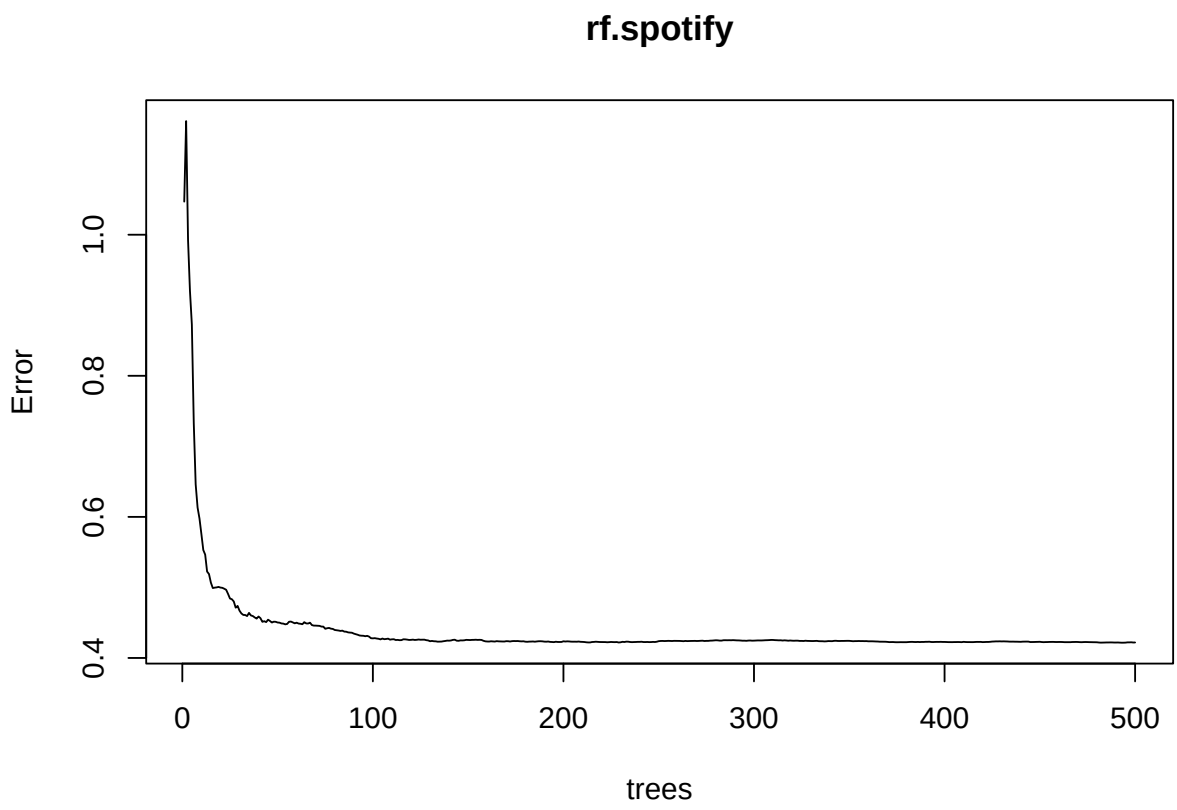
```
#show importance of variables
importance(bag.spotify)
```

```
##                       %IncMSE IncNodePurity
```

```
## artist_count          2.10501       1.6185
## released_year        24.89088      41.5636
## released_month       10.59058      23.1005
## released_day          3.80654       6.5021
## in_spotify_playlists 33.61000     518.1669
## in_spotify_charts    10.36226      28.7290
## in_apple_playlists    6.01047      17.0306
## in_apple_charts       1.48088      14.6642
## in_deezer_playlists  14.23221      39.5965
## in_deezer_charts     10.57832      35.4429
## in_shazam_charts      2.83961       7.1391
## bpm                  -0.08376      12.1933
## key                  -0.28041       5.6609
## mode                  1.91413       0.8947
## danceability_.        5.92373      14.4598
## valence_.             2.19390      13.5051
## energy_.              3.94622       6.7704
## acousticness_.        3.38893       9.5631
## instrumentalness_.    2.04520       1.4152
## liveness_.           -0.46561       5.8867
## speechiness_.         0.83613       5.1855
```

The MSE for this model is surprisingly low compared to the other models that we made so far. One last thing we should check is that if we use a slightly different method called Random Forest. This method is used to decorrelate the trees so that the prediction is less variable. Lets set mtry = 5 (chosen because usually m is chosen as the square root of the total number of predictors = sqrt(20)) which will test 5 random variables at each split in the tree.

```
#fit a random forest model
rf.spotify = randomForest(streams ~., data=train.spotify, mtry = 5, importance=TRUE)
plot(rf.spotify)
```



This is the Error for the Random Forest as it produces more trees. Finally, lets test the MSE for this model to see if its lower than previous ones.

```
#predict with the forest model
yhat.forest = predict(rf.spotify, newdata = test.spotify)
mean((yhat.forest - test.spotify$streams)^2)
```

```
## [1] 0.297
```

```
#show importance of variables
importance(rf.spotify)
```

```
##                      %IncMSE IncNodePurity
## artist_count         -0.81216         4.440
## released_year        25.08275       103.861
## released_month        1.93426        22.605
## released_day          2.15064        12.099
## in_spotify_playlists 23.97958       222.234
## in_spotify_charts     6.00404        27.558
## in_apple_playlists    7.28964        82.725
## in_apple_charts       3.73966        31.091
## in_deezer_playlists  12.13898       136.072
## in_deezer_charts      6.35867        25.485
## in_shazam_charts      4.56281        14.435
## bpm                   0.09820        21.195
```

```
## key                  -0.19615        12.513
## mode                  0.03217         2.266
## danceability_.        5.84771        21.409
## valence_.             3.69944        18.578
## energy_.              1.44682        11.140
## acousticness_.        3.33001        10.716
## instrumentalness_.    1.65510         2.584
## liveness_.            1.51801         8.810
## speechiness_.         0.67047         9.641
```

The MSE of .297 is actually very similar to the previous Bagging method. From all of our models, it seems that the Bagging and Random Forest model produces the best results. Also after using the `importance()` function, many of the variables we deemed important earlier are shown to have a large affect on the MSE. We can also see that released_year is an important variable in both of these models and from the plot we made earlier, we know that released_year does have an affect on the number of streams a song has.

Findings

From our analysis on the Spotify Dataset, we found that the most important variables in predicting the number of streams a song has is the in_playlists variables as well as the release year of the song. Intuitively this makes sense when thinking about the type of data we have. The more recent the song is, the less total streams it will have and the more playlists that the song is in, the more streams it will receive. After testing many models, we came to the conclusion that the best one was the tree based models which had the lowest MSE of all the models tested. One of the most important figures shown is the PCA plot which shows how the variables correlate with one another. We can see from this that all the playlist variables are in similar directions and the released_year (which negatively correlates with # of streams) is in the opposite direction. This visualizes many of the relationships in the data and provides insight as to which ones are important for predicting the number of streams. Mentioned earlier, Another important finding is that the release_year is only negatively correlated with the number of streams because more recent songs do not have enough time to gain the maximum number of streams because the data is taken at a specific point in time. Also, from the plot of released_year against streams, we notice that the older songs have less streams than songs in the years after, which could tell us that the 'hype' or 'enjoyment' for the songs start to decrease the longer that they are out. Many of the other variables in this data set are actually shown to have little to no affect on predicting the number of streams a song has. This is because the taste in music of people is widely varied, so there is not one type of music that is extremely dominant. For models such as K-nearest neighbors, we decided to leave out many of these variables 1. because KNN works better with less predictors and 2. because they were not necessary for the prediction accuracy. On the other hand, for the tree based methods, we did leave all of the predictors in because when we tested them, the model worked better with more predictors. The Bagged model was the best performing model with an Test MSE of .2847. This means that this model did the best at predicting the number of streams on the test data set. If given a new data set with new songs, the best model to predict the performance of the songs would be the bagging model.