Machine Learning Semester Project

This project was conducted for the course: STAT 253 Statistical Machine Learning. We analyzed data from Spotify with the goal of predicting song popularity!

AUTHOR

Nicholas Di and Amy Xu

PUBLISHED

May 1, 2020

Data Context

We selected the dataset “Top Spotify songs from 2010-2019 - BY YEAR” from Kaggle, which consists of the top songs by year in the world on Spotify, and the data is based on Billboard. The dataset contains 13 variables to be explored, including information about the songs such as the song’s title, the song’s artist, the genre of the track, the year in which the song was in the Billboard rankings, its duration and acousticness. There are also variables describing the music, such as beats per minute (which characterizes the tempo of the song), energy (the higher the value, the more energetic the song is), danceability (the higher the value, the easier it is to dance to this song), loudness (measured in dB), liveness (the higher the value, the more likely the song is a live recording), valence (the higher the value, the more positive the mood is for the song), and speechiness (the higher the value, the more spoken words the song contains). The outcome variable in this dataset is popularity, where the higher the value, the more popular the song is.

The data were extracted from: http://organizeyourmusic.playlistmachinery.com/, and the dataset was constructed by Leonardo Henrique, updated in 2020. He was interested in what we could know about the specific music genre based on the popularity of the songs, and what elements would contribute to this popularity.

Research questions

Regression: How can we predict the energy level of a song based on all other predictors?

Classification: How can we predict whether the song is amongst the most popular based on all other predictors?

```
# Load data and required packages
library (       caret    )
library (       ggplot2  )
library (       dplyr    )
library (       readr    )
library (       ISLR     )
library (       splines  )
library (       caret    )
library (       stats    )
library (       lattice  )
library (       leaps    )
library (       gam      )
top_spotify <-       read_csv(
'https://www.dropbox.com/s/fi22whryueo4q85/top10s.csv?dl=1')
```

```
# Any code to clean the data
top_spotify_new <-       top_spotify %>%      select  (       -       artist   ,-
'top genre',-       title    ,-       ...1    )
#There seems to be an outlier
top_spotify_new <-       filter  (       top_spotify_new, bpm      >       1
)
```

Initial investigation 1: ignoring nonlinearity (for now)

We ordinary least squares (OLS) regression, forward and/or backward selection, and LASSO to build initial models for our quantitative outcome as a function of the predictors of interest.

OLS Model

Fit the ordinary least squares (OLS) regression model:

```
OLS      <-       lm      (       nrgy     ~        year     +        acous
+        bpm      +        pop      +        dnce     +        live     +
spch     +        dur      , data =        top_spotify_new)
summary (       OLS     )
```

```
Call:
lm(formula = nrgy ~ year + acous + bpm + pop + dnce + live +
    spch + dur, data = top_spotify_new)

Residuals:
    Min      1Q  Median      3Q     Max
```

```
-43.223  -8.030   1.092   8.540  31.239

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.347e+03  4.228e+02   5.552 4.26e-08 ***
year        -1.119e+00  2.097e-01  -5.337 1.34e-07 ***
acous       -4.123e-01  2.611e-02 -15.790  < 2e-16 ***
bpm         -3.427e-03  2.178e-02  -0.157  0.87504
pop         -4.448e-02  3.713e-02  -1.198  0.23131
dnce         1.833e-03  4.190e-02   0.044  0.96511
live         1.239e-01  4.012e-02   3.087  0.00212 **
spch         2.046e-01  6.916e-02   2.958  0.00322 **
dur         -7.281e-02  1.568e-02  -4.644 4.20e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 12.51 on 593 degrees of freedom
Multiple R-squared:  0.402, Adjusted R-squared:  0.3939
F-statistic: 49.83 on 8 and 593 DF,  p-value: < 2.2e-16
```

```
set.seed(       253     )
OLS_cv   <-       train   (
    nrgy     ~        year     +        acous    +        bpm      +        pop
+        dnce     +        live     +        spch     +        dur      ,
    data =        top_spotify_new,
    method =        "lm"     ,
    trControl =        trainControl(       method =        "cv"     , number =
9       )       ,
    na.action =        na.omit )
```

Backward Stepwise Selection Model

Fit the Backward Stepwise Selection model:

```
full_model <-       lm      (       nrgy     ~        .        , data =
top_spotify_new)
summary (       full_model)
```

```
Call:
lm(formula = nrgy ~ ., data = top_spotify_new)

Residuals:
     Min       1Q   Median       3Q      Max
-28.7877  -6.2817   0.6035   6.8994  27.0338

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 812.704006 344.864487   2.357 0.018769 *
year         -0.350636   0.171100  -2.049 0.040874 *
bpm          -0.004972   0.017149  -0.290 0.771957
```

```
dnce         -0.106738   0.037418  -2.853 0.004488 **
dB            4.479083   0.268874  16.659  < 2e-16 ***
live          0.103313   0.031586   3.271 0.001135 **
val           0.113503   0.022958   4.944 9.99e-07 ***
dur          -0.017890   0.012827  -1.395 0.163629
acous        -0.289294   0.021547 -13.426  < 2e-16 ***
spch          0.206303   0.055593   3.711 0.000226 ***
pop          -0.074613   0.029247  -2.551 0.010988 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.84 on 591 degrees of freedom
Multiple R-squared:  0.6312,    Adjusted R-squared:  0.625
F-statistic: 101.2 on 10 and 591 DF,  p-value: < 2.2e-16
```

```
set.seed(       253     )
back_step_mod <-       train   (
    nrgy     ~        .       ,
    data =        top_spotify_new,
    method =        'leapBackward',
    tuneGrid =        data.frame(       nvmax =       1       :       10      )
,
    trControl =        trainControl(       method =       'cv'     ,number =
9       )       ,
    metric =        'MAE'    ,
    na.action =        na.omit
)
```

Forward Stepwise Selection Model

Fit the Forward Stepwise Selection model:

```
set.seed(       253     )
for_step_mod <-       train   (
    nrgy     ~        .       ,
    data =        top_spotify_new,
    method =        'leapForward',
    tuneGrid =        data.frame(       nvmax =       1       :       10      )
,
    trControl =        trainControl(       method =       'cv'     ,number =
9       )       ,
    metric =        'MAE'    ,
    na.action =        na.omit
)
```

LASSO Model

Fit the LASSO model:

```
set.seed(       253     )
lasso_mod <-       train   (
    nrgy     ~        .       ,
    data =        top_spotify_new,
    method =         "glmnet",
    tuneGrid =         data.frame(       alpha =        1       , lambda =
seq     (       0       , 10      , length.out =        100     )       )       ,
    trControl =        trainControl(       method =        "cv"     , number =
9       , selectionFunction =        'oneSE' )       ,
    metric =        "MAE"    ,
    na.action =        na.omit )
```

Compare performances of different models:

Estimate test performance of the models from these different methods. Report and interpret (with units) these estimates along with a measure of uncertainty in the estimate (SD is most readily available from `caret`).

Examine OLS model outputs:

```
summary (       OLS_cv   )
```

```
Call:
lm(formula = .outcome ~ ., data = dat)

Residuals:
    Min      1Q  Median      3Q     Max
-43.223  -8.030   1.092   8.540  31.239

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.347e+03  4.228e+02   5.552 4.26e-08 ***
year        -1.119e+00  2.097e-01  -5.337 1.34e-07 ***
acous       -4.123e-01  2.611e-02 -15.790  < 2e-16 ***
bpm         -3.427e-03  2.178e-02  -0.157  0.87504
pop         -4.448e-02  3.713e-02  -1.198  0.23131
dnce         1.833e-03  4.190e-02   0.044  0.96511
live         1.239e-01  4.012e-02   3.087  0.00212 **
spch         2.046e-01  6.916e-02   2.958  0.00322 **
dur         -7.281e-02  1.568e-02  -4.644 4.20e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 12.51 on 593 degrees of freedom
Multiple R-squared:  0.402, Adjusted R-squared:  0.3939
F-statistic: 49.83 on 8 and 593 DF,  p-value: < 2.2e-16
```

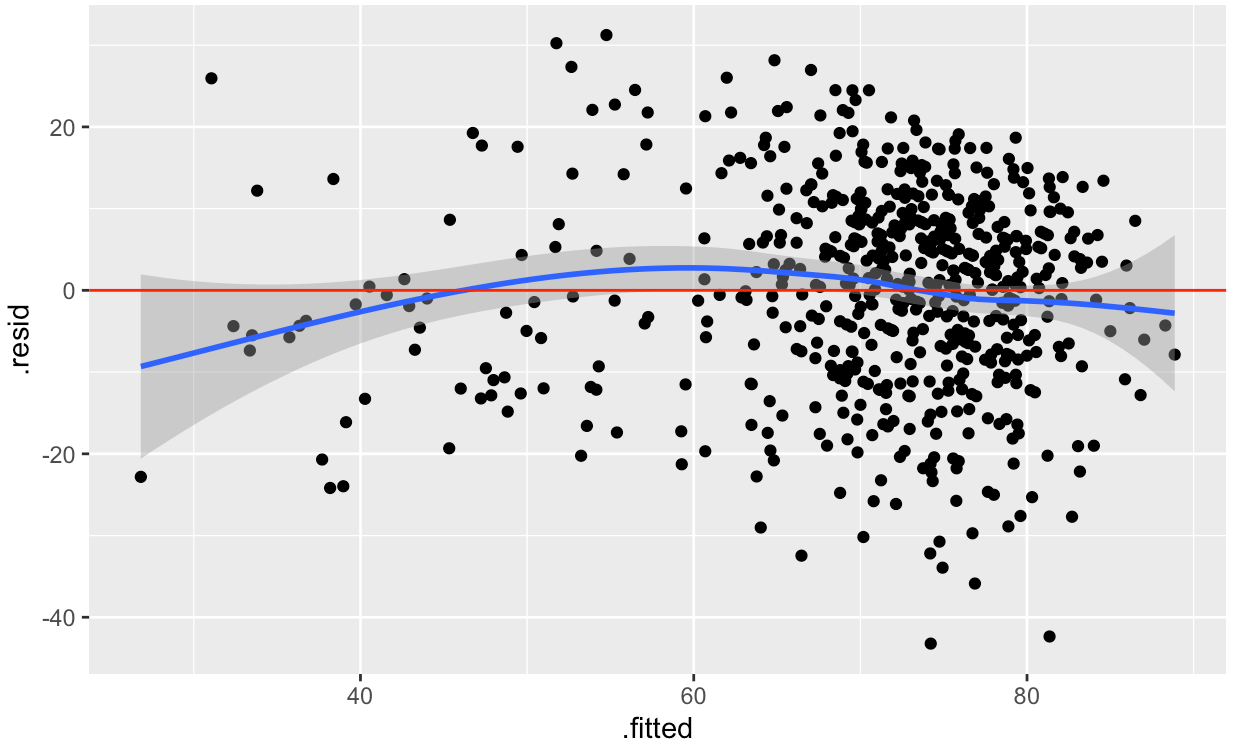
```
OLS_cv  $       results
```

```
  intercept     RMSE  Rsquared      MAE   RMSESD RsquaredSD   MAESD
1      TRUE 12.62445 0.3880631 10.13206 1.140868  0.1381483 0.83748
```

On average, we're off in top song energy predictions by about 10.13206 points.

Residual plot for OLS model:

```
OLS_mod_output <-       broom    ::      augment (       OLS      , newdata =
top_spotify_new)

ggplot  (       OLS_mod_output, aes     (       x =         .fitted  , y =
.resid   )       )        +
    geom_point(        )        +
    geom_smooth(        )        +
    geom_hline(       yintercept =        0        , color =        "red"    )
```



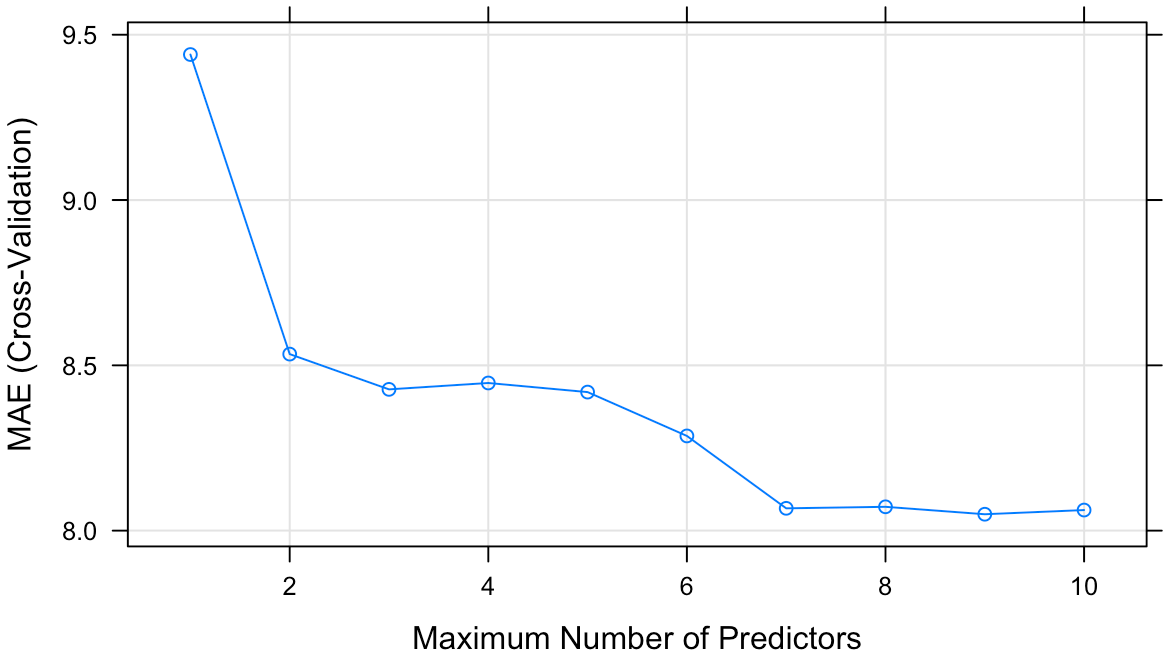
Examine Backward Stepwise Selection model output:

```
summary (       back_step_mod)
```

```
Subset selection object
10 Variables  (and intercept)
      Forced in Forced out
year      FALSE      FALSE
bpm       FALSE      FALSE
dnce      FALSE      FALSE
dB        FALSE      FALSE
live      FALSE      FALSE
val       FALSE      FALSE
dur       FALSE      FALSE
```

```
acous      FALSE      FALSE
spch       FALSE      FALSE
pop        FALSE      FALSE
1 subsets of each size up to 9
Selection Algorithm: backward
         year bpm dnce dB  live val dur acous spch pop
1  ( 1 ) " "  " " " " "  "*" " "  " " " " " " " "   " "  " "
2  ( 1 ) " "  " " " " "  "*" " "  " " " " " " "*"   " "  " "
3  ( 1 ) " "  " " " " "  "*" " "  " " " " " " "*"   "*"  " "
4  ( 1 ) " "  " " " " "  "*" " "  "*" " " " " "*"   "*"  " "
5  ( 1 ) " "  " " " " "  "*" "*"  "*" " " " " "*"   "*"  " "
6  ( 1 ) " "  " " " " "*"  "*" "*"  "*" " " " " "*"   "*"  " "
7  ( 1 ) " "  " " " " "*"  "*" "*"  "*" " " " " "*"   "*"  "*"
8  ( 1 ) "*"  " " " " "*"  "*" "*"  "*" " " " " "*"   "*"  "*"
9  ( 1 ) "*"  " " " " "*"  "*" "*"  "*" "*" "*" "*"   "*"  "*"
```

```
plot    (       back_step_mod)
```




```
back_step_mod$       bestTune
```

```
  nvmax
9     9
```

We chose the model with 6 predictors. Although all 10 yields better model metrics, we believe that we will run into problems of over-fitting.

On average, we're off in top song energy predictions by about 8.286 percentage points, if we use the model with 6 predictors.

```
coef    (       back_step_mod$       finalModel, id =       6       )
```

```
(Intercept)        dnce          dB        live         val
 97.4672886  -0.1228588   4.5587752   0.1128596   0.1264183
      acous        spch
 -0.2930557   0.2006552
```

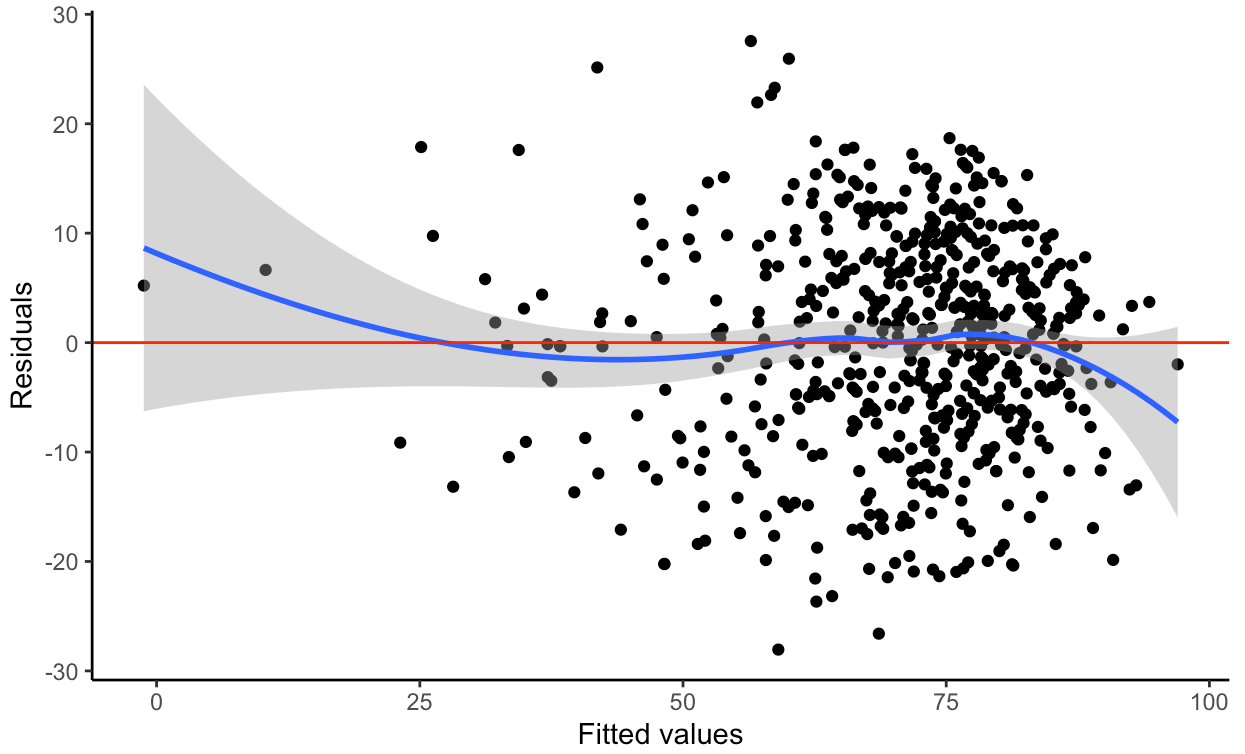
```
back_step_mod$       results  %>%      filter  (       nvmax    ==      6       )
```

```
  nvmax     RMSE  Rsquared      MAE    RMSESD RsquaredSD    MAESD
1     6 10.18581 0.5910122 8.286481 0.9244708  0.1070956 0.825095
```

```
back_step_mod_eq <-       lm      (       nrgy     ~        dnce     +        dB
+        live     +        val      +        acous    +        spch     , data =
top_spotify_new)
```

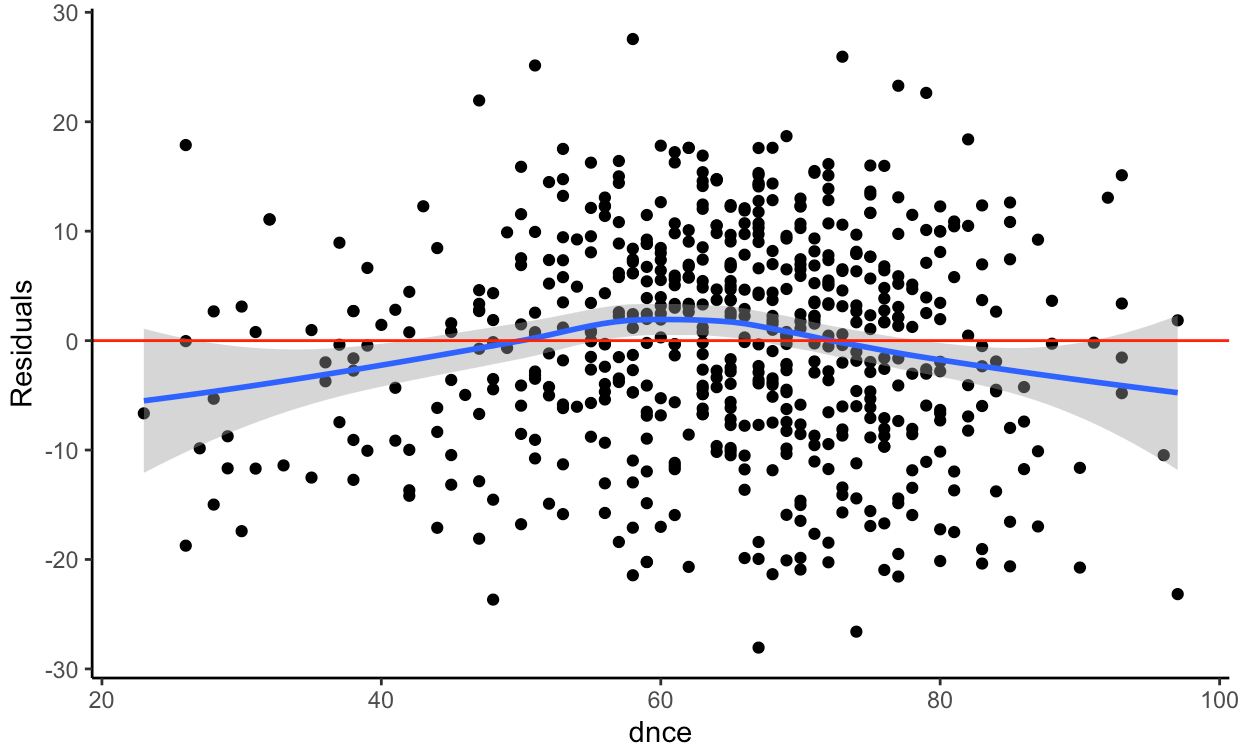
Residual plot for Backward Step-wise Selection model:

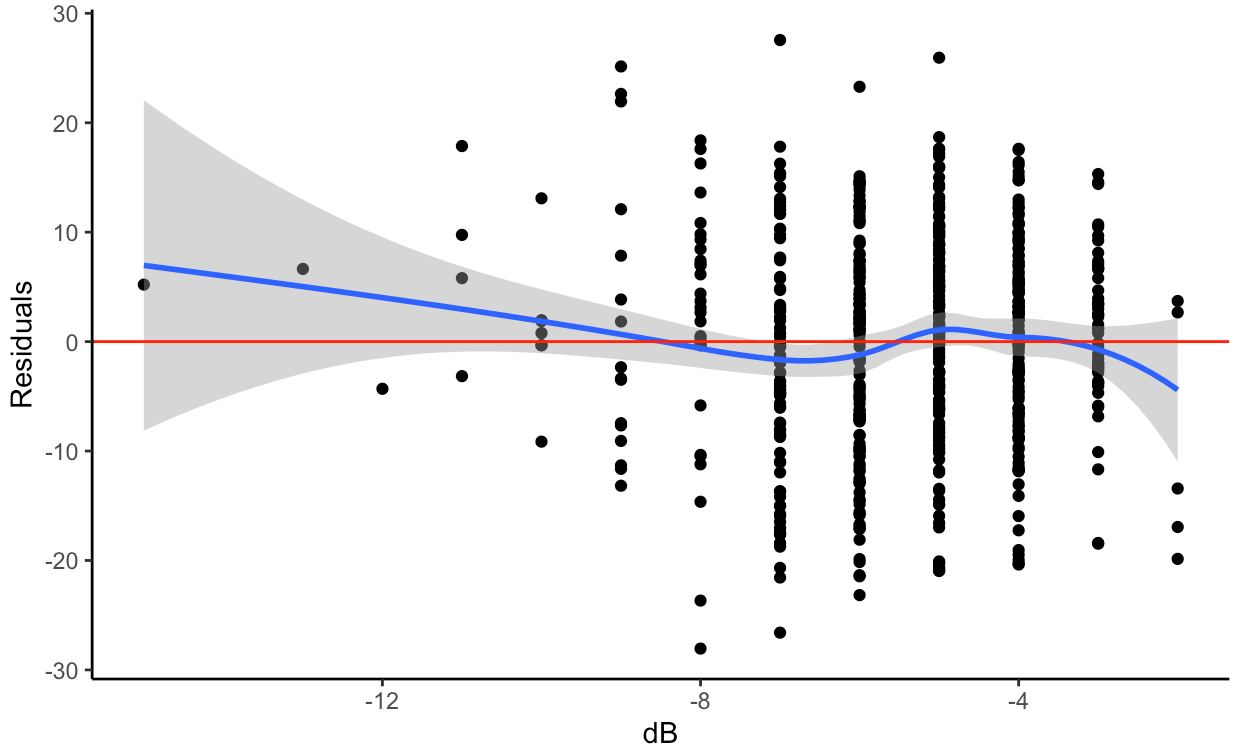
```
back_step_mod_out <-       top_spotify_new %>%
    mutate  (
        fitted =        predict (       back_step_mod_eq, newdata =
top_spotify_new)       ,
        resid =        nrgy     -        fitted
    )

ggplot  (       back_step_mod_out, aes     (       x =        fitted   , y =
resid   )       )        +
    geom_point(       )        +
    geom_smooth(       )        +
    geom_hline(       yintercept =        0       , color =        "red"    )
+
    theme_classic(       )        +
    labs    (       x =        "Fitted values", y =        "Residuals")
```

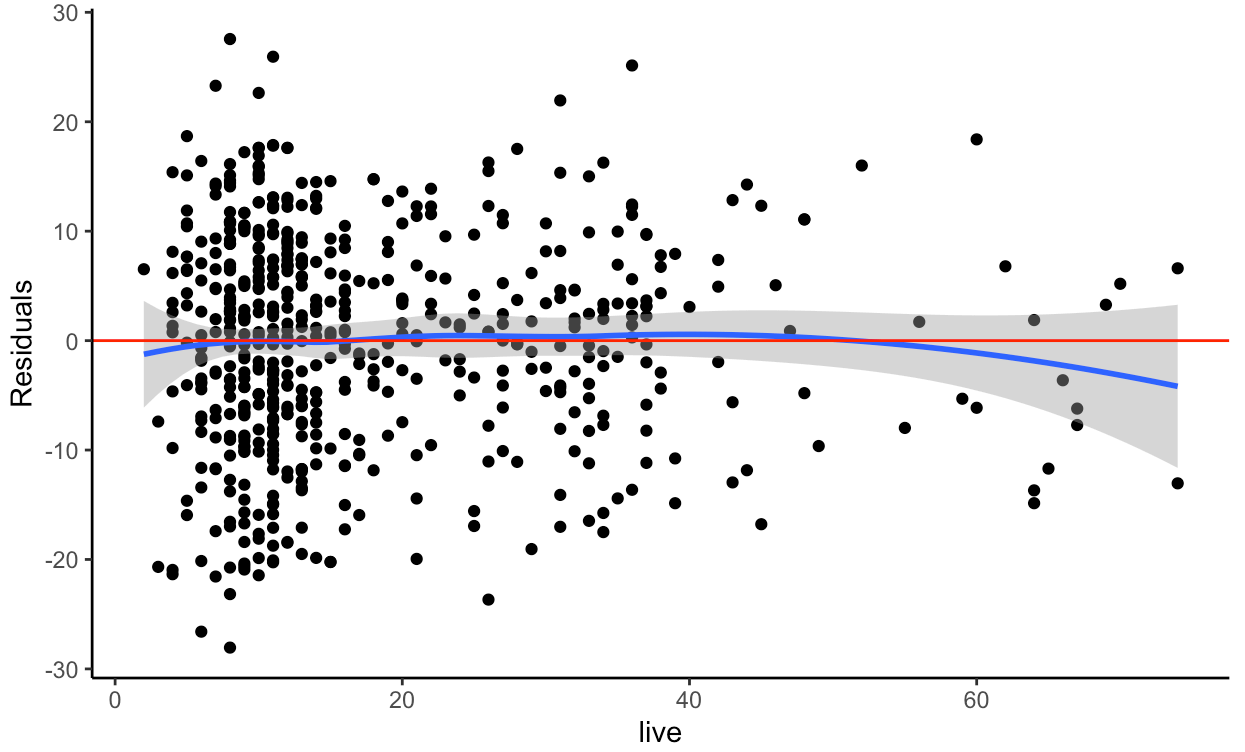


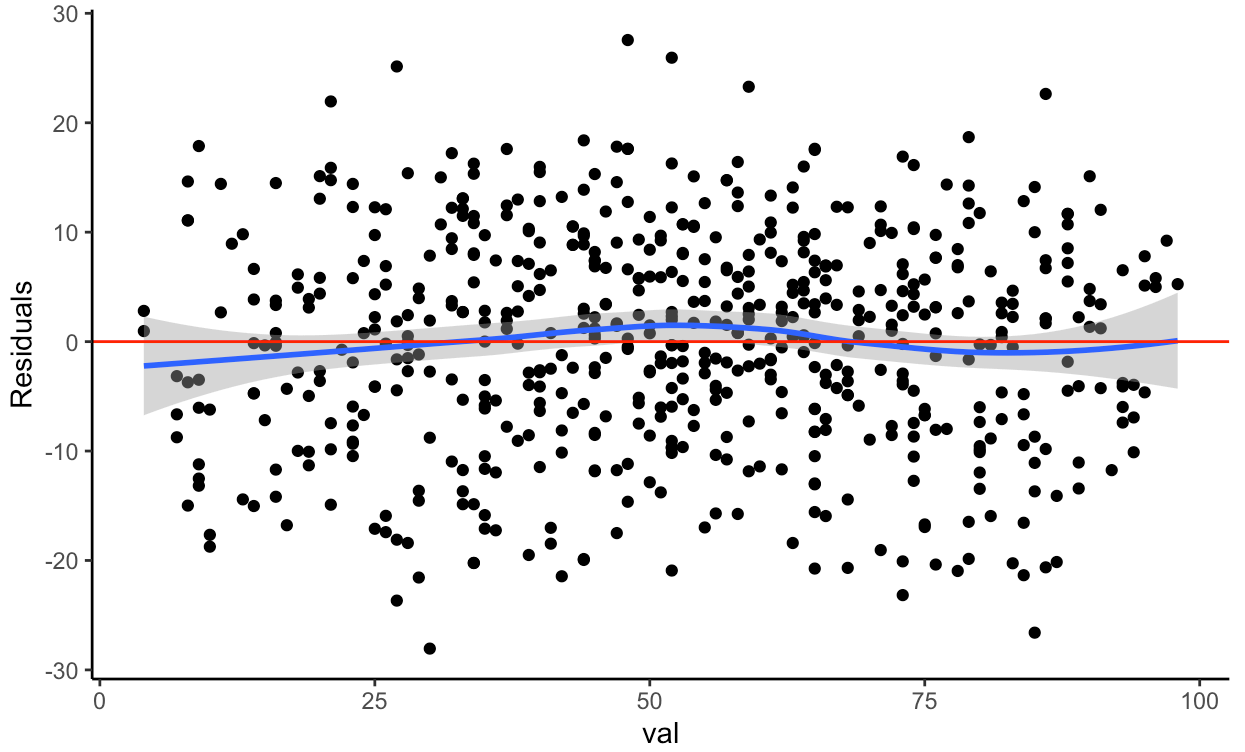
For Loop looking at all predictors in the Backward Stepwise Selection model:

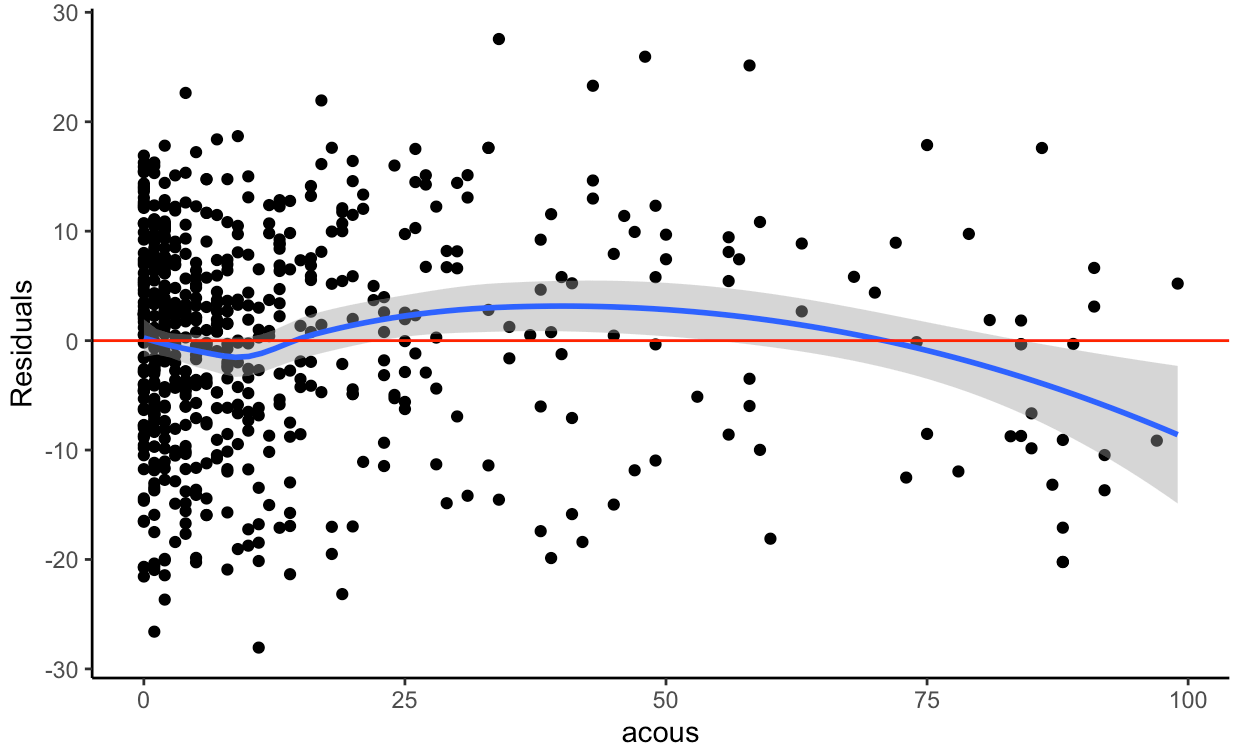
```
predictors <-       setdiff (       colnames(       top_spotify_new)       , c
(       "year"   ,"bpm"    ,"nrgy"   ,"dur"    ,"spch"   )       )
for      (       pred     in       predictors)       {
    p        <-       ggplot   (       back_step_mod_out, aes     (       x =
.data    [[      pred    ]       ]       , y =        resid    )       )       +
        geom_point(       )       +
        geom_smooth(       )       +
        geom_hline(       yintercept =       0       , color =       "red"    )
+
        theme_classic(       )       +
        labs    (       x =       pred    , y =       "Residuals")
    print   (       p       )
}
```

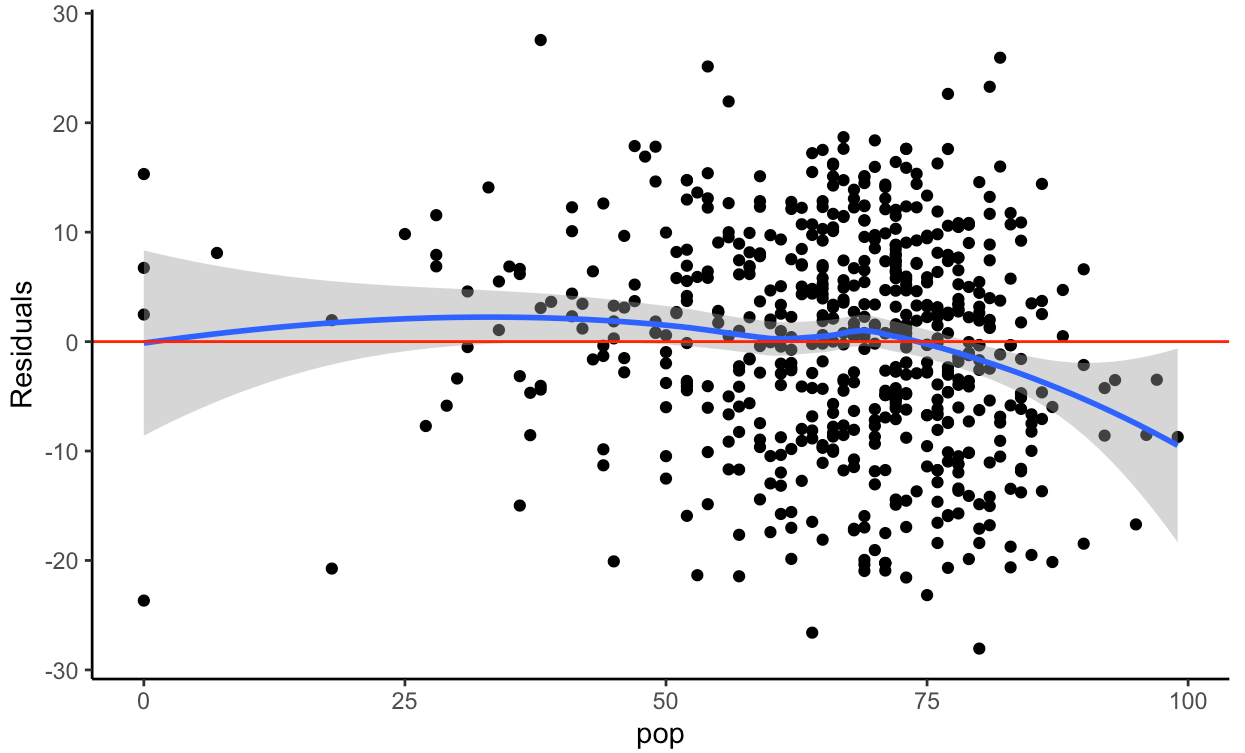

Residuals
30
20
10
0
-10
-20
-30
20
40
60
80
100
dnce


Residuals
30
20
10
0
-10
-20
-30
-12
-8
-4
dB


Residuals
30
20
10
0
-10
-20
-30
0
20
40
60
live


Residuals
30
20
10
0
-10
-20
-30
0
25
50
75
100
val





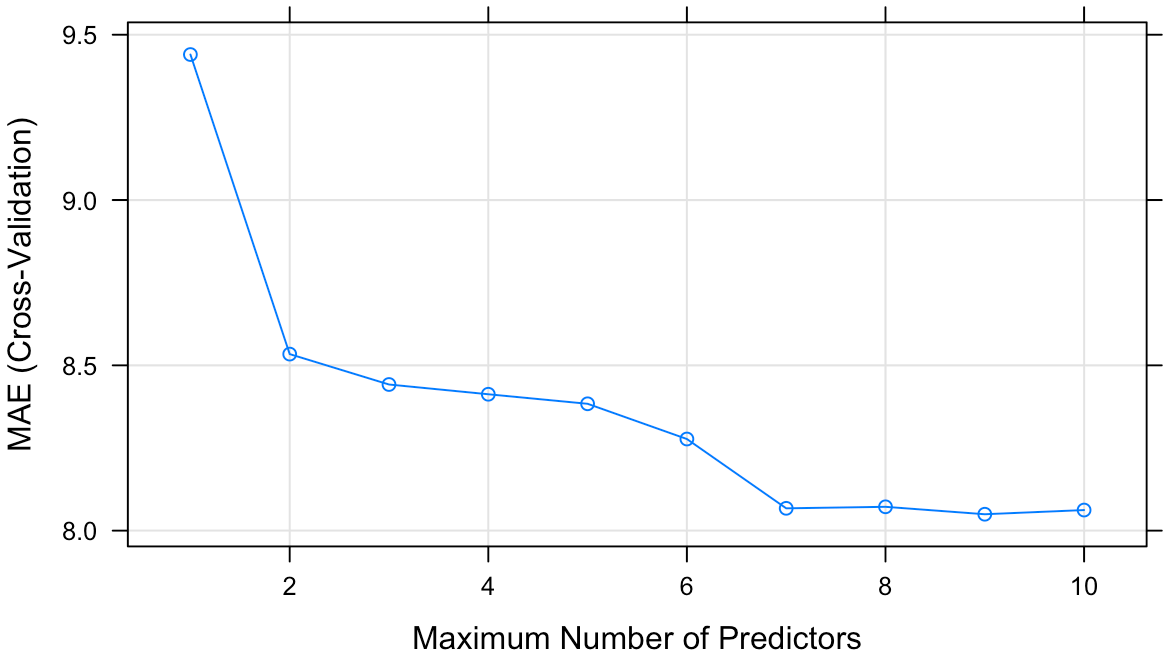
Examine Forward Stepwise Selection model output:

```
summary (       for_step_mod)
```

```
Subset selection object
10 Variables  (and intercept)
      Forced in Forced out
year      FALSE      FALSE
bpm       FALSE      FALSE
```

```
dnce      FALSE      FALSE
dB        FALSE      FALSE
live      FALSE      FALSE
val       FALSE      FALSE
dur       FALSE      FALSE
acous     FALSE      FALSE
spch      FALSE      FALSE
pop       FALSE      FALSE
1 subsets of each size up to 9
Selection Algorithm: forward
         year bpm dnce dB  live val dur acous spch pop
1  ( 1 ) " "  " " " " "*" " "  " " " " " " " "  " "
2  ( 1 ) " "  " " " " "*" " "  " " " " " " "*"  " "
3  ( 1 ) " "  " " " " "*" " "  " " " " " " "*"  "*"
4  ( 1 ) " "  " " " " "*" " "  "*" " " " " "*"  "*"
5  ( 1 ) " "  " " " " "*" "*"  "*" " " " " "*"  "*"
6  ( 1 ) " "  " " "*" "*" "*"  "*" " " " " "*"  "*"
7  ( 1 ) " "  " " "*" "*" "*"  "*" " " " " "*"  "*"
8  ( 1 ) "*"  " " "*" "*" "*"  "*" " " " " "*"  "*"
9  ( 1 ) "*"  " " "*" "*" "*"  "*" "*" "*" "*"  "*"
```

```
plot    (       for_step_mod)
```




```
for_step_mod$       results
```

```
   nvmax      RMSE  Rsquared      MAE    RMSESD RsquaredSD     MAESD
1      1 11.993192 0.4404595 9.440092 1.1663410 0.08326330 0.7602086
2      2 10.477550 0.5683678 8.533998 0.6892741 0.09392739 0.6832649
3      3 10.421451 0.5699856 8.442041 0.8481219 0.10845490 0.7918324
4      4 10.333804 0.5765861 8.412541 0.8391433 0.11095779 0.7986408
5      5 10.260625 0.5846211 8.383712 0.8495003 0.11238655 0.7669223
6      6 10.142928 0.5941810 8.277152 0.8950607 0.10739226 0.8186073
```

```
7      7  9.950075 0.6082421 8.067531 0.9987578 0.11136222 0.8886527
8      8  9.940663 0.6092391 8.072114 0.9905555 0.10712761 0.9066449
9      9  9.902507 0.6120420 8.049632 0.9827727 0.10630862 0.9021970
10    10  9.915936 0.6110377 8.062302 0.9829506 0.10702524 0.9087888
```

Using Forward selection, we chose the model with 5 predictors. The five predictors being acous, dB, live, spch, and val.

On average, we're off in top song energy predictions by about 8.383712 percentage points.

```
coef    (       for_step_mod$       finalModel, id =       5       )
```

```
(Intercept)          dB        live         val       acous
91.50592402  4.65576857  0.11831750  0.09041016 -0.28031999
       spch
 0.22151861
```

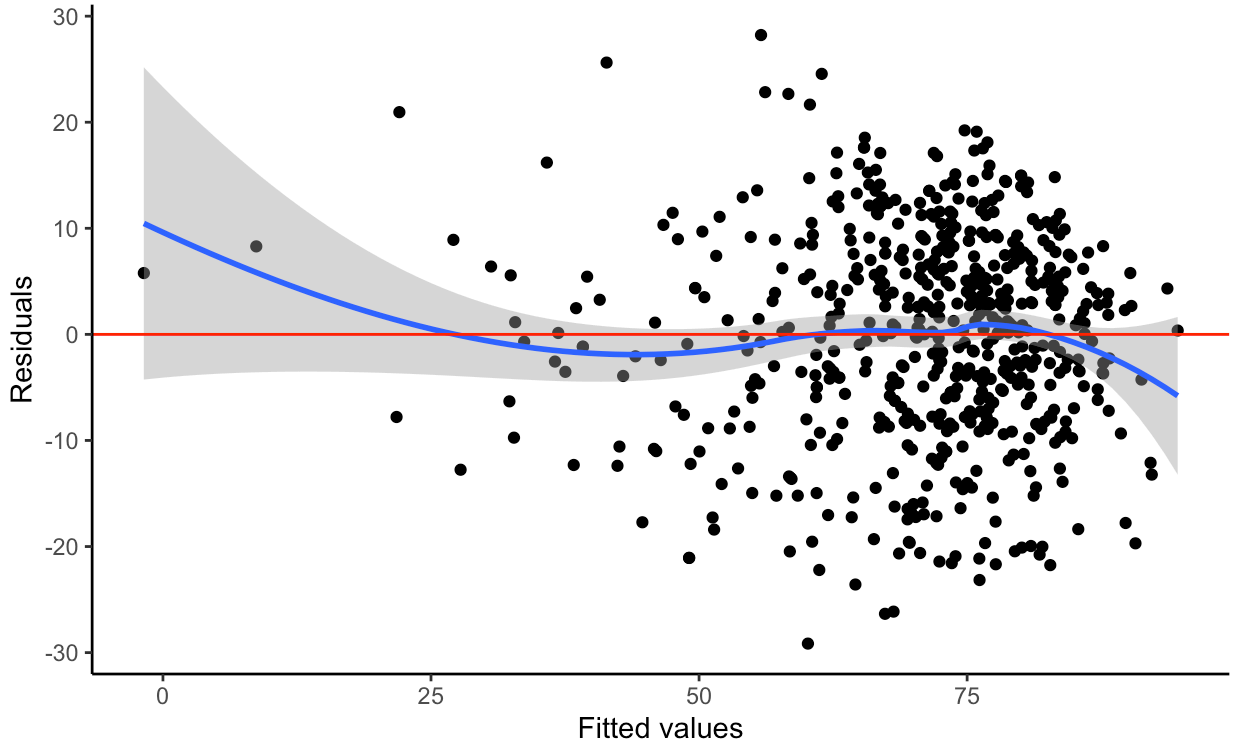
```
for_step_mod$       results  %>%      filter  (       nvmax   ==      5       )
```

```
  nvmax     RMSE  Rsquared      MAE    RMSESD RsquaredSD     MAESD
1     5 10.26063 0.5846211 8.383712 0.8495003  0.1123866 0.7669223
```

```
for_step_mod_eq <-       lm      (       nrgy     ~        acous    +        dB
+        val      +        spch     +        live    , data =       top_spotify_new
)
```

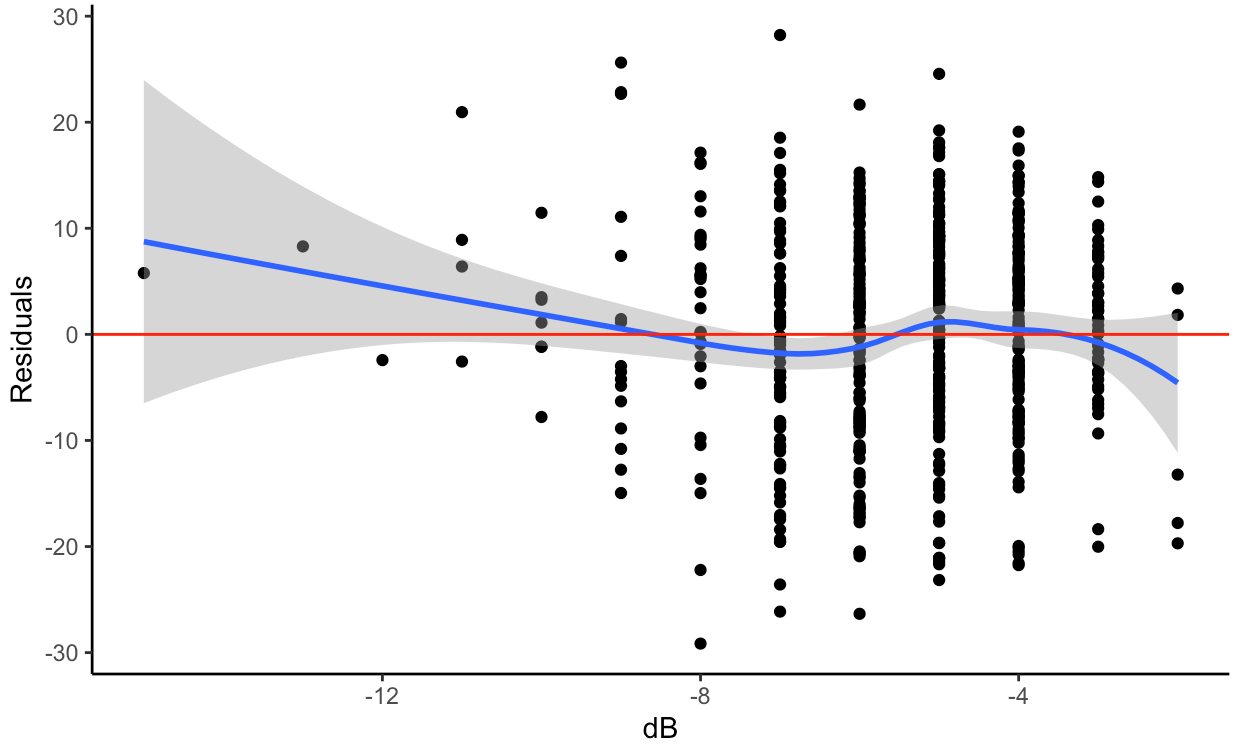
Residual plot for Forward Stepwise Selection model:

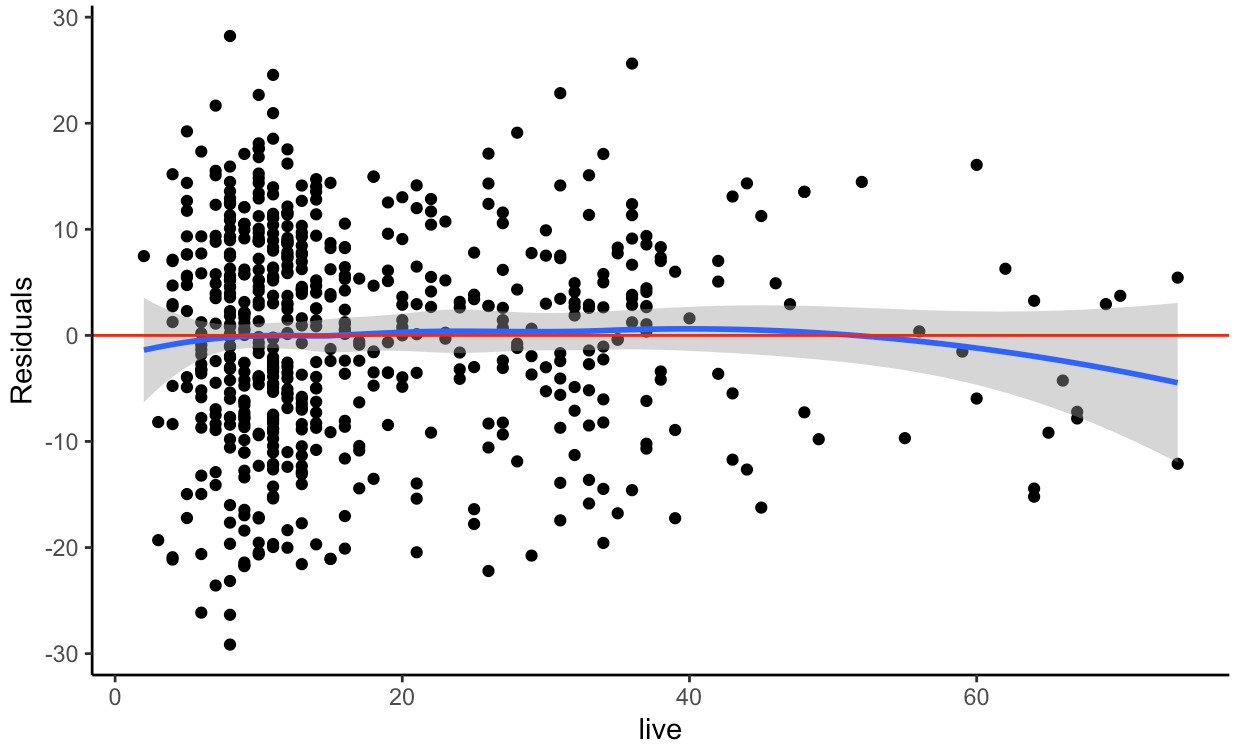
```
for_step_mod_out <-       top_spotify_new %>%
    mutate  (
        fitted =        predict (       for_step_mod_eq, newdata =
top_spotify_new)       ,
        resid =        nrgy     -        fitted
    )

ggplot  (       for_step_mod_out, aes     (       x =        fitted   , y =
resid   )       )       +
    geom_point(       )       +
    geom_smooth(       )       +
    geom_hline(       yintercept =       0       , color =       "red"    )
+
    theme_classic(       )       +
    labs    (       x =       "Fitted values", y =       "Residuals")
```

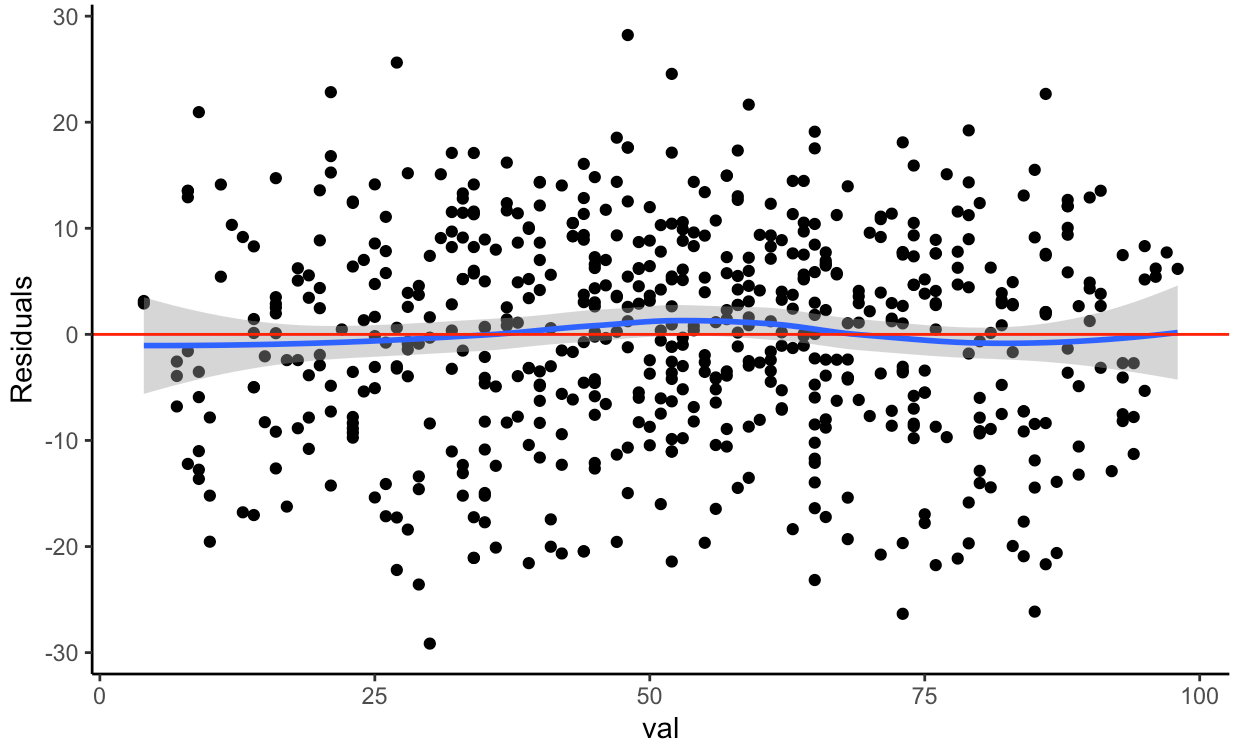


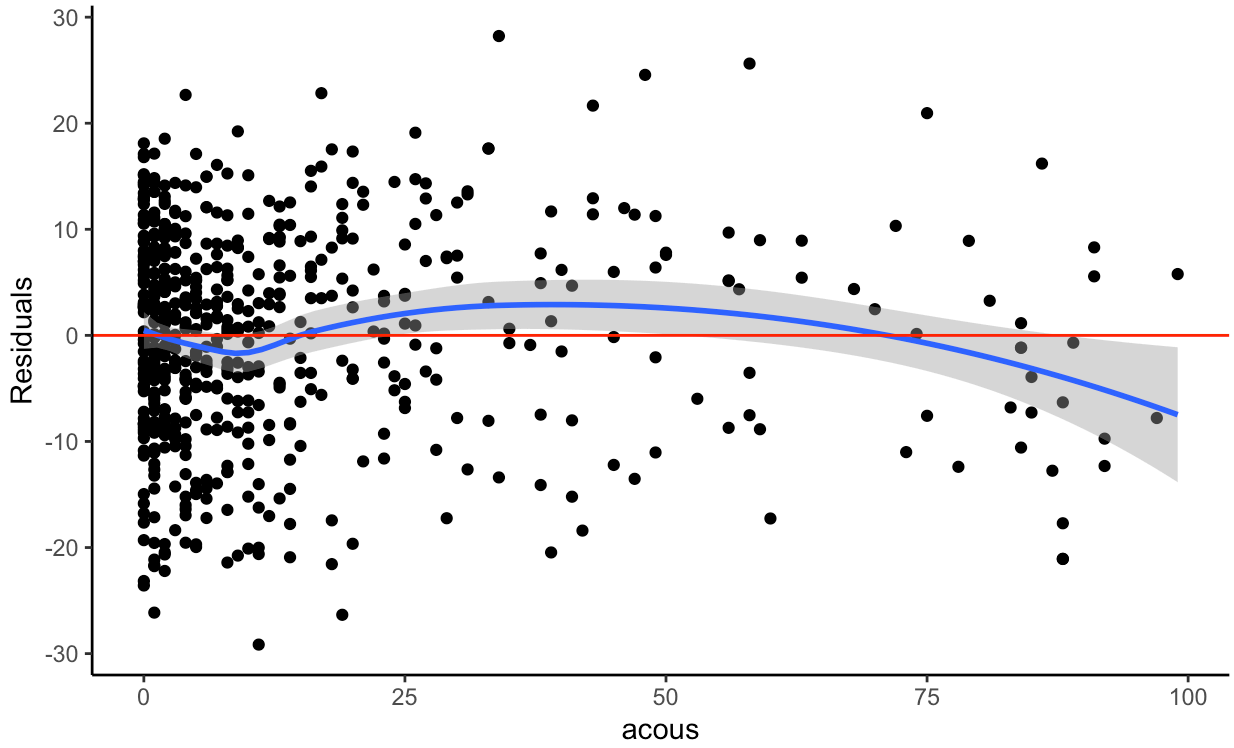
For Loop looking at all predictors in the Forward Stepwise Selection model:

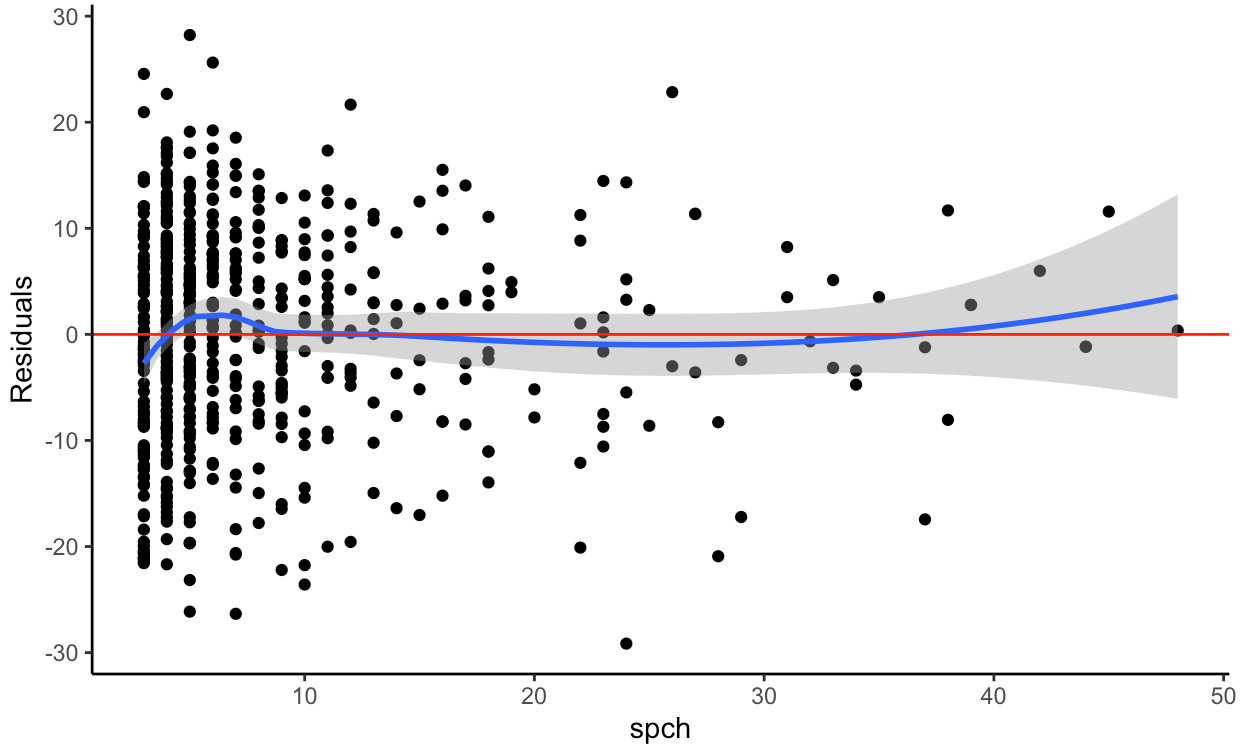
```
predictors <-       setdiff (       colnames(       top_spotify_new)       , c
(       "year"   ,"bpm"    ,"nrgy"   ,"dur"    ,"pop"    , "dnce"   )       )
for      (       pred     in       predictors)       {
    p        <-       ggplot   (       for_step_mod_out, aes     (       x =
.data    [[      pred    ]       ]       , y =        resid    )       )       +
        geom_point(       )       +
        geom_smooth(       )       +
        geom_hline(       yintercept =       0       , color =       "red"    )
+
        theme_classic(       )       +
        labs     (       x =       pred     , y =       "Residuals")
    print    (       p       )
}
```


Residuals
30
20
10
0
-10
-20
-30
-12
-8
-4
dB


Residuals
30
20
10
0
-10
-20
-30
0
20
40
60
live

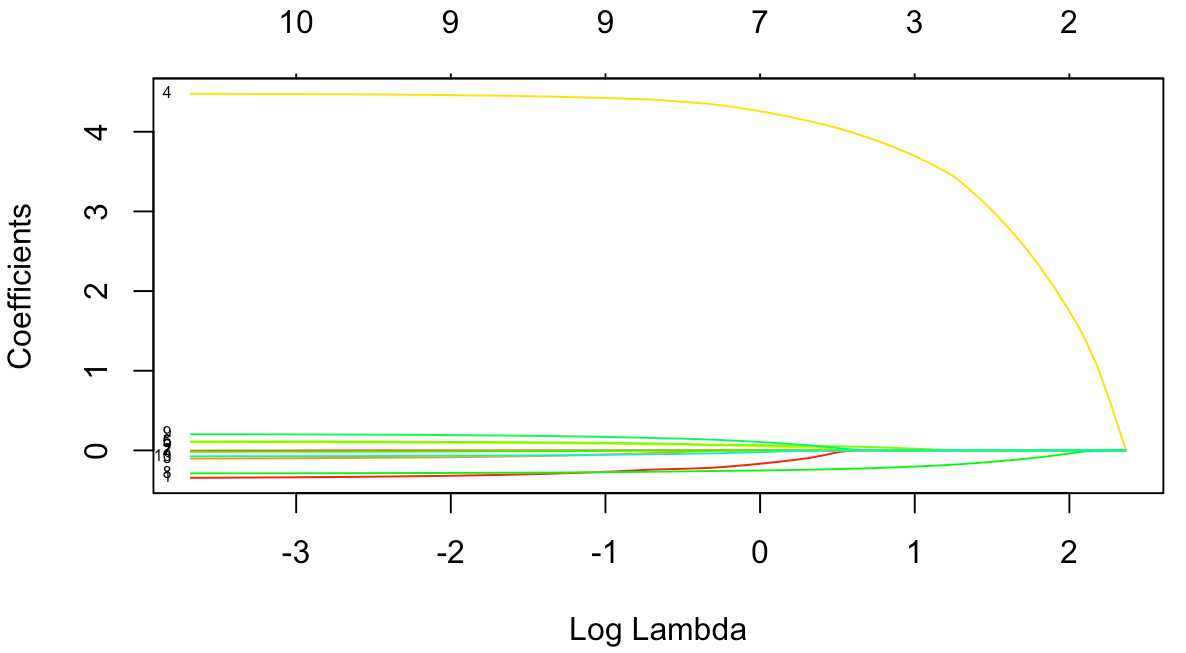

Residuals
30
20
10
0
-10
-20
-30
0
25
50
75
100
val


Residuals
30
20
10
0
-10
-20
-30
0
25
50
75
100
acous



Examine LASSO model output:

```
plot    (       lasso_mod$       finalModel, xvar =       "lambda", label =
TRUE    , col =         rainbow (       20      )       )
```




```
lasso_mod$       bestTune
```

```
   alpha   lambda
11     1 1.010101
```

```
# lasso_mod$results
rownames(       lasso_mod$       finalModel$       beta     )       [       c
(       4       ,8       )       ]
```

```
[1] "dB"    "acous"
```

We chose a lambda value of 1.010101, dB and acous seem to be two of the strongest/persistent predictors when it comes to energy-level.

```
coef    (       lasso_mod$       finalModel, 1.010101)
```

```
11 x 1 sparse Matrix of class "dgCMatrix"
                      s1
(Intercept) 427.22094252
year         -0.16551620
bpm           .
dnce          .
dB            4.25384058
live          0.05257947
val           0.06575686
dur           .
acous        -0.25154488
spch          0.10397394
pop          -0.02164891
```

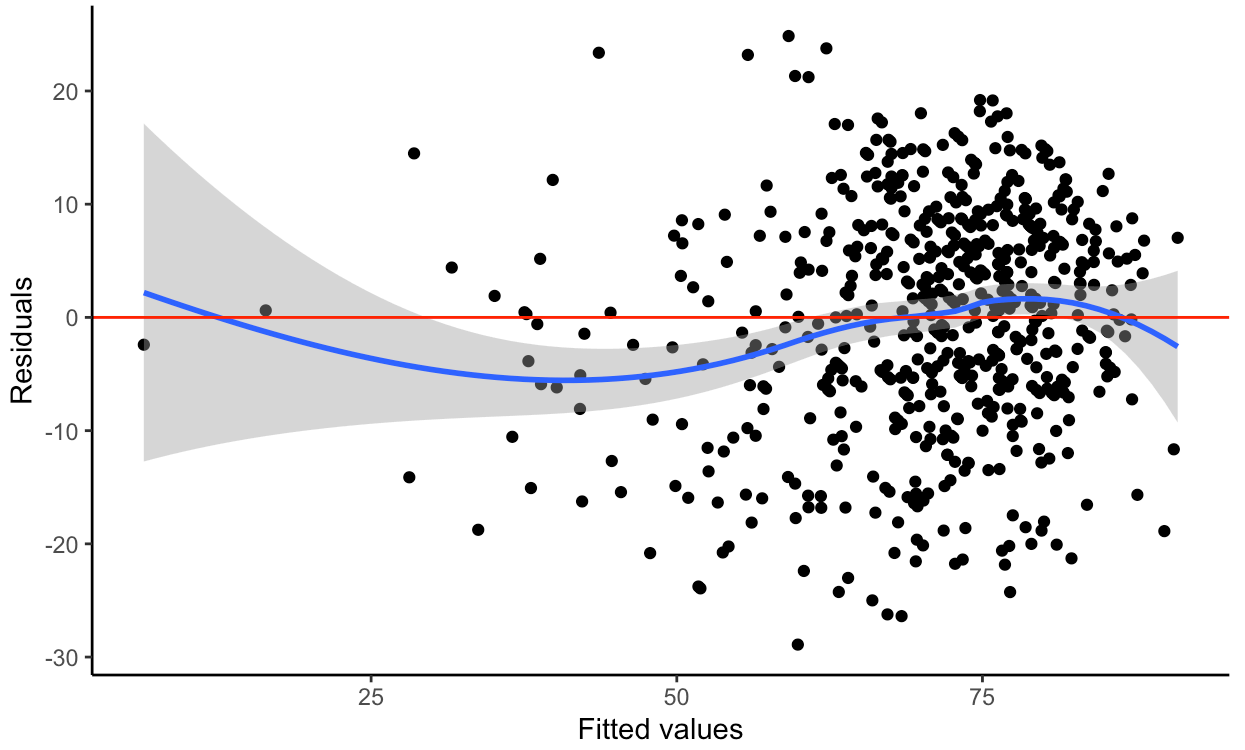
```
lasso_mod$       results [       11       ,]
```

```
   alpha   lambda     RMSE Rsquared     MAE    RMSESD RsquaredSD
11     1 1.010101 10.17634 0.597547 8.32769 0.8552498  0.1057273
       MAESD
11 0.7932597
```

On average, we're off in top song energy predictions by about 8.32769 percentage points using LASSO with a lambda of 1.010101.

Residual plot for LASSO model:

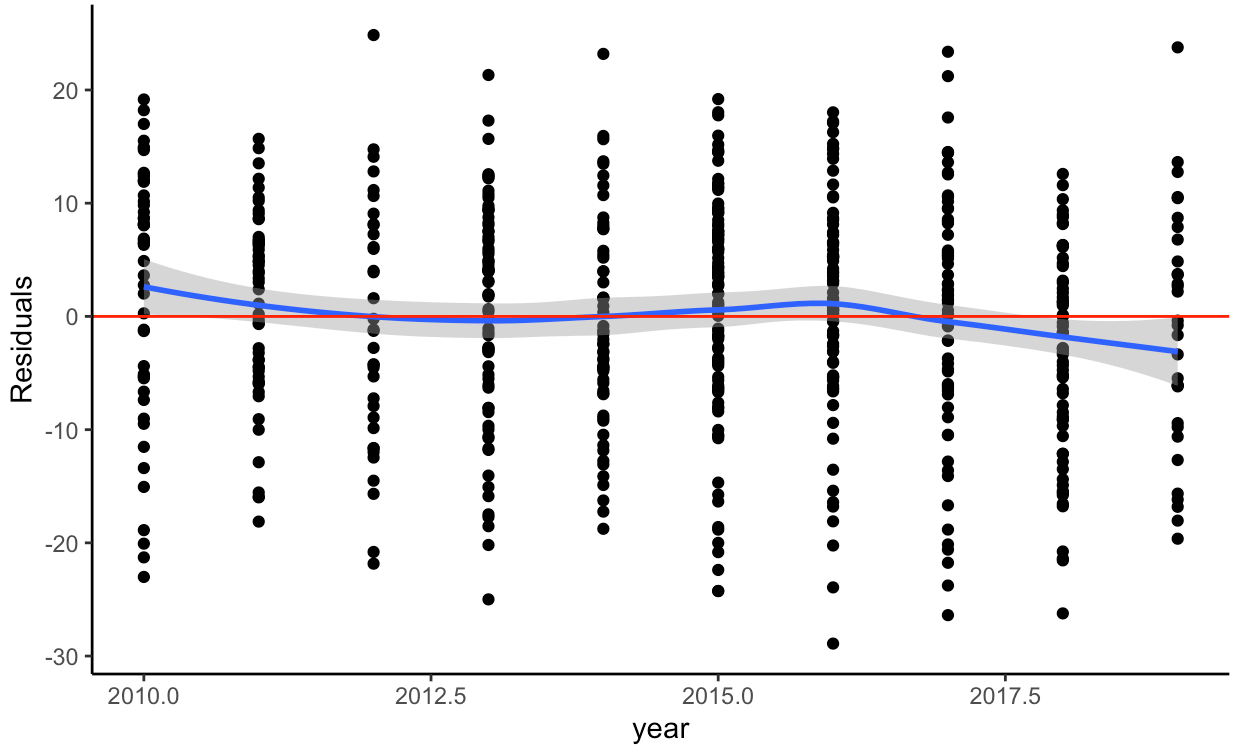
```
lasso_mod_out <-       top_spotify_new %>%
    mutate  (
        fitted =        predict (       lasso_mod, newdata =        top_spotify_new
)       ,
        resid =        nrgy     -        fitted  )
ggplot  (       lasso_mod_out, aes     (       x =        fitted  , y =        resid
)       )       +
    geom_point(       )       +
    geom_smooth(       )       +
    geom_hline(       yintercept =        0       , color =        "red"    )
+
    theme_classic(       )       +
    labs    (       x =        "Fitted values", y =        "Residuals")
```

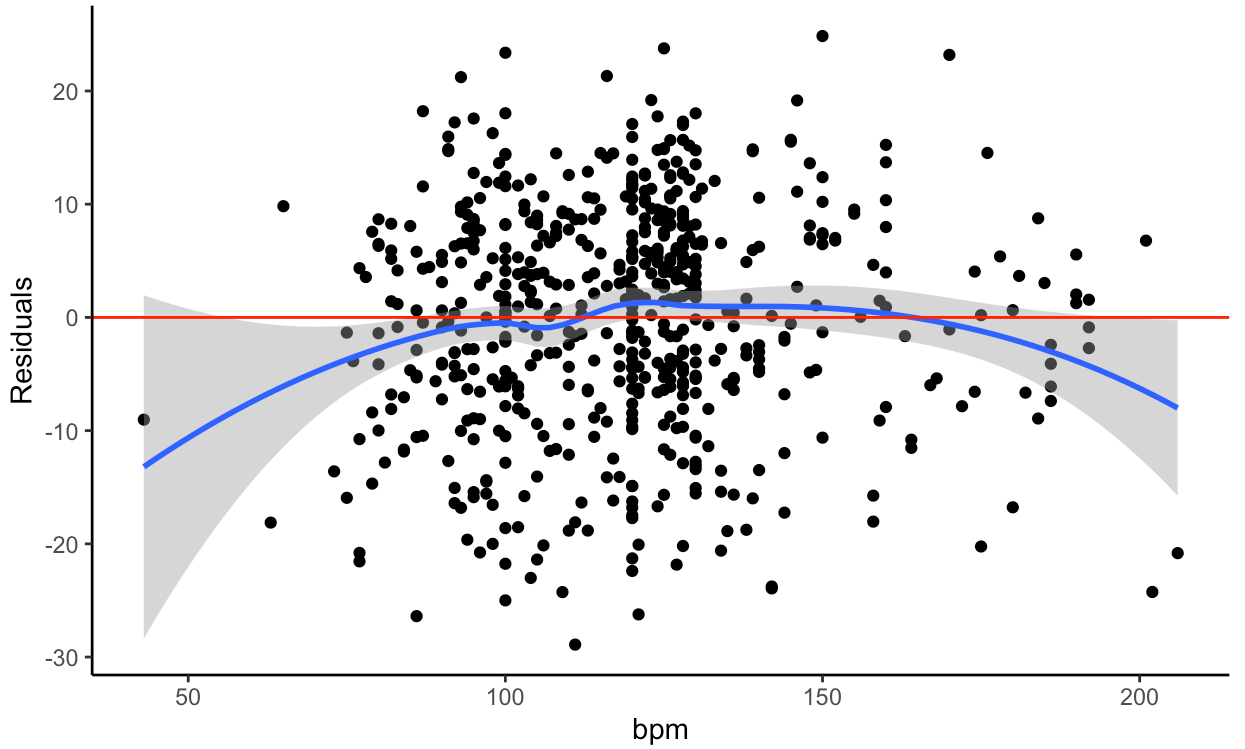


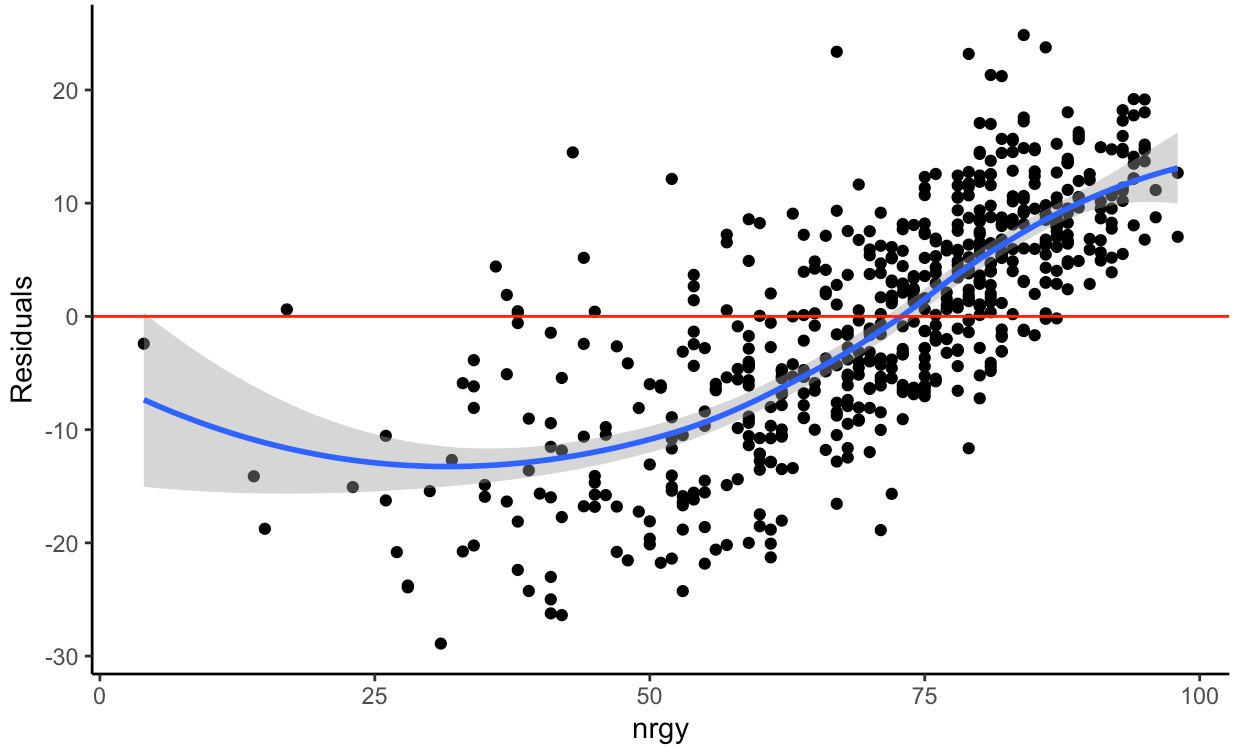
There does not seem to be any noticable patterns of over and underpredicting here!

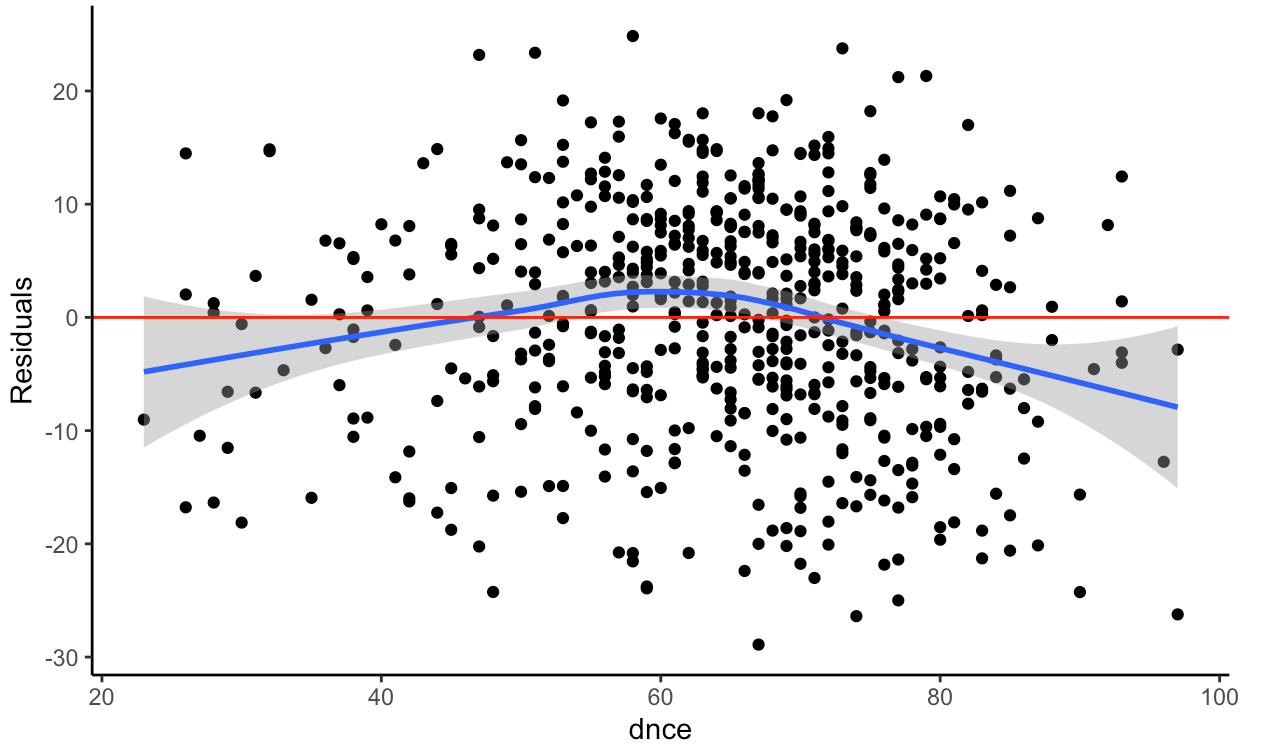
For Loop for LASSO model:

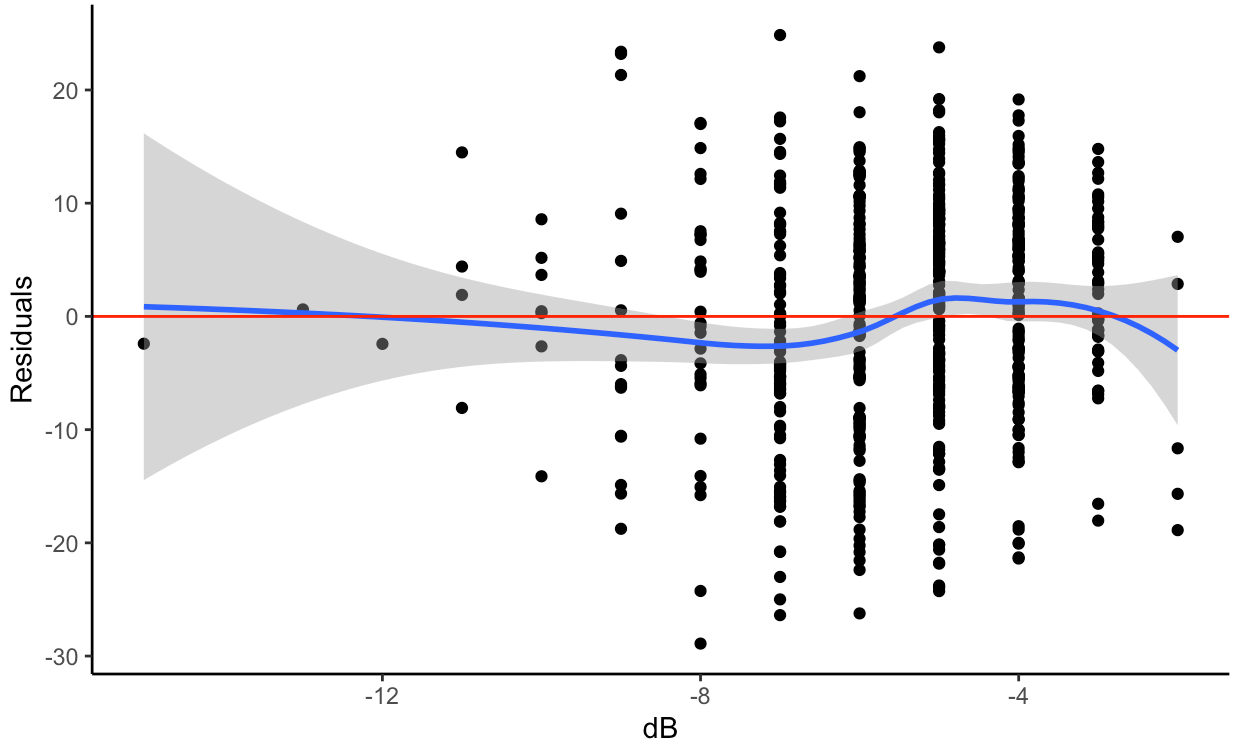
```
predictors <- setdiff(colnames(top_spotify_new), "Top Spotify")
for (pred in predictors) {
  p <- ggplot(lasso_mod_out, aes(x = .data[[pred]], y = resid)) +
    geom_point() +
    geom_smooth() +
    geom_hline(yintercept = 0, color = "red") +
    theme_classic() +
    labs(x = pred, y = "Residuals")
  print(p)
}
```

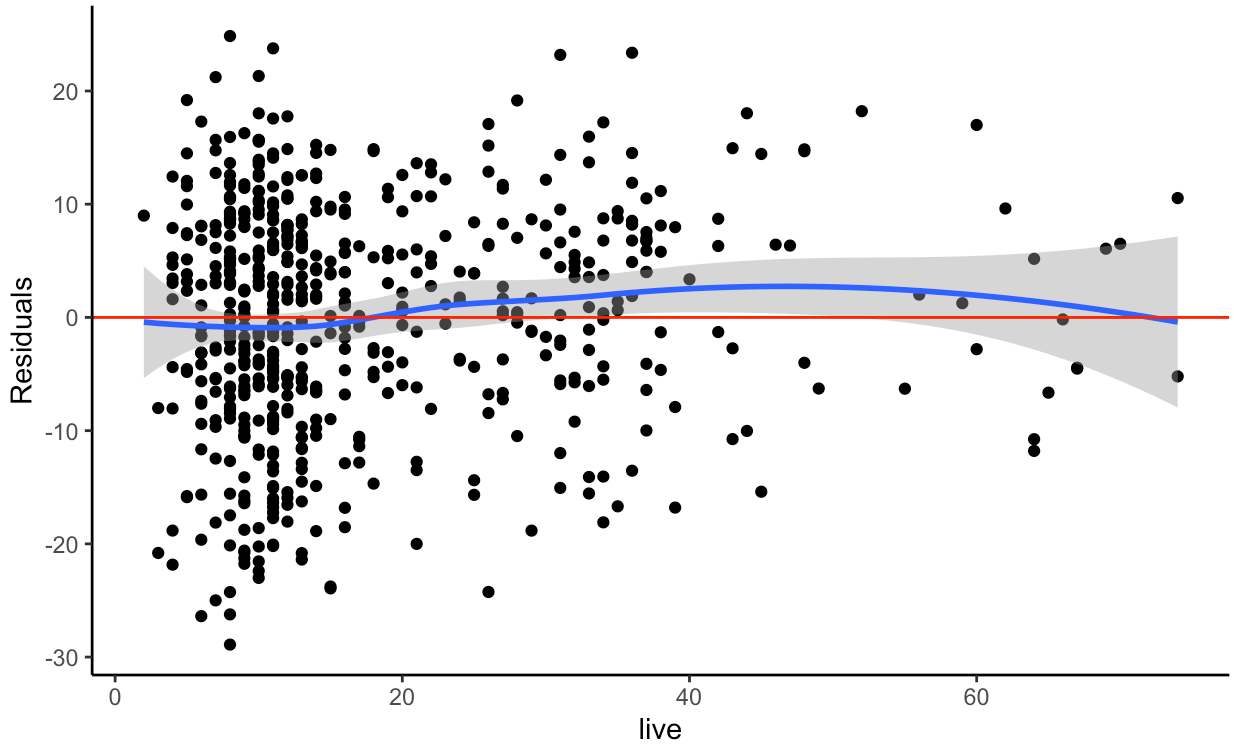

Residuals
20
10
0
-10
-20
-30
2010.0
2012.5
2015.0
2017.5
year

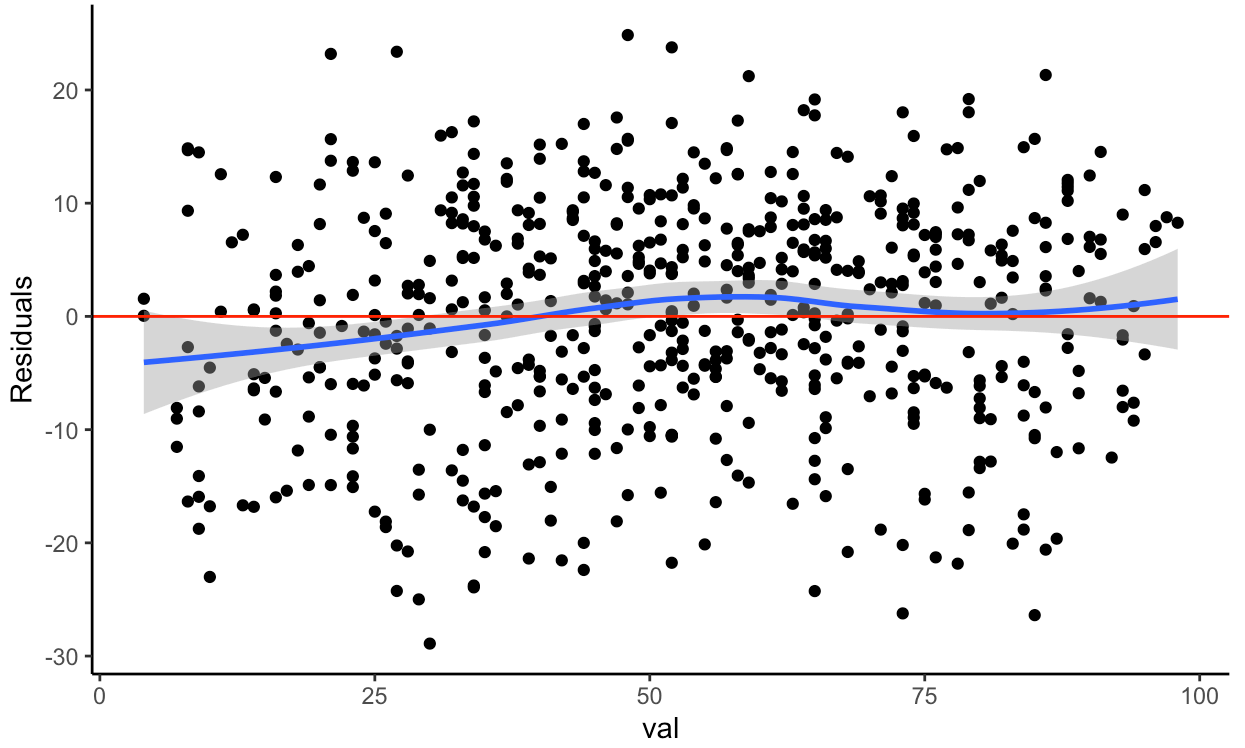

Residuals
20
10
0
-10
-20
-30
50
100
150
200
bpm

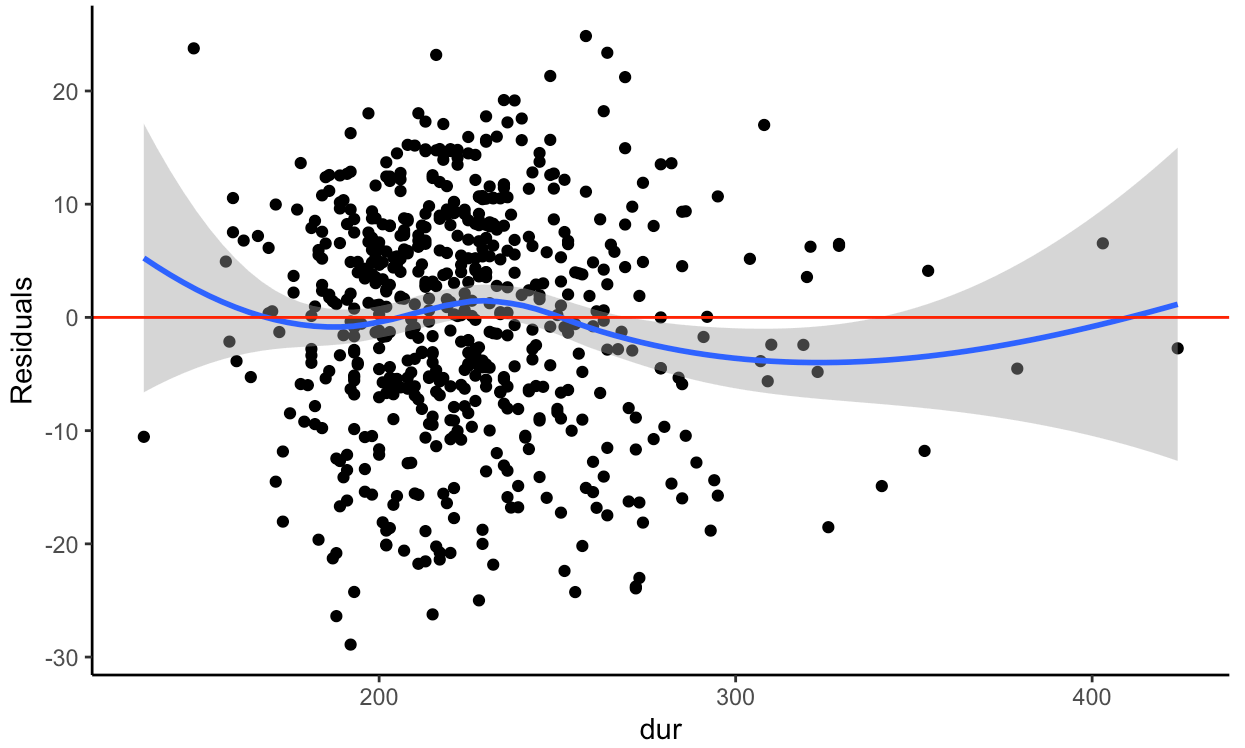

Residuals
20
10
0
-10
-20
-30
0
25
50
75
100
nrgy

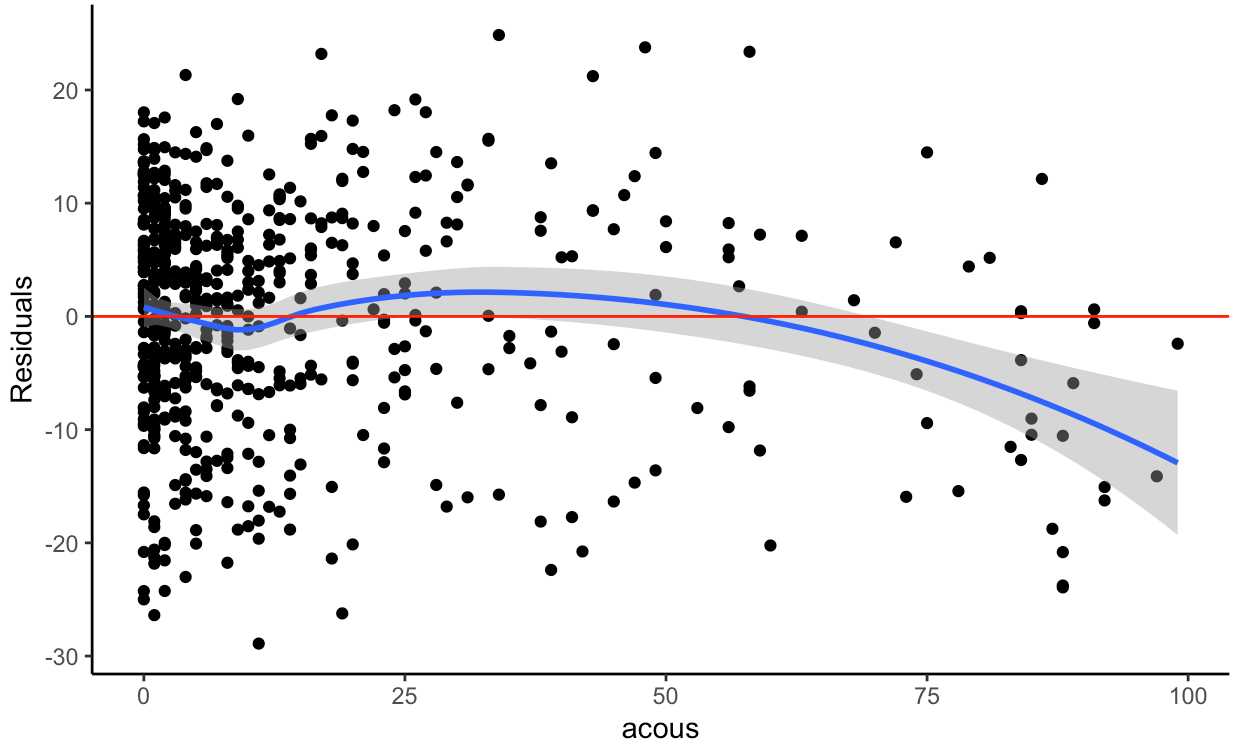

Residuals
20
10
0
-10
-20
-30
20
40
60
80
100
dnce

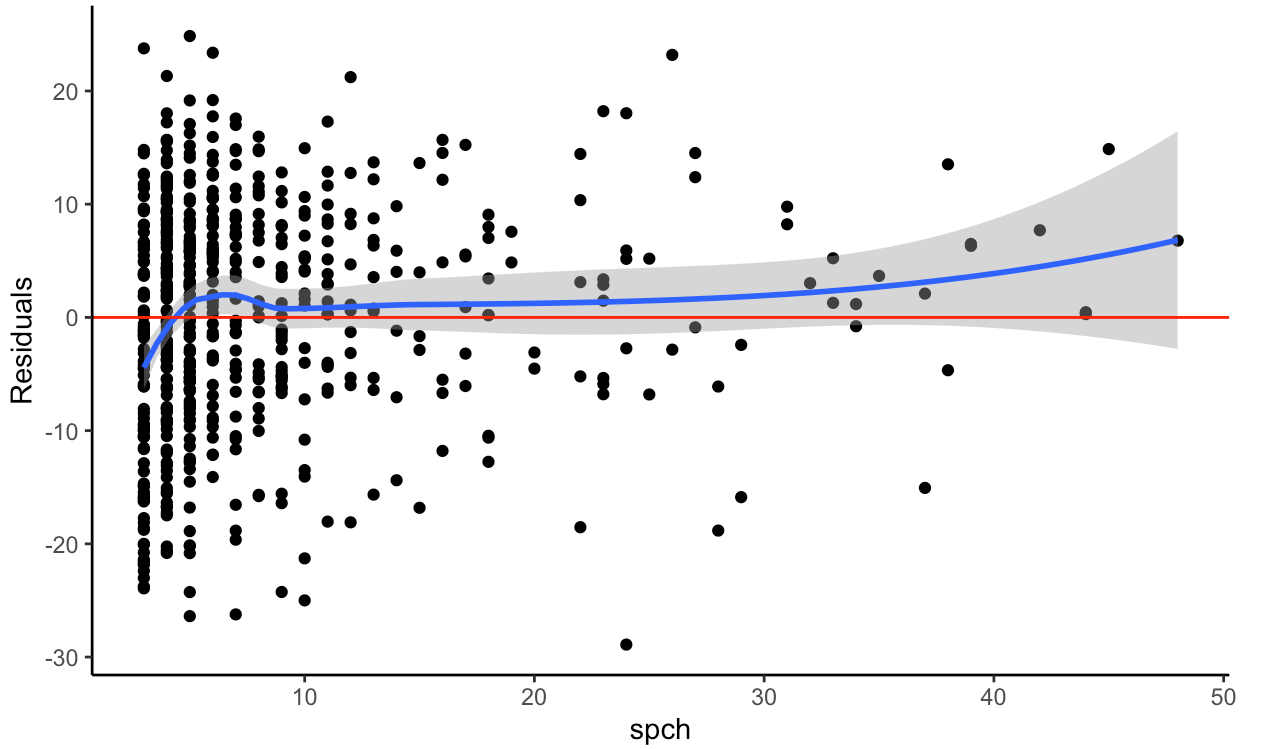

Residuals
20
10
0
-10
-20
-30
-12
-8
-4
dB

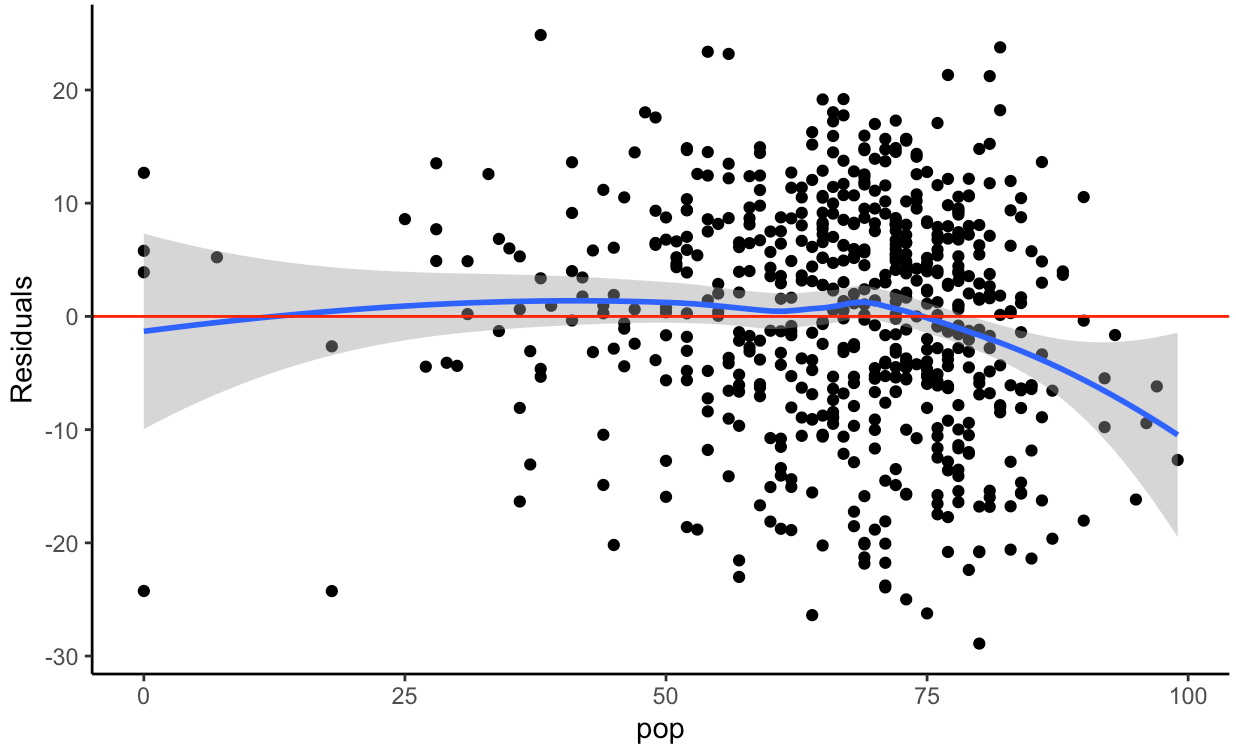

Residuals
20
10
0
-10
-20
-30
0
20
40
60
live


Residuals
20
10
0
-10
-20
-30
0
25
50
75
100
val


Residuals
20
10
0
-10
-20
-30
200
300
400
dur


Residuals
20
10
0
-10
-20
-30
0
25
50
75
100
acous


Residuals
20
10
0
-10
-20
-30
10
20
30
40
50
spch



Model	Training MAE	MAESD
OLS Model	10.13206	0.83748
Backward	8.286481	0.825095
Forward	8.383712	0.7669223
LASSO	8.32769	0.7932597

Comparing the four models, LASSO and Backward Stepwise Selection models seem to be yielding the best results as the predictions for top song energy level are closest to the test value.

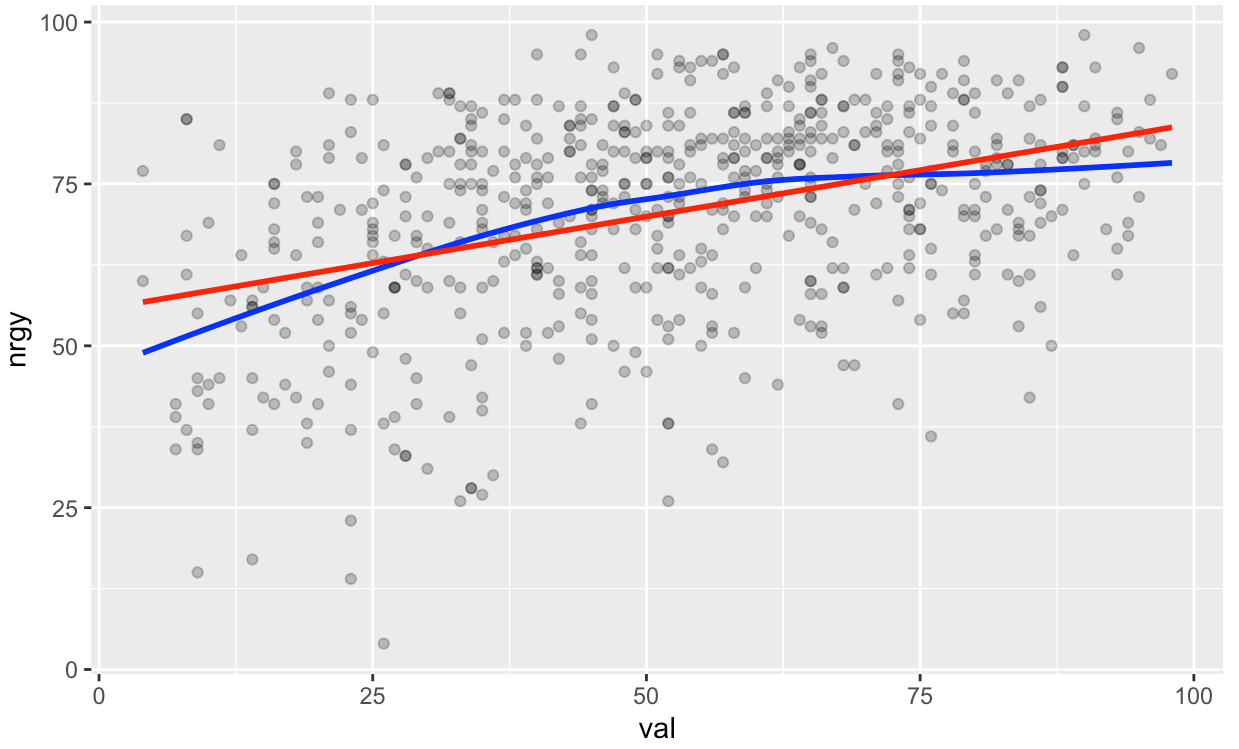
Compare insights from variable importance analyses from the different methods (stepwise and LASSO, but not OLS). Are there variables for which the methods reach consensus? What insights are expected? Surprising?

Across all models, the “top” 3 predictors for song energy level are acous (Acousticness- the higher the value the more acoustic the song is), dB(Loudness, the higher the value, the louder the song), and val(Valence, the higher the value, the more positive mood for the song). This is mostly consistent with our expectation, as when the song is louder, and more positive, the song has a higher energy level. It is a bit surprising that when a song is more acoustic, it is less energetic.

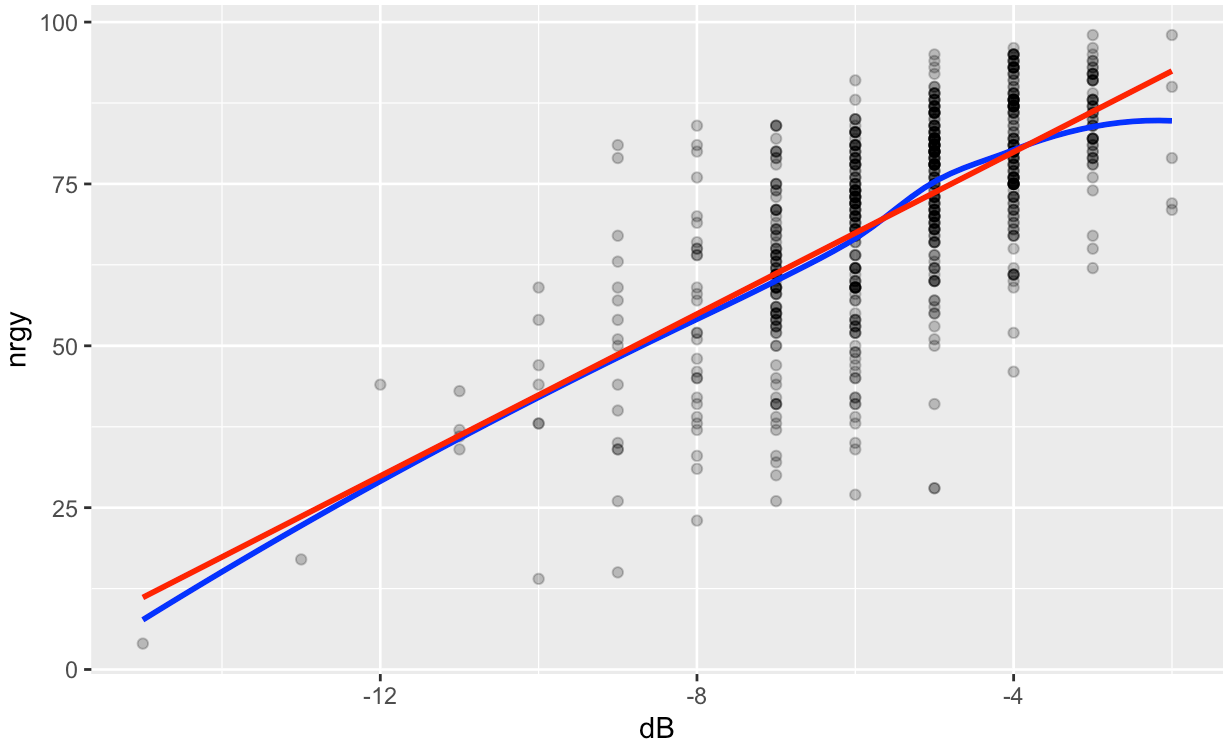
Investigation 2: Accounting for nonlinearity

Update your stepwise selection model(s) and LASSO model to use natural splines for the quantitative predictors:

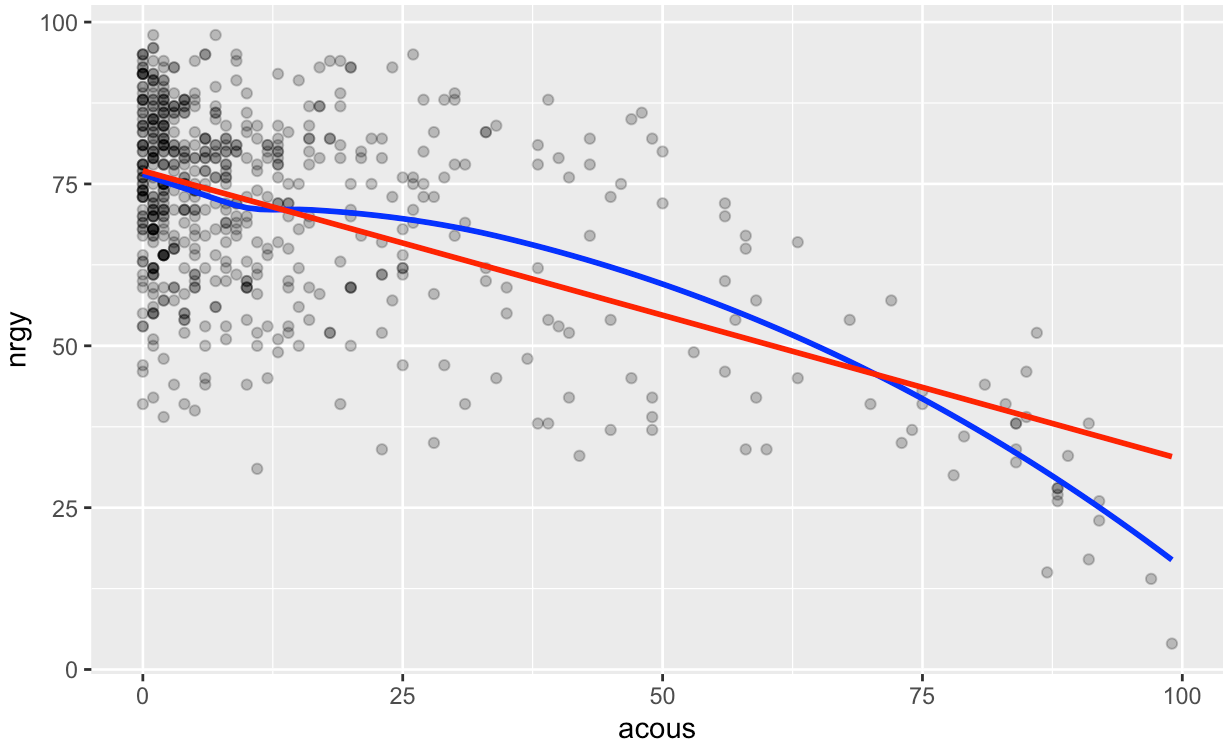
```
ggplot  (       top_spotify_new, aes     (       x =       val     , y =
nrgy    )       )       +
    geom_point(       alpha =       0.25    )       +
    geom_smooth(       color =       "blue"   , se =       FALSE    )       +
    geom_smooth(       method =       "lm"    , color =       "red"    , se =
FALSE    )
```




```
ggplot  (       top_spotify_new, aes     (       x =       dB      , y =
nrgy    )       )       +
    geom_point(       alpha =       0.25    )       +
    geom_smooth(       color =       "blue"   , se =       FALSE    )       +
    geom_smooth(       method =       "lm"    , color =       "red"    , se =
FALSE    )
```

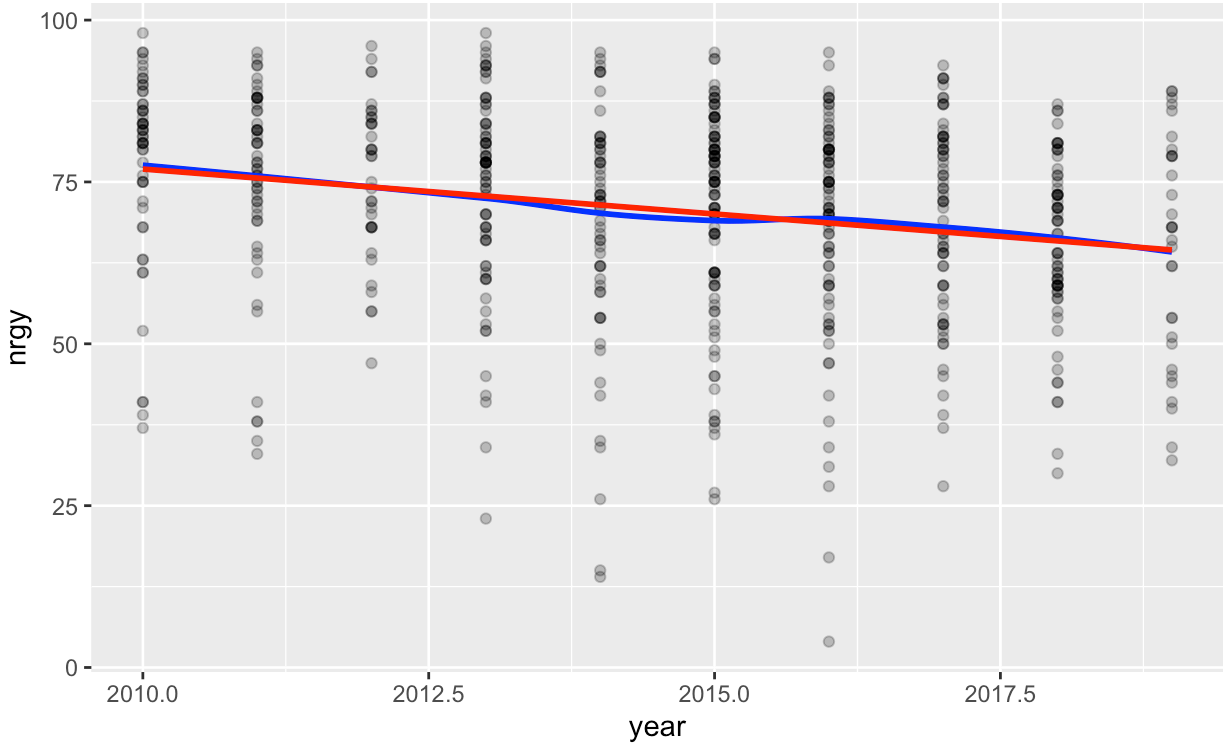



```
ggplot(top_spotify_new, aes(x = acous, y = nrgy)) +
    geom_point(alpha = 0.25) +
    geom_smooth(color = "blue", se = FALSE) +
    geom_smooth(method = "lm", color = "red", se = FALSE)
```




```
ggplot(top_spotify_new, aes(x = year, y = nrgy)) +
```

```
    geom_point(        alpha =        0.25     )        +
    geom_smooth(        color =         "blue"   , se =         FALSE    )        +
    geom_smooth(        method =         "lm"     , color =         "red"    , se =
FALSE    )
```



Backward Stepwise Selection model with natural splines

Update the Backward Stepwise Selection model to use natural splines for the quantitative predictors:

```
set.seed(        253      )
back_spline_mod <-        train    (
    nrgy      ~         ns       (        acous    , 3        )        +         ns
(        pop      , 3        )        +         ns       (        dnce     , 3        )
+         ns       (        live     , 3        )        +         ns       (        dB       ,
3        )        +         ns       (        val      , 3        )        ,
    data =         top_spotify_new,
    method =          "leapBackward",
    trControl =         trainControl(        method =         "cv"     , number =
9        , selectionFunction =         "oneSE" )        ,
    metric =         "MAE"    ,
    na.action =         na.omit
)
```

Forward Stepwise Selection model with natural splines

Update the Forward Stepwise Selection model to use natural splines for the quantitative predictors:

```
set.seed(       253     )
for_spline_mod <-       train   (
    nrgy     ~        ns      (       year     , 3       )       +       ns
(       acous    , 3       )       +       ns      (       bpm      , 3       )
+        ns      (       pop      , 3       )       +        ns      (       dnce     ,
3       )       +        ns      (       live     , 3       )       +        ns
(       spch     , 3       )       +        ns      (       dur      , 3       )
,
    data =        top_spotify_new,
    method =        "leapForward",
    trControl =        trainControl(       method =       "cv"     , number =
9       , selectionFunction =       "oneSE" )       ,
    metric =        "MAE"    ,
    na.action =        na.omit
)
```

LASSO model with natural splines

Update the LASSO model to use natural splines for the quantitative predictors:

```
set.seed(       253     )
LASSO_spline_mod <-       train   (
    nrgy     ~        ns      (       year     , 3       )       +       ns
(       acous    , 3       )       +       ns      (       bpm      , 3       )
+        ns      (       pop      , 3       )       +        ns      (       dnce     ,
3       )       +        ns      (       live     , 3       )       +        ns
(       spch     , 3       )       +        ns      (       dur      , 3       )
,
    data =        top_spotify_new,
    method =        "glmnet",
    tuneGrid =        data.frame(       alpha =       1       , lambda =
seq     (       0       , 10      , length.out =       100     )       )       ,
    trControl =        trainControl(       method =       "cv"     , number =
9       , selectionFunction =       "oneSE" )       ,
    metric =        "MAE"    ,
    na.action =        na.omit
)
```

Compare insights from variable importance analyses

Compare insights from variable importance analyses here and the corresponding results from Investigation 1. Now after having accounted for nonlinearity, have the most relevant predictors changed?

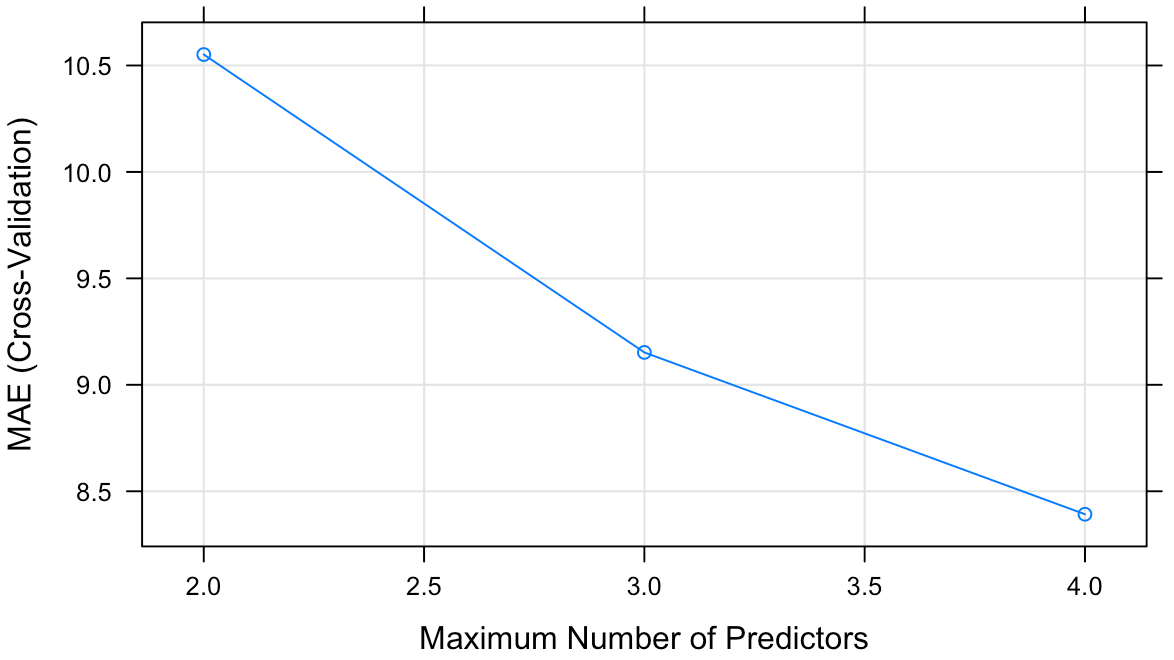
- Note that if some (but not all) of the spline terms are selected in the final models, the whole predictor should be treated as selected.

Examine Backward Stepwise Selection model with natural splines output:

```
summary (       back_spline_mod)
```

```
Subset selection object
18 Variables  (and intercept)
              Forced in Forced out
ns(acous, 3)1     FALSE      FALSE
ns(acous, 3)2     FALSE      FALSE
ns(acous, 3)3     FALSE      FALSE
ns(pop, 3)1       FALSE      FALSE
ns(pop, 3)2       FALSE      FALSE
ns(pop, 3)3       FALSE      FALSE
ns(dnce, 3)1      FALSE      FALSE
ns(dnce, 3)2      FALSE      FALSE
ns(dnce, 3)3      FALSE      FALSE
ns(live, 3)1      FALSE      FALSE
ns(live, 3)2      FALSE      FALSE
ns(live, 3)3      FALSE      FALSE
ns(dB, 3)1        FALSE      FALSE
ns(dB, 3)2        FALSE      FALSE
ns(dB, 3)3        FALSE      FALSE
ns(val, 3)1       FALSE      FALSE
ns(val, 3)2       FALSE      FALSE
ns(val, 3)3       FALSE      FALSE
1 subsets of each size up to 4
Selection Algorithm: backward
         ns(acous, 3)1 ns(acous, 3)2 ns(acous, 3)3 ns(pop, 3)1
1  ( 1 ) " "           " "           " "           " "
2  ( 1 ) " "           " "           "*"           " "
3  ( 1 ) " "           "*"           "*"           " "
4  ( 1 ) " "           "*"           "*"           " "
         ns(pop, 3)2 ns(pop, 3)3 ns(dnce, 3)1 ns(dnce, 3)2
1  ( 1 ) " "         " "         " "          " "
2  ( 1 ) " "         " "         " "          " "
3  ( 1 ) " "         " "         " "          " "
4  ( 1 ) " "         " "         " "          " "
         ns(dnce, 3)3 ns(live, 3)1 ns(live, 3)2 ns(live, 3)3
1  ( 1 ) " "          " "          " "          " "
2  ( 1 ) " "          " "          " "          " "
3  ( 1 ) " "          " "          " "          " "
4  ( 1 ) " "          " "          " "          " "
         ns(dB, 3)1 ns(dB, 3)2 ns(dB, 3)3 ns(val, 3)1 ns(val, 3)2
1  ( 1 ) " "        " "        "*"        " "         " "
2  ( 1 ) " "        " "        "*"        " "         " "
3  ( 1 ) " "        " "        "*"        " "         " "
4  ( 1 ) "*"        " "        "*"        " "         " "
         ns(val, 3)3
1  ( 1 ) " "
2  ( 1 ) " "
3  ( 1 ) " "
4  ( 1 ) " "
```

```
plot    (       back_spline_mod)
```




```
back_spline_mod$       bestTune
```

```
  nvmax
3     4
```

```
back_spline_mod$       results
```

```
  nvmax     RMSE  Rsquared       MAE    RMSESD RsquaredSD     MAESD
1     2 12.94073 0.3520225 10.551403 1.0723596 0.09809830 0.7221742
2     3 11.21314 0.5103530  9.152245 0.9208166 0.07104034 0.7258275
3     4 10.28440 0.5900269  8.392090 0.8083434 0.09138549 0.7327469
```

According to the Backward Stepwise Selection model with natural splines, the top predictors for song energy level are acous and dB.

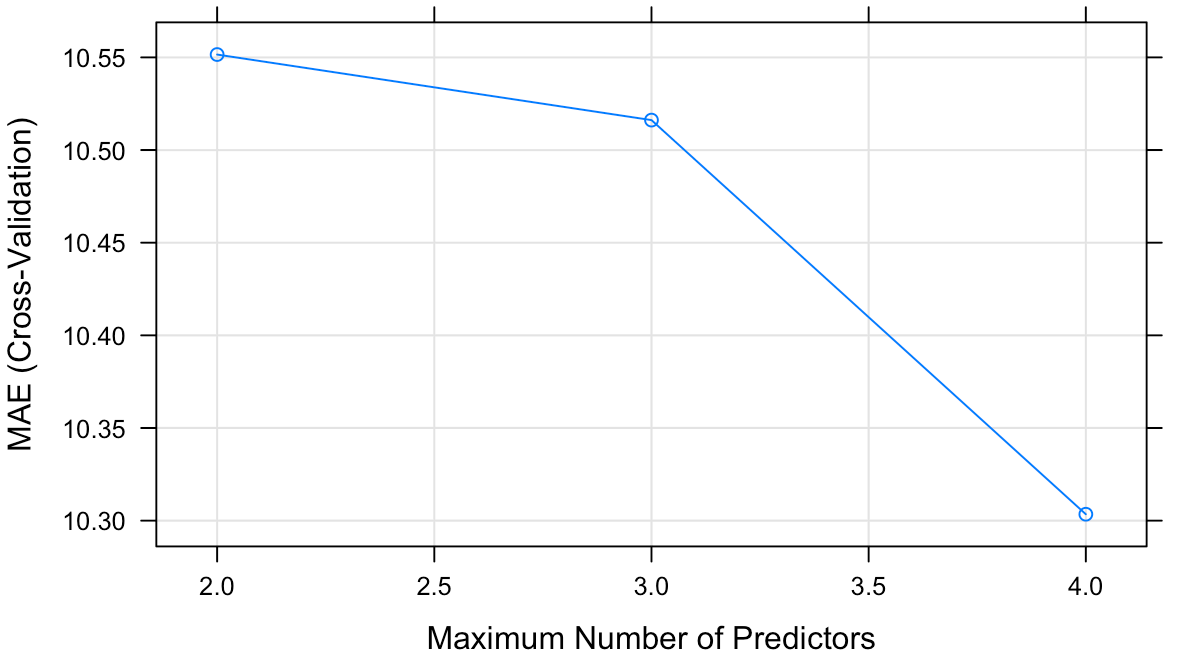
Examine Forward Stepwise Selection model output:

```
summary (       for_spline_mod)
```

```
Subset selection object
24 Variables  (and intercept)
              Forced in Forced out
ns(year, 3)1      FALSE      FALSE
ns(year, 3)2      FALSE      FALSE
ns(year, 3)3      FALSE      FALSE
ns(acous, 3)1     FALSE      FALSE
ns(acous, 3)2     FALSE      FALSE
ns(acous, 3)3     FALSE      FALSE
ns(bpm, 3)1       FALSE      FALSE
ns(bpm, 3)2       FALSE      FALSE
```

```
ns(bpm, 3)3       FALSE      FALSE
ns(pop, 3)1       FALSE      FALSE
ns(pop, 3)2       FALSE      FALSE
ns(pop, 3)3       FALSE      FALSE
ns(dnce, 3)1      FALSE      FALSE
ns(dnce, 3)2      FALSE      FALSE
ns(dnce, 3)3      FALSE      FALSE
ns(live, 3)1      FALSE      FALSE
ns(live, 3)2      FALSE      FALSE
ns(live, 3)3      FALSE      FALSE
ns(spch, 3)1      FALSE      FALSE
ns(spch, 3)2      FALSE      FALSE
ns(spch, 3)3      FALSE      FALSE
ns(dur, 3)1       FALSE      FALSE
ns(dur, 3)2       FALSE      FALSE
ns(dur, 3)3       FALSE      FALSE
1 subsets of each size up to 3
Selection Algorithm: forward
         ns(year, 3)1 ns(year, 3)2 ns(year, 3)3 ns(acous, 3)1
1  ( 1 ) " "          " "          " "          " "
2  ( 1 ) " "          " "          " "          " "
3  ( 1 ) " "          " "          " "          " "
         ns(acous, 3)2 ns(acous, 3)3 ns(bpm, 3)1 ns(bpm, 3)2
1  ( 1 ) " "           "*"           " "         " "
2  ( 1 ) "*"           "*"           " "         " "
3  ( 1 ) "*"           "*"           " "         " "
         ns(bpm, 3)3 ns(pop, 3)1 ns(pop, 3)2 ns(pop, 3)3 ns(dnce, 3)1
1  ( 1 ) " "         " "         " "         " "         " "
2  ( 1 ) " "         " "         " "         " "         " "
3  ( 1 ) " "         " "         " "         " "         " "
         ns(dnce, 3)2 ns(dnce, 3)3 ns(live, 3)1 ns(live, 3)2
1  ( 1 ) " "          " "          " "          " "
2  ( 1 ) " "          " "          " "          " "
3  ( 1 ) " "          " "          " "          " "
         ns(live, 3)3 ns(spch, 3)1 ns(spch, 3)2 ns(spch, 3)3
1  ( 1 ) " "          " "          " "          " "
2  ( 1 ) " "          " "          " "          " "
3  ( 1 ) " "          " "          "*"          " "
         ns(dur, 3)1 ns(dur, 3)2 ns(dur, 3)3
1  ( 1 ) " "         " "         " "
2  ( 1 ) " "         " "         " "
3  ( 1 ) " "         " "         " "
```

```
plot    (       for_spline_mod)
```




```
for_spline_mod$       bestTune
```

```
  nvmax
2     3
```

```
for_spline_mod$       results
```

```
  nvmax     RMSE  Rsquared      MAE    RMSESD RsquaredSD     MAESD
1     2 12.86781 0.3498257 10.55154 0.9235936 0.10765745 0.6993457
2     3 12.92971 0.3449085 10.51615 0.9716964 0.09693563 0.6348611
3     4 12.69644 0.3685762 10.30346 0.8366573 0.10280846 0.6553263
```

According to the Forward Stepwise Selection model with natural splines, the top predictor for song energy level is acous.

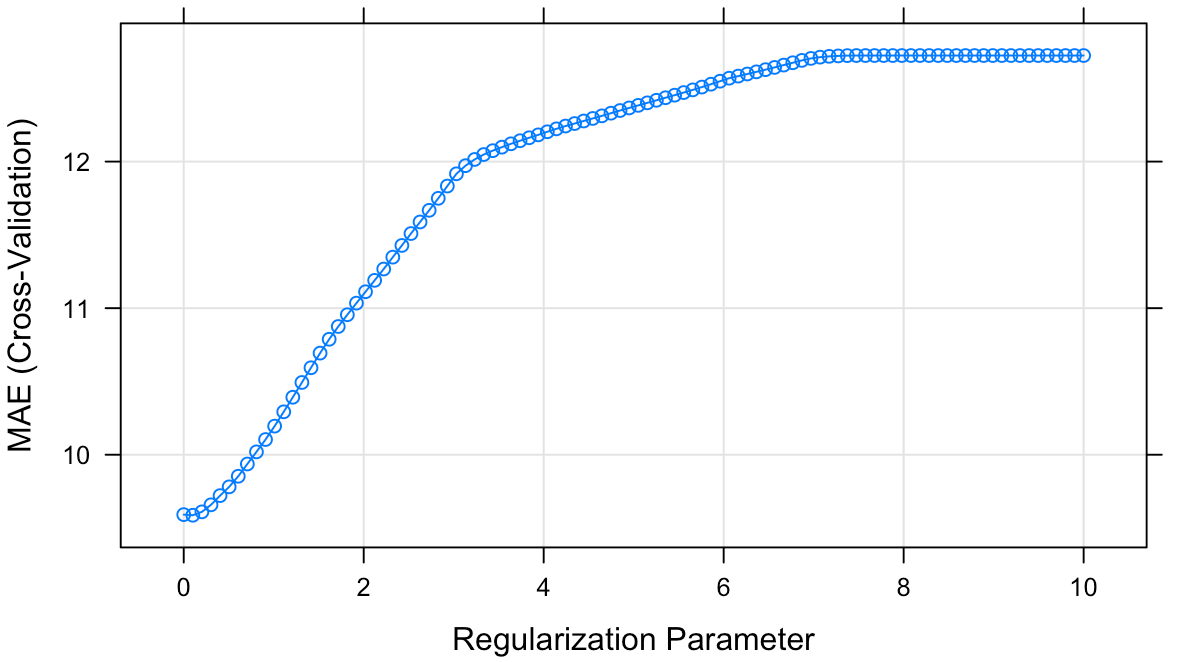
Examine LASSO model with natural splines output:

```
summary (       LASSO_spline_mod)
```

```
           Length Class      Mode
a0           72   -none-     numeric
beta       1728   dgCMatrix  S4
df           72   -none-     numeric
dim           2   -none-     numeric
lambda       72   -none-     numeric
dev.ratio    72   -none-     numeric
nulldev       1   -none-     numeric
npasses       1   -none-     numeric
jerr          1   -none-     numeric
offset        1   -none-     logical
```

```
call           5   -none-     call
nobs           1   -none-     numeric
lambdaOpt      1   -none-     numeric
xNames        24   -none-     character
problemType    1   -none-     character
tuneValue      2   data.frame list
obsLevels      1   -none-     logical
param          0   -none-     list
```

```
plot    (       LASSO_spline_mod)
```




```
LASSO_spline_mod$       bestTune
```

```
  alpha    lambda
6     1 0.5050505
```

```
# LASSO_spline_mod$results
```

The lambda value provided by the LASSO model with splines is 0.5050505.

GAM with LOESS terms

Fit a GAM using LOESS terms using the set of variables deemed to be most relevant based on your investigations so far.

- How does test performance of the GAM compare to other models you explored?
- Do you gain any insights from the GAM output plots for each predictor?

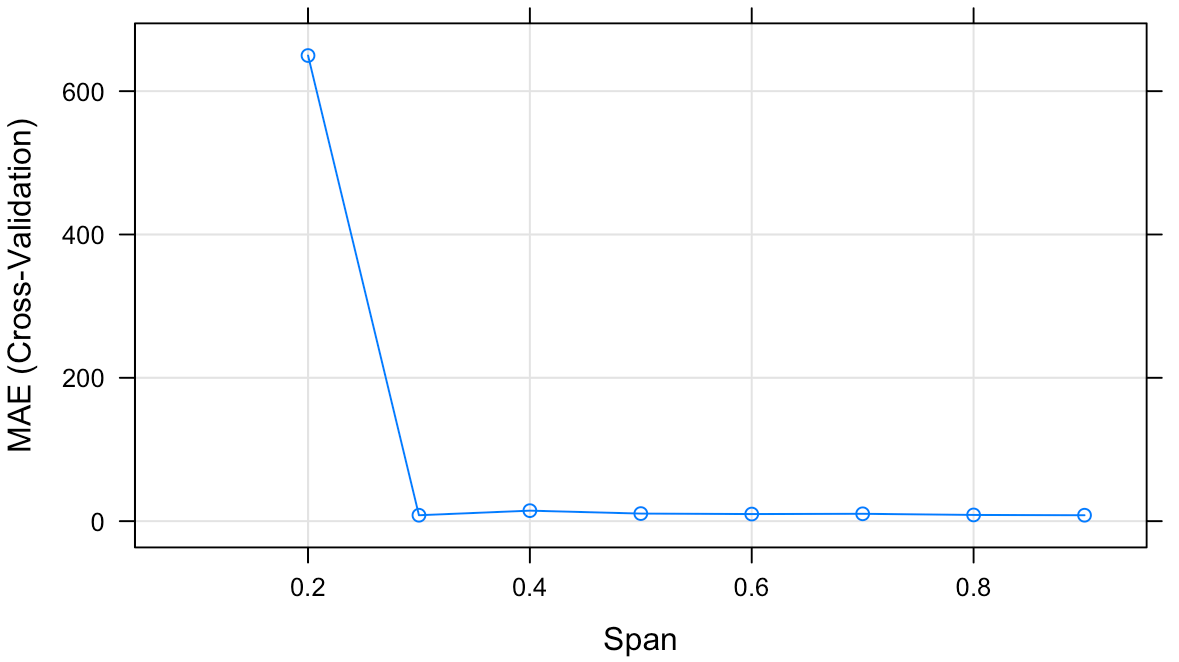
```
set.seed(       253     )
gam_mod  <-       train   (
    nrgy     ~        acous    +        val      +        dB      ,
    data =        top_spotify_new,
    method =        "gamLoess",
    tuneGrid =         data.frame(       degree =        1       , span =        seq
(       0.1     , 0.9      , by =        0.1      )       )       ,
    trControl =        trainControl(       method =        "cv"     , number =
9       , selectionFunction =        "best"   )       ,
    metric =        "MAE"    ,
    na.action =        na.omit
)
```

Examine GAM with LOESS output:

```
gam_mod $       results [       3       ,]
```

```
  degree span     RMSE  Rsquared      MAE   RMSESD RsquaredSD
3      1  0.3 10.20914 0.5906928 8.252745 0.892152  0.1020878
      MAESD
3 0.8103724
```

```
plot    (       gam_mod )
```

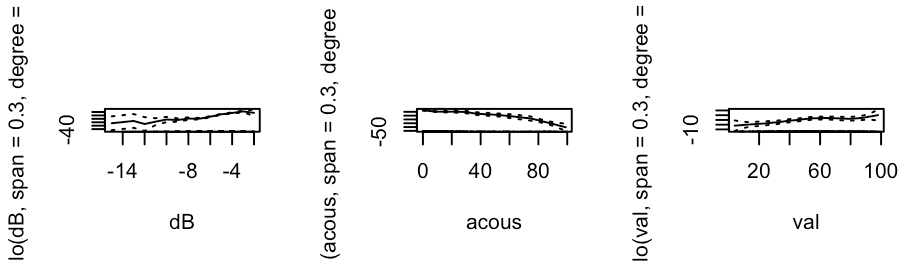



```
#Metrics for the best model
gam_mod $       results  %>%
    filter  (       span    ==      gam_mod $       bestTune$       span    )
```

```
  degree span     RMSE  Rsquared      MAE   RMSESD RsquaredSD
1      1  0.3 10.20914 0.5906928 8.252745 0.892152  0.1020878
```

```
      MAESD
1 0.8103724
```

```
#Graphing Each Predictor
par     (       mfrow =         c       (       3       ,4      )       )
# Sets up a grid of plots
plot    (       gam_mod $       finalModel, se =        TRUE    )
# Dashed lines are +/- 2 SEs
```



GAM with a span of 0.3 offers a MAE of 8.252745, indicating that we our predictions for top song energy level would be off by 8.368585 percentage points in this case. This result is actually better than all four previous models fitted in the first section.

Summarize investigations

Decide on an overall best model based on your investigations so far. To do this, make clear your analysis goals. Predictive accuracy? Interpretability? A combination of both?

Overall, based on the output given by all of the models we fitted above, it seems that a GAM with LOESS model achieves the lowest MAE for our dataset. For our analysis, since we want to correctly predict the energy level of a popular song, we care about the predictive accuracy of the model. We are also interested in knowing what contributes to an energetic song, thus interpretability is also essential for the model. Therefore splines doesn't seem the most straightforward choice for us, whereas either GAM with LOESS or LASSO seems like a better option.

Societal impact

Are there any harms that may come from your analyses and/or how the data were collected? What cautions do you want to keep in mind when communicating your work?

Our models takes a harmless look at the deciding elements of an energetic song, as under the environment of a global pandemic where social interactions are limited, it is important to look for means to maintain a positive mood, and it seems that listening to uplifting pop music is a favorable way to do so. Given our dataset, though, since the source is Spotify and Billboard, our scope of pop music is limited and may result in a certain pattern in our predictions. We want to caution that good music choice should by no means be limited, and it should always be optimal to listen to whatever one's heart desires.

Classification analysis (Methods)

We used logistic regression and random forest for building classification models.

Logistic Regression

We converted the predictor "pop" to categorical, assigning the observations with value above 75 to be top songs.

```
top_spotify_new$       IsPop     <-       "NO"
top_spotify_new$       IsPop    [       top_spotify_new$       pop       >=
75       ]         <-        "YES"
table    (       top_spotify_new$       IsPop    )
table    (       top_spotify_new$       pop      )
top_spotify_new$       IsPop     <-       factor   (       top_spotify_new$
IsPop    )
```

We then fit the logistic regression model predicting whether a given song is a popular song with all other predictors. We selected the metrics Accuracy so that the model we fit would prioritize making the most accurate predictions.

```
set.seed(       253      )
logistic_mod <-       train    (
    IsPop     ~         .        -       pop      ,
    data =        top_spotify_new,
    method =        "glm"    ,
    family =        "binomial",
    trControl =        trainControl(       method =        "cv"     , number =
10       )       ,
    metric =        "Accuracy",
    na.action =        na.omit
```

```
)
summary (       logistic_mod$       results )
```

```
 parameter           Accuracy          Kappa
Length:1           Min.   :0.7124   Min.   :0.2109
Class :character   1st Qu.:0.7124   1st Qu.:0.2109
Mode  :character   Median :0.7124   Median :0.2109
                   Mean   :0.7124   Mean   :0.2109
                   3rd Qu.:0.7124   3rd Qu.:0.2109
                   Max.   :0.7124   Max.   :0.2109
  AccuracySD         KappaSD
Min.   :0.02915   Min.   :0.07924
1st Qu.:0.02915   1st Qu.:0.07924
Median :0.02915   Median :0.07924
Mean   :0.02915   Mean   :0.07924
3rd Qu.:0.02915   3rd Qu.:0.07924
Max.   :0.02915   Max.   :0.07924
```

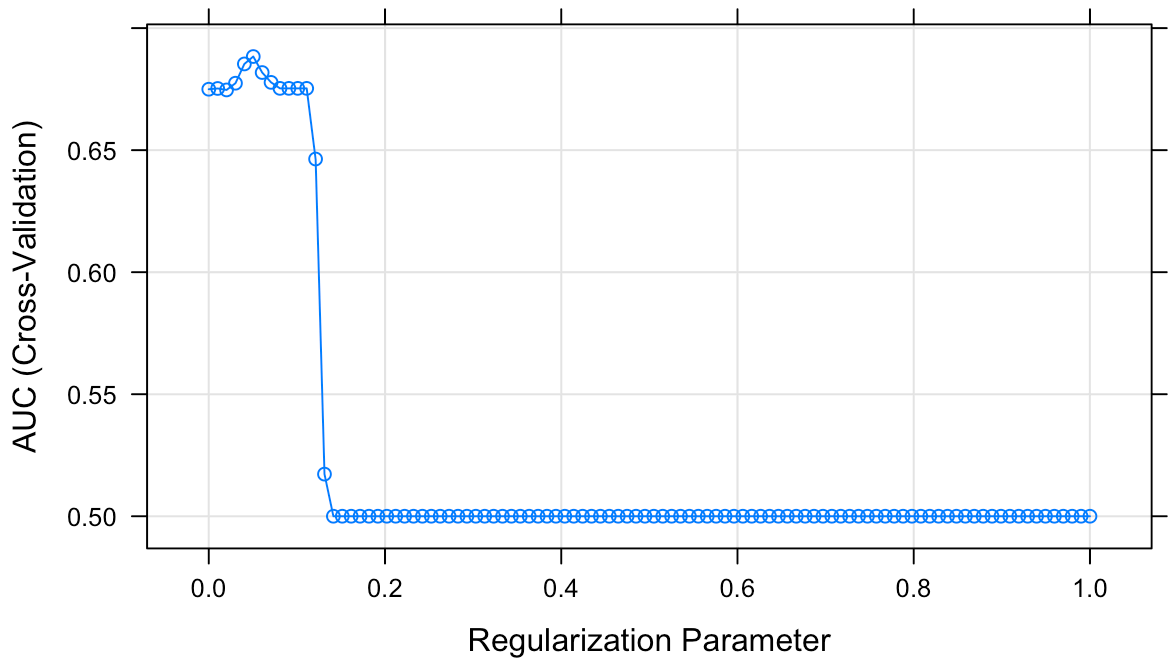
```
coefficients(       logistic_mod$       finalModel)       %>%       exp      (
)
```

```
  (Intercept)          year           bpm          nrgy          dnce
1.508016e-192  1.246311e+00  1.000621e+00  9.717299e-01  1.007890e+00
           dB          live           val           dur         acous
 1.147383e+00  9.915256e-01  1.006670e+00  9.968091e-01  1.001887e+00
         spch
 9.881283e-01
```

We also fit the LASSO logistic regression, gaining insight about variable importance.

```
twoClassSummaryCustom <-       function (       data     , lev      =       NULL     ,
model     =       NULL     )       {
    if       (       length   (       lev     )       >       2       )
{
        stop     (       paste    (       "Your outcome has", length   (       lev
)       , "levels. The twoClassSummary() function isn't appropriate.")       )
    }
    caret    :::      requireNamespaceQuietStop(       "pROC"   )
    if       (       !       all     (       levels   (       data     [       ,
"pred"   ]       )       ==       lev     )       )       {
        stop     (       "levels of observed and predicted data do not match")
    }
    rocObject <-       try      (       pROC     ::       roc      (       data     $
obs      , data     [       , lev      [       1       ]       ]       , direction
=       ">"      ,
        quiet =       TRUE     )       , silent =       TRUE     )
    rocAUC    <-       if       (       inherits(       rocObject, "try-error")
)
        NA
    else      rocObject$       auc
    out       <-       c       (       rocAUC   , sensitivity(       data     [       ,
"pred"   ]       , data     [       , "obs"    ]       ,
        lev      [       1       ]       )       , specificity(       data     [       ,
```

```
"pred" ] , data [ , "obs" ] , lev [ 2 ]
) )
    out2 <- postResample( data [ , "pred" ] , data
[ , "obs" ] )
    out <- c ( out , out2 [ 1 ]
)
    names ( out ) <- c ( "AUC" , "Sens" ,
"Spec" , "Accuracy")
    out
}
set.seed( 253 )
lasso_logistic_mod <- train (
    IsPop ~ . - pop ,
    data = top_spotify_new,
    method = "glmnet",
    family = "binomial",
    tuneGrid = data.frame( alpha = 1 , lambda =
seq ( 0 , 1 , length.out = 100 ) ) ,
    trControl = trainControl( method = "cv" , number =
10 , selectionFunction = "oneSE" , classProbs = TRUE ,
summaryFunction = twoClassSummaryCustom) ,
    metric = "AUC" ,
    na.action = na.omit
)

plot ( lasso_logistic_mod)
```



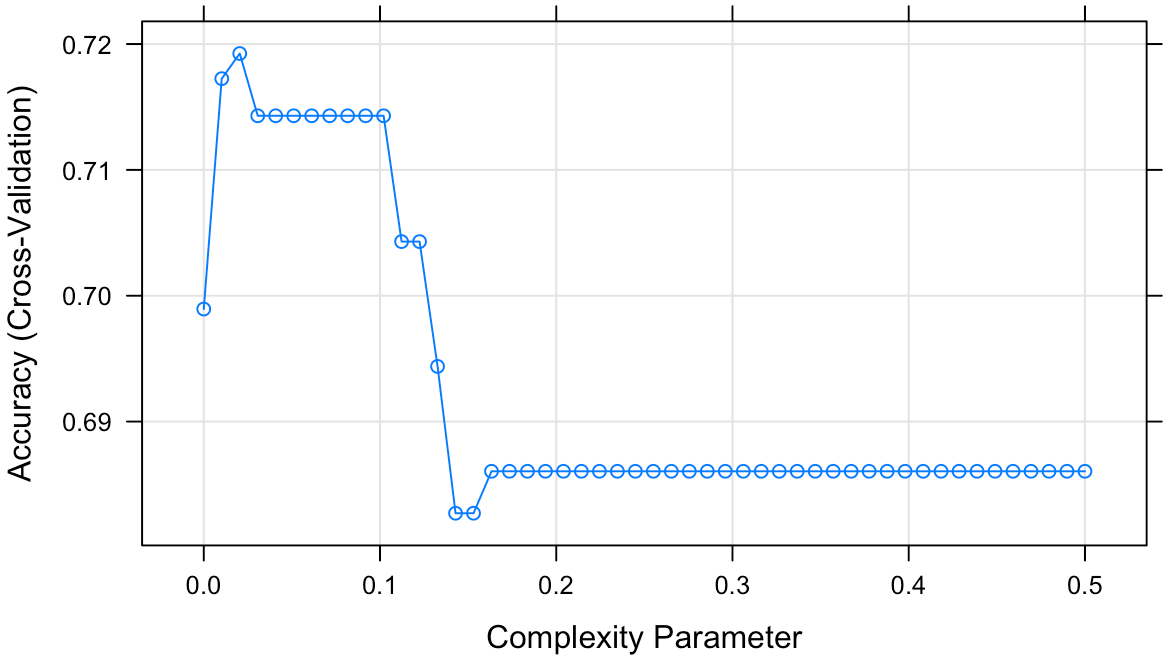

```
lasso_logistic_mod$ bestTune
lasso_logistic_mod$ results
lasso_logistic_mod$ results %>%
    filter ( lambda == lasso_logistic_mod$ bestTune$ lambda
```

```
)
plot     (        lasso_logistic_mod$        finalModel, xvar =         "lambda", label
=         TRUE      , col =          rainbow (         20       )         , ylim =
c        (         -        0.5      ,7        )        )

rownames(        lasso_logistic_mod$        finalModel$        beta     )        [
c        (        5        ,3        ,1        )        ]
```

Trees and Random Forest

We fit trees and random forest to make predictions as well, using all other predictors to predict whether an observation is a popular song or not. The metrics we selected is Accuracy, so that the model would prioritize making accurate predictions.

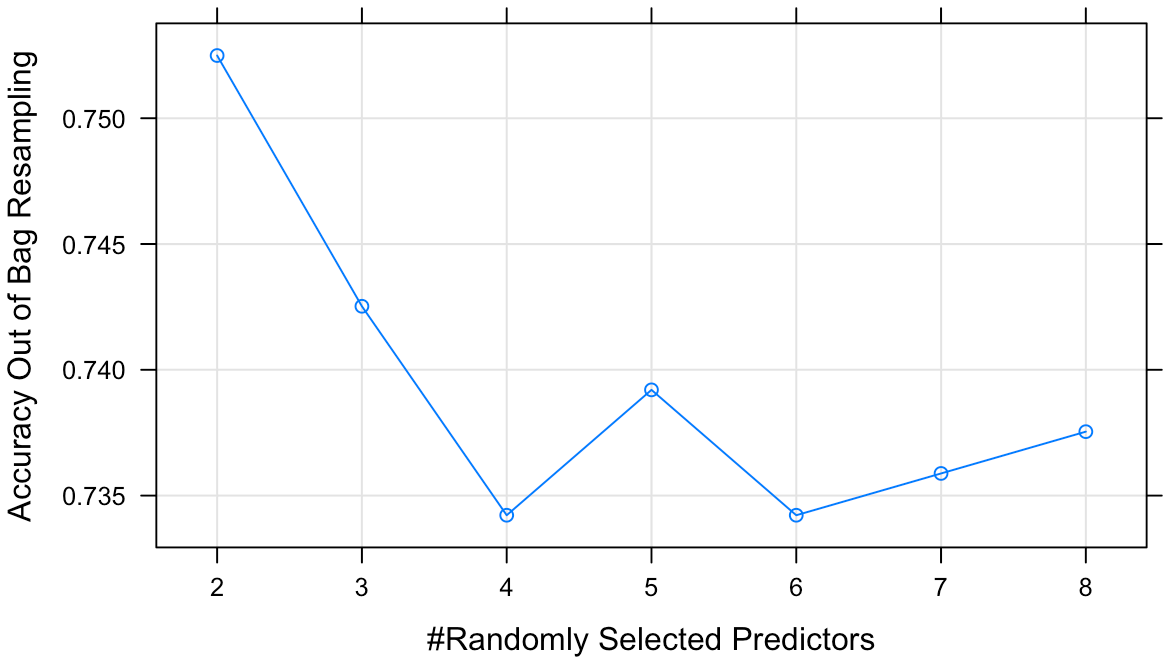
```
set.seed(        253      )
tree_mod <-        train    (
    IsPop     ~          .         -         pop       ,
    data =          top_spotify_new,
    method =          "rpart"  ,
    tuneGrid =          data.frame(         cp =          seq      (        0         , 0.5      ,
length.out =          50       )        )         ,
    trControl =          trainControl(         method =          "cv"     , number =
10       , selectionFunction =          "oneSE" )        ,
    metric =          "Accuracy",
    na.action =          na.omit
)

plot     (        tree_mod)
```



```
tree_mod$       results   %>%
    filter  (       cp      ==      tree_mod$       bestTune$       cp      )
```

```
        cp  Accuracy    Kappa AccuracySD KappaSD
1 0.1020408 0.7143012 0.181339 0.03461561 0.13422
```

```
rf_mod    <-       train    (
    IsPop     ~         .       -       pop      ,
    data =         top_spotify_new,
    method =         "rf"     ,
    tuneGrid =         data.frame(       mtry =         c       (       2       ,3
,4       ,5       ,6       ,7       ,8       )       )       ,
    trControl =         trainControl(       method =        "oob"    , selectionFunction
=        "best"   )       ,
    metric =         "Accuracy",
    ntree =         750      ,
# To force fitting 1000 trees (can help with stability of results)
    na.action =         na.omit
)
plot    (       rf_mod   )
```




```
rf_mod   $       results
```

```
   Accuracy     Kappa mtry
1 0.7524917 0.3347524    2
2 0.7425249 0.3147939    3
3 0.7342193 0.3007419    4
4 0.7392027 0.3082283    5
5 0.7342193 0.2961637    6
6 0.7358804 0.2947911    7
7 0.7375415 0.3026788    8
```

```
rf_mod  $       finalModel
```

```
Call:
 randomForest(x = x, y = y, ntree = 750, mtry = min(param$mtry,      ncol(x)))
               Type of random forest: classification
                     Number of trees: 750
No. of variables tried at each split: 2

        OOB estimate of  error rate: 26.08%
Confusion matrix:
     NO YES class.error
NO  384  29  0.07021792
YES 128  61  0.67724868
```

Our model is better at identifying songs that are not popular. In our context, it helps us avoid bad songs, which is preferable than the more lenient alternative which is more likely to falsely categorize a song as popular.

```
var_imp_rf <-       randomForest::      importance(       rf_mod  $       finalModel
)

# Sort by importance with dplyr's arrange()
var_imp_rf <-       data.frame(
        predictor =         rownames(       var_imp_rf)       ,
        MeanDecreaseGini =         var_imp_rf[       ,"MeanDecreaseGini"]
    )       %>%
    arrange (       desc    (       MeanDecreaseGini)       )

# Top 10
head    (       var_imp_rf, 10      )
```

```
     predictor MeanDecreaseGini
year       year         34.01757
val         val         29.91566
dur         dur         29.62221
nrgy       nrgy         29.08262
bpm         bpm         27.71691
dnce       dnce         26.28274
acous     acous         24.00443
live       live         23.57061
spch       spch         18.39002
dB           dB         15.18356
```

It seems that the most important predictor, given contributions to decreasing the Gini index, is year, which is pretty interesting. This tells us that knowing what year the song is released would offer us much insight into whether the song is likely to be popular.

Classification Analysis (Results- Variable Importance)

For our logistic regression model, we utilized a LASSO logistic regression to gain insight on variable importance. The results demonstrate that dnce, and bpm are the most important variables for predicting the popularity of a song. These results are sensible because it is plausible that more upbeat songs that you can dance to will be valuable traits that may lead a song to be more popular.

In our random forest model, it shows that year is by far the most important variable for predicting song popularity, which also corresponds to the most important variable as determined by the variable importance measure of a single decision tree. This is because it lowers to Gini index the most on average for all the trees. Year is not very insightful, though, because it is possible that the popular songs featured in this dataset came more from particular years than others. It doesn't really help us predict the future popularity of a song. More interestingly, the energy displayed by a song has the second most meaningful mean decrease in the Gini index. Again, this is sensible because the goal of a song often times is to portray energy to its listener, so it makes sense that songs that accomplish this goal would be more popular. 22

Classification analysis (Summary)

Compare models

We have compared a logistic regression and a decision tree model. To complement the models, we have ran a LASSO logistic regression and a random forest. We are trying to predict whether a song will be relatively popular. We have created our own binary variable with a threshold of > 75 in pop to be considered popular (IsPop = YES). In all models, the most important variable seemed to be year. In this context, songs released in a certain year seem to be the most popular. Other important variables were energy levels and dance-ibility, both of which intuitively make sense as they would be more commonly enjoyed among music listeners.

Evaluation metrics

Logistic Regression Accuracy: 0.7124414

Logistic Regression Accuracy SD: 0.0291491

Lasso Logistic Regression Accuracy: 0.6860489

Lasso Logistic Regression Accuracy SD: 0.003961554

Lasso Logistic Regression AUC: 0.6715804

Lasso Logistic Regression AUC SD: 0.07519223

Decision Tree Accuracy: 0.7143012

Decision Tree Accuracy SD: 0.03461561

Random Forest Accuracy: 0.7524917

Random Forest Confusion Matrix: NA

```
PREDICTED

  NO YES class.error
```

NO 380 33 0.07990315

YES 122 67 0.64550265

NIR: (413)/(413+189) = 68.6%

The NIR is calculated using the whole training set.

Broadly summarize conclusions from looking at these evaluation metrics and their measures of uncertainty:

We can see that the model that gives us the least amount of variance is lasso logistic regression. However, we note that we are not trying to build the model with the smallest variance, as the variance is just a way to look at the uncertainty of in estimation of test performance, which is not so high when using lasso logistic regression. Random Forest seems to have the highest accuracy. With an OOB estimate error rate of 25.75%, this is reflective of our accuracy.

Overall most preferable model

The overall most preferable model would be our random forest model. The accuracy in this model far outweighs all other models at an accuracy of 75.25% Random forests also provide out of bag error estimations, which give us an accountable measurement of error in our model.

With an overall accuracy of 75.2%, and a NIR of 68.6%, we believe this model shows an acceptable amount of error.

If using OOB error estimation, display the test (OOB) confusion matrix, and use it to interpret the strengths and weaknesses of the final model:

We can see that we have a sensitivity of (67)/(67+112) = 37.43% and a specificity of (380)/(380+33) = 92.01%. Our model is good at correctly predicting songs that won't be as popular. However, our model is bad at correctly predicting songs that won't be popular.

```
PopularSongs <-       top_spotify_new[       top_spotify_new$       pop       >=
75       , ]
table    (       PopularSongs$       year     )
```

```
2010 2011 2012 2013 2014 2015 2016 2017 2018 2019
   9   10   10   16   12   24   23   25   32   28
```

```
table    (       top_spotify_new$       year     )
```

```
2010 2011 2012 2013 2014 2015 2016 2017 2018 2019
  51   53   35   71   58   95   79   65   64   31
```

We can see that although years are fairly balanced, the years when looking at popular songs seem to be skewed towards later years. This means makes sense as the more “popular” songs seem to be the more recent ones.