Master Thesis - “Predicting the popularity of songs based on their audio features”

Student: Julien Pflüger Student number: 571630 Study program: MScBA Business Analytics & Management Rotterdam School of Management (EUR)

Loading the libraries.

Loading the tracks-dataset into the global environment.

Looking at different general characteristics of the dataset, such as the number of unique tracks and artists as well as the minimum duration of any track contained in the set to ensure that there are none mistakenly included that have a length of 0.

```
# check the number of unique track IDs and artists
length(unique(tracks$id))
```

```
## [1] 586672
```

```
length(unique(tracks$artists))
```

```
## [1] 114030
```

```
# check the shortest duration of any track contained in the dataset to ensure that there are
 none with 0 length
min(tracks$duration_ms)
```

```
## [1] 3344
```

Standardize the date-format for all rows by removing “day” and “month”, thus only keeping the year information.

Subset more contemporary songs, released since 2000 and thus from 2000-2021. The reasoning behind this subset is to account for differences in taste and popular songs throughout history e.g. around 1920 the most popular songs belonged to the genres “Operas” and “Marches” while around 2020 the most popular genres are arguably “Hip-Hop/Rap” and “Techno”.

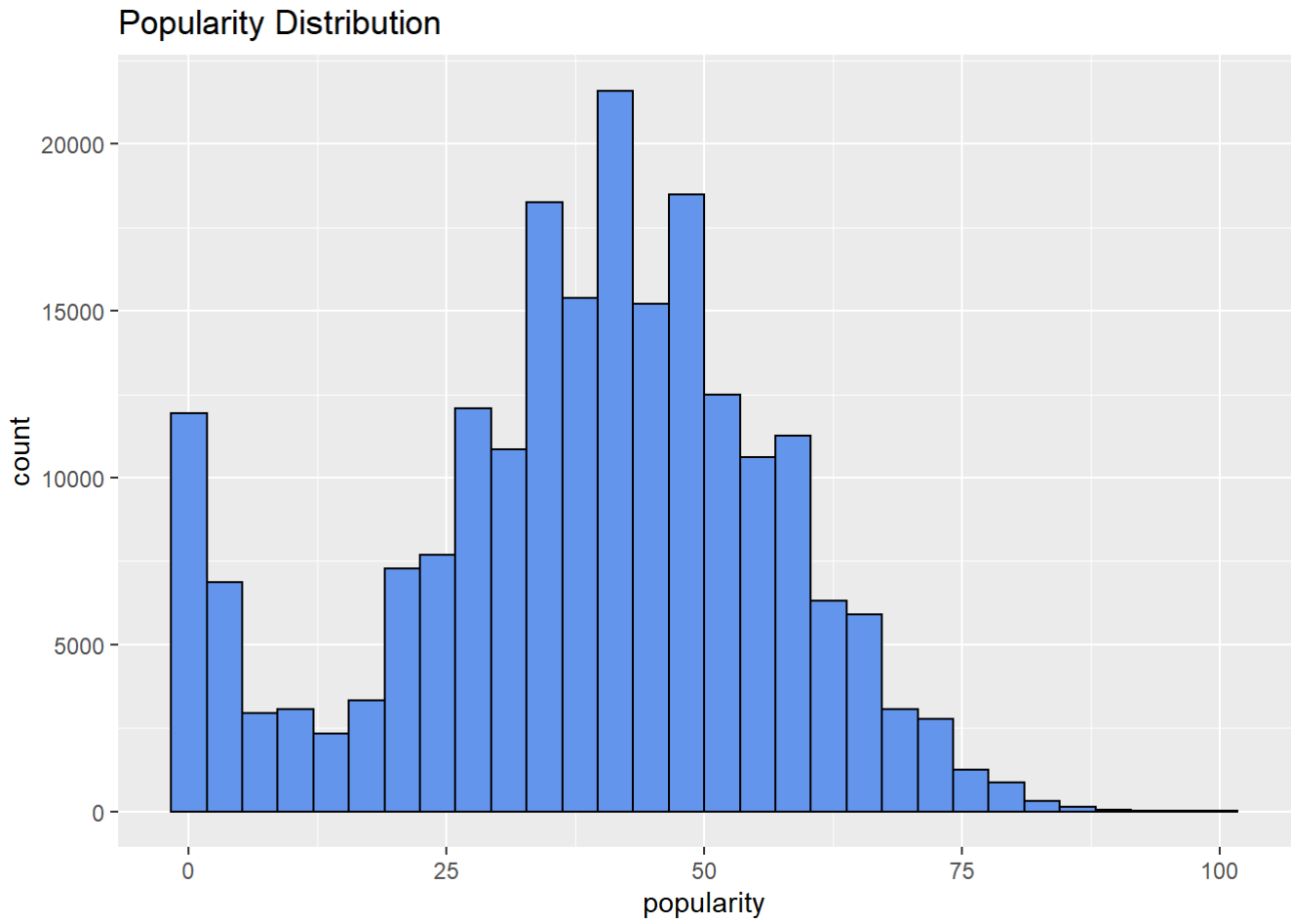
```
tracks.2k <- subset(tracks, release_year >= "2000")
```

Transform the duration of a song from milliseconds to seconds for sake of interpretability (e.g., looking at the descriptive statistics).

Filter out only those variables of interest.

Creating a histogram of the frequency of each popularity score over the subset tracks.2k.

```
ggplot(data = tracks.2k, aes(x=popularity)) + geom_histogram(bins = 30, fill = "cornflowerblu
e", color = "black") + ggtitle("Popularity Distribution")
```




```
# checking the frequency of popularity in terms of counts
tracks.2k %>%
  group_by(popularity) %>%
  summarise(counts = n())
```

```
## # A tibble: 101 x 2
##    popularity counts
##         <dbl>  <int>
##  1          0   8910
##  2          1   3029
##  3          2   2374
##  4          3   1853
##  5          4   1361
##  6          5   1280
##  7          6   1035
##  8          7   1008
##  9          8    914
## 10          9    832
## # ... with 91 more rows
```

As can be seen, the overall distribution of popularity seems relatively normal. However, several songs seem to have a popularity rank of 0. Considering that a score of 0 infers that the song has neither been played recently nor frequently, songs with a score of 0 are removed from the dataset.

Looking at the new number of unique artists and tracks contained in the updated sample.

```
# checking the number of unique songs and artists of the filtered subset
length(unique(tracks.2k$id))
```

```
## [1] 203382
```

```
length(unique(tracks.2k$artists))
```

```
## [1] 67364
```

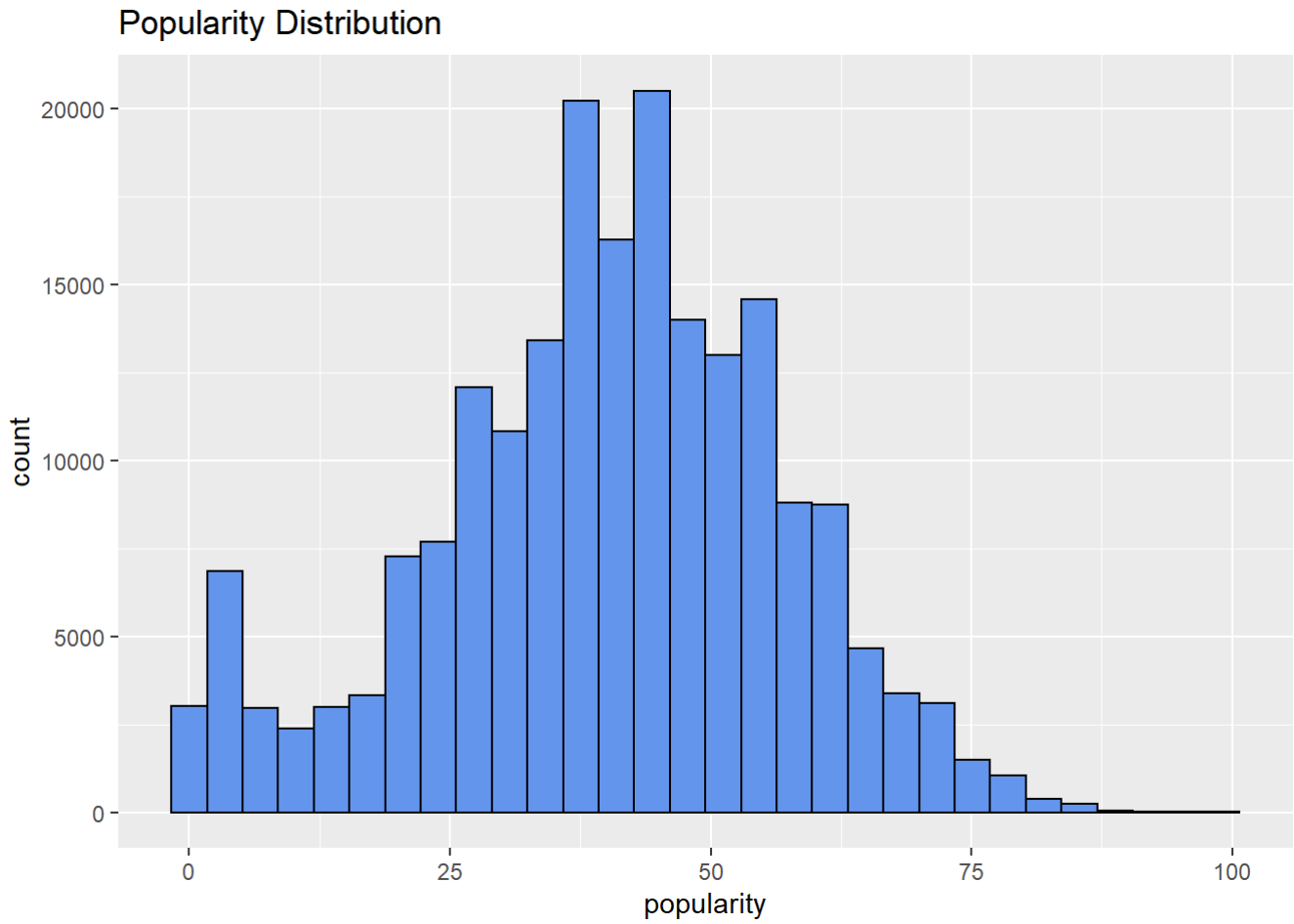
Checking whether the tracks.2k subset contains any missing values (i.e., NAs) to handle if necessary.

```
# FALSE means there are no NAs; TRUE means there are some indeed
any(is.na(tracks.2k))
```

```
## [1] FALSE
```

Looking at the new histogram of popularity after removing the songs with a popularity score of 0.

```
ggplot(data = tracks.2k, aes(x=popularity)) + geom_histogram(bins = 30, fill = "cornflowerblu
e", color = "black") + ggtitle("Popularity Distribution")
```




```
# checking the frequency of popularity in terms of counts
tracks.2k %>%
  group_by(popularity) %>%
  summarise(counts = n())
```

```
## # A tibble: 100 x 2
##    popularity counts
##         <dbl>  <int>
##  1          1   3029
##  2          2   2374
##  3          3   1853
##  4          4   1361
##  5          5   1280
##  6          6   1035
##  7          7   1008
##  8          8    914
##  9          9    832
## 10         10    790
## # ... with 90 more rows
```

Looking at the descriptive statistics of the final subset tracks.2k and the structure of each variable in case some variable type needs to be transformed.

```
# structure of tracks.2k
str(tracks.2k)
```

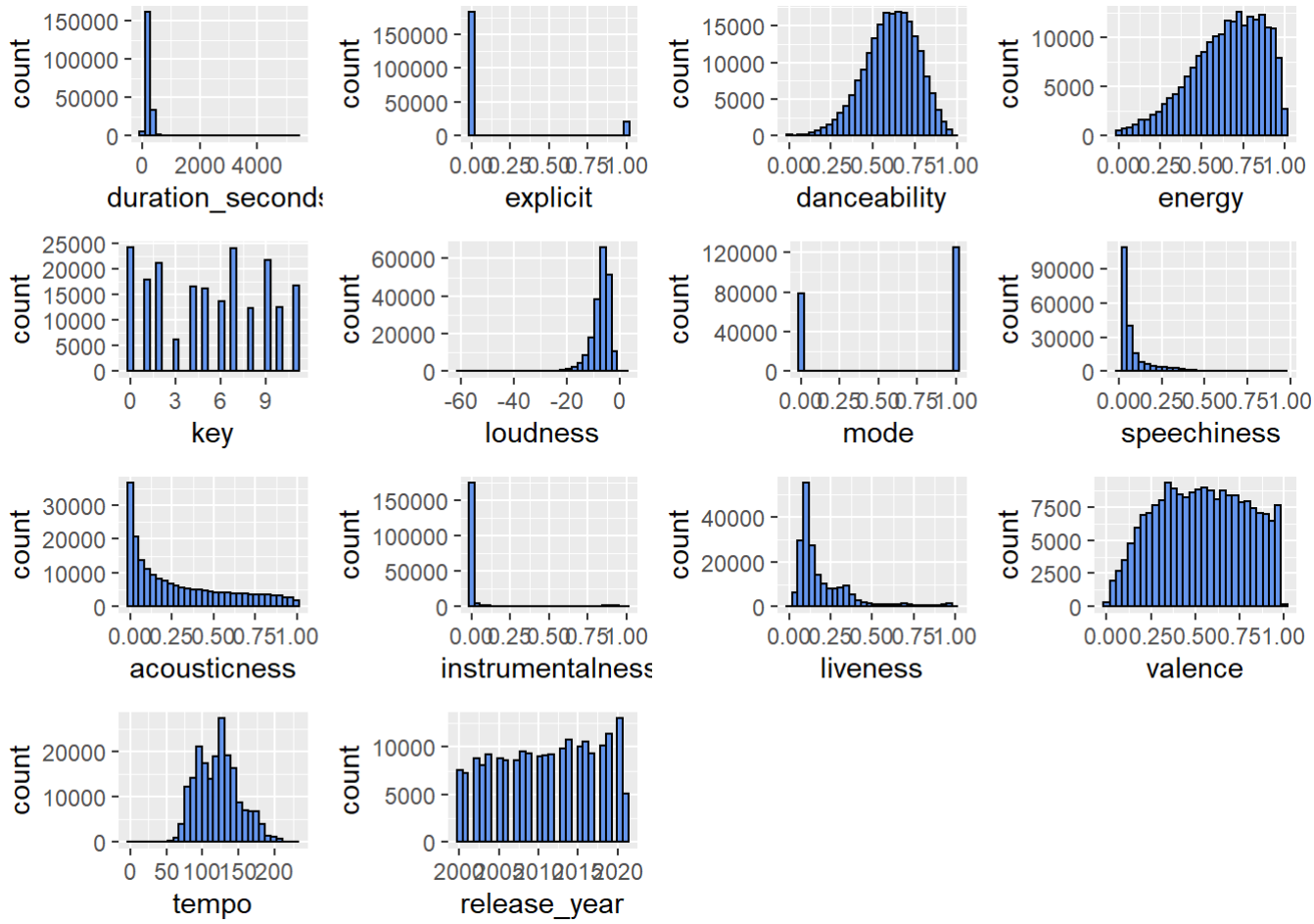
```
## 'data.frame':    203382 obs. of  18 variables:
##  $ id              : chr  "6catF1lDhNTjjGa2GxRQNN" "6Pkt6qVikqPBt9bEQy8iTz" "4aSw1QJIMwYSo
DEgzgdCJL" "0ZMMtH875IR2TfkyC4PolD" ...
##  $ name            : chr  "You'll Never Walk Alone - Mono; 2002 Remaster" "A Lover's Conce
rto" "Ferry Cross the Mersey - Mono; 2002 Remaster" "Don't Let the Sun Catch You Crying (Mai
n) - Mono" ...
##  $ popularity      : num  56 41 40 34 26 25 29 25 23 23 ...
##  $ duration_seconds: num  160 160 142 157 187 ...
##  $ explicit        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ artists         : chr  "['Gerry & The Pacemakers']" "['The Toys']" "['Gerry & The Pacem
akers']" "['Gerry & The Pacemakers']" ...
##  $ danceability    : num  0.484 0.671 0.405 0.477 0.319 0.269 0.617 0.409 0.459 0.399 ...
##  $ energy          : num  0.265 0.867 0.365 0.352 0.201 0.129 0.711 0.639 0.344 0.296 ...
##  $ key             : num  0 2 6 1 7 7 9 6 1 6 ...
##  $ loudness        : num  -11.1 -2.71 -10.23 -14.16 -17.8 ...
##  $ mode            : num  1 1 0 1 1 0 1 1 1 0 ...
##  $ speechiness     : num  0.0322 0.0571 0.0289 0.03 0.0623 0.0576 0.0297 0.0302 0.0301 0.0
294 ...
##  $ acousticness    : num  0.394 0.436 0.255 0.406 0.887 0.938 0.36 0.288 0.764 0.5 ...
##  $ instrumentalness: num  0.00 0.00 4.68e-06 0.00 0.00 4.87e-06 1.59e-06 0.00 5.94e-04 0.0
0 ...
##  $ liveness        : num  0.149 0.139 0.163 0.122 0.904 0.683 0.0841 0.343 0.111 0.0719
...
##  $ valence         : num  0.285 0.839 0.588 0.478 0.239 0.16 0.963 0.928 0.487 0.658 ...
##  $ tempo           : num  114 121 105 107 117 ...
##  $ release_year    : num  2008 2020 2008 2008 2018 ...
```

```
# all variables of interest are numeric, so no need to transform any of them

# stargazer format
stargazer(tracks.2k,
          type = "text", min.max = TRUE, mean.sd = TRUE,
          nobs = FALSE, median = FALSE, iqr = FALSE,
          digits = 3, align = TRUE,
          title = "Summary Statistics")
```

```
##
## Summary Statistics
## ===============================================================================
## Statistic          Mean    St. Dev.   Min   Pctl(25) Pctl(75)    Max
## -------------------------------------------------------------------------------
## popularity        40.058    16.795     1       30       51       100
## duration_seconds  231.680  100.712  23.534  191.187  261.693  5,403.500
## explicit           0.102    0.303      0       0        0         1
## danceability       0.607    0.158    0.000   0.506    0.722     0.991
## energy             0.653    0.217    0.000   0.511    0.828     1.000
## key                5.321    3.567      0       2        9        11
## loudness          -7.342    3.703   -60.000  -8.708   -5.025    2.534
## mode               0.613    0.487      0       0        1         1
## speechiness        0.089    0.109    0.000   0.034    0.090     0.966
## acousticness       0.298    0.292    0.000   0.039    0.510     0.996
## instrumentalness   0.067    0.211      0       0      0.001       1
## liveness           0.206    0.181    0.000   0.096    0.262     1.000
## valence            0.537    0.250    0.000   0.336    0.744     1.000
## tempo             121.760   29.119   0.000   98.024  139.920   229.862
## release_year     2,010.946  6.202    2,000   2,006    2,016     2,021
## -------------------------------------------------------------------------------
```

Looking at the other variables' histograms to get a better impression of their distribution.

```
# subsetting only the predictors from tracks.2k
df.predictors <- tracks.2k %>%
  select(duration_seconds, explicit, danceability,
         energy, key, loudness, mode, speechiness, acousticness,
         instrumentalness, liveness, valence, tempo, release_year)

# creating a list to contain all individual histogram objects
myplots <- list()

# creating a loop to create a histogram for each individual predictor
for (i in 1:ncol(df.predictors)) {
  col <- names(df.predictors)[i]
  ggp <- ggplot(df.predictors, aes_string(x = col)) +
    geom_histogram(bins = 30, fill = "cornflowerblue", color = "black")
  # + geom_vline(xintercept = mean(df.predictors[[col]]), col = "red", lwd=1.5)
  myplots[[i]] <- ggp
}

# plotting all predictor histograms together in one grid
plot_grid(plotlist = myplots, label_size = 10)
```




```
# removing the objects no longer required
df.predictors <- NULL
myplots <- NULL
ggp <- NULL
```

Splitting the data

Splitting the data into train and test set after setting a random seed to allow for a reproducibility of the exact same analysis. NOTE: the argument “strata” allows for a popularity distribution across the train-test split that is close to that of the original/entire dataset.

The distribution of individual popularity scores within the train- and test-set can be found below. As can be drawn from the dataframes obtained, the distribution of popularity seems to be approximately equal across the train-test split.

```
### The code provides an overview about whether the split is accurately stratified
tracks.2k.train %>% count(popularity) %>%
  mutate(prop = n / sum(n))
```

```
##    popularity    n         prop
## 1           1 2126 1.493334e-02
## 2           2 1662 1.167414e-02
## 3           3 1312 9.215684e-03
## 4           4  974 6.841521e-03
## 5           5  882 6.195299e-03
## 6           6  713 5.008218e-03
## 7           7  684 4.804518e-03
## 8           8  628 4.411166e-03
## 9           9  560 3.933523e-03
## 10         10  547 3.842210e-03
## 11         11  524 3.680654e-03
## 12         12  485 3.406712e-03
## 13         13  480 3.371592e-03
## 14         14  524 3.680654e-03
## 15         15  626 4.397117e-03
## 16         16  700 4.916904e-03
## 17         17  729 5.120605e-03
## 18         18  855 6.005647e-03
## 19         19 1188 8.344689e-03
## 20         20 1145 8.042651e-03
## 21         21 1320 9.271877e-03
## 22         22 1513 1.062754e-02
## 23         23 1807 1.269264e-02
## 24         24 1787 1.255215e-02
## 25         25 1801 1.265049e-02
## 26         26 1984 1.393591e-02
## 27         27 2064 1.449784e-02
## 28         28 2132 1.497549e-02
## 29         29 2291 1.609233e-02
## 30         30 2317 1.627495e-02
## 31         31 2462 1.729345e-02
## 32         32 2775 1.949201e-02
## 33         33 2962 2.080553e-02
## 34         34 3128 2.197154e-02
## 35         35 3323 2.334125e-02
## 36         36 3390 2.381187e-02
## 37         37 3527 2.477417e-02
## 38         38 3561 2.501299e-02
## 39         39 3659 2.570136e-02
## 40         40 3897 2.737311e-02
## 41         41 3747 2.631949e-02
## 42         42 3784 2.657938e-02
## 43         43 3744 2.629841e-02
## 44         44 3625 2.546254e-02
## 45         45 3685 2.588399e-02
## 46         46 3309 2.324291e-02
## 47         47 3363 2.362221e-02
## 48         48 3258 2.288468e-02
## 49         49 3191 2.241406e-02
## 50         50 3143 2.207690e-02
## 51         51 3039 2.134639e-02
## 52         52 2833 1.989941e-02
## 53         53 2743 1.926724e-02
## 54         54 2615 1.836815e-02
## 55         55 2484 1.744799e-02
## 56         56 2358 1.656294e-02
```

```
## 57         57 2169 1.523538e-02
## 58         58 2056 1.444165e-02
## 59         59 1927 1.353554e-02
## 60         60 1725 1.211666e-02
## 61         61 1581 1.110518e-02
## 62         62 1520 1.067671e-02
## 63         63 1322 9.285925e-03
## 64         64 1186 8.330641e-03
## 65         65 1121 7.874071e-03
## 66         66  996 6.996052e-03
## 67         67  875 6.146130e-03
## 68         68  764 5.366450e-03
## 69         69  764 5.366450e-03
## 70         70  640 4.495455e-03
## 71         71  570 4.003765e-03
## 72         72  506 3.554219e-03
## 73         73  426 2.992287e-03
## 74         74  374 2.627032e-03
## 75         75  353 2.479525e-03
## 76         76  297 2.086172e-03
## 77         77  245 1.720917e-03
## 78         78  187 1.313516e-03
## 79         79  170 1.194105e-03
## 80         80  137 9.623084e-04
## 81         81  105 7.375356e-04
## 82         82   90 6.321734e-04
## 83         83   73 5.127629e-04
## 84         84   55 3.863282e-04
## 85         85   36 2.528694e-04
## 86         86   38 2.669177e-04
## 87         87   29 2.037003e-04
## 88         88   17 1.194105e-04
## 89         89   10 7.024149e-05
## 90         90   12 8.428979e-05
## 91         91    9 6.321734e-05
## 92         92    9 6.321734e-05
## 93         94    3 2.107245e-05
## 94         96    1 7.024149e-06
## 95         98    1 7.024149e-06
## 96         99    1 7.024149e-06
## 97        100    1 7.024149e-06
```

```
tracks.2k.test %>% count(popularity) %>%
  mutate(prop = n / sum(n))
```

```
##    popularity    n         prop
## 1           1  903 1.479940e-02
## 2           2  712 1.166907e-02
## 3           3  541 8.866527e-03
## 4           4  387 6.342599e-03
## 5           5  398 6.522879e-03
## 6           6  322 5.277304e-03
## 7           7  324 5.310083e-03
## 8           8  286 4.687295e-03
## 9           9  272 4.457847e-03
## 10         10  243 3.982562e-03
## 11         11  227 3.720336e-03
## 12         12  192 3.146716e-03
## 13         13  217 3.556444e-03
## 14         14  223 3.654779e-03
## 15         15  248 4.064508e-03
## 16         16  309 5.064245e-03
## 17         17  344 5.637865e-03
## 18         18  384 6.293431e-03
## 19         19  460 7.539006e-03
## 20         20  498 8.161794e-03
## 21         21  514 8.424020e-03
## 22         22  647 1.060378e-02
## 23         23  764 1.252131e-02
## 24         24  740 1.212797e-02
## 25         25  800 1.311132e-02
## 26         26  842 1.379966e-02
## 27         27  875 1.434050e-02
## 28         28  944 1.547135e-02
## 29         29  948 1.553691e-02
## 30         30 1019 1.670054e-02
## 31         31 1095 1.794611e-02
## 32         32 1167 1.912613e-02
## 33         33 1277 2.092894e-02
## 34         34 1329 2.178117e-02
## 35         35 1393 2.283008e-02
## 36         36 1453 2.381343e-02
## 37         37 1515 2.482955e-02
## 38         38 1499 2.456733e-02
## 39         39 1615 2.646847e-02
## 40         40 1615 2.646847e-02
## 41         41 1656 2.714042e-02
## 42         42 1584 2.596040e-02
## 43         43 1575 2.581290e-02
## 44         44 1597 2.617346e-02
## 45         45 1509 2.473122e-02
## 46         46 1477 2.420677e-02
## 47         47 1436 2.353481e-02
## 48         48 1391 2.279730e-02
## 49         49 1374 2.251868e-02
## 50         50 1330 2.179756e-02
## 51         51 1360 2.228924e-02
## 52         52 1287 2.109283e-02
## 53         53 1210 1.983086e-02
## 54         54 1112 1.822473e-02
## 55         55 1061 1.738888e-02
## 56         56  997 1.633998e-02
```

```
## 57         57  914 1.497968e-02
## 58         58  866 1.419300e-02
## 59         59  884 1.448800e-02
## 60         60  708 1.160351e-02
## 61         61  685 1.122656e-02
## 62         62  599 9.817097e-03
## 63         63  597 9.784319e-03
## 64         64  498 8.161794e-03
## 65         65  449 7.358726e-03
## 66         66  419 6.867051e-03
## 67         67  360 5.900092e-03
## 68         68  340 5.572309e-03
## 69         69  274 4.490625e-03
## 70         70  279 4.572571e-03
## 71         71  258 4.228399e-03
## 72         72  235 3.851449e-03
## 73         73  193 3.163105e-03
## 74         74  195 3.195883e-03
## 75         75  134 2.196145e-03
## 76         76  122 1.999476e-03
## 77         77   96 1.573358e-03
## 78         78   89 1.458634e-03
## 79         79   67 1.098073e-03
## 80         80   58 9.505703e-04
## 81         81   40 6.555658e-04
## 82         82   37 6.063983e-04
## 83         83   29 4.752852e-04
## 84         84   31 5.080635e-04
## 85         85   21 3.441720e-04
## 86         86   13 2.130589e-04
## 87         87    9 1.475023e-04
## 88         88    2 3.277829e-05
## 89         89    6 9.833486e-05
## 90         91    2 3.277829e-05
## 91         92    1 1.638914e-05
## 92         93    2 3.277829e-05
## 93         94    3 4.916743e-05
## 94         95    1 1.638914e-05
## 95         96    1 1.638914e-05
## 96         97    2 3.277829e-05
```

R A N D O M F O R E S T - M O D E L I N G

The model is tuned on the training data using 10-fold stratified cross validation. Therefore, the folds are created in the following chunk.

```
set.seed(1995)
cv.folds.no.year <- tracks.2k.train %>% vfold_cv(v = 10, strata = popularity)
```

Having created the necessary train-test split and CV-folds based on popularity, a recipe including all 13 predictors of interest in the train set model is created for the random forest.

```
rf.recipe.no.year <- recipe(popularity ~ duration_seconds + explicit + danceability + energy
                    + key + loudness + mode + speechiness + acousticness + instrumentalness
                    + liveness + valence + tempo, data = tracks.2k.train)
rf.recipe.no.year
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor         13
```

Below, the RF model is specified to use 500 trees, a number of trees that can be reasonably assumed to be sufficient for random forests. Additionally, the argument “mtry” specifies the number of predictors the model uses for each split. Since the argument is set to tune(), the algorithm autonomously tests each potential number of variables available for each split.

Having defined the model, the workflow can be created, which combines both the recipe and the model. This workflow can also be tuned later on.

```
rf.tune.wf.no.year <- workflow() %>%
  add_recipe(rf.recipe.no.year) %>%
  add_model(rf.model.tune.no.year)

rf.tune.wf.no.year
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 0 Recipe Steps
##
## -- Model -----------------------------------------------------------------------
## Random Forest Model Specification (regression)
##
## Main Arguments:
##   mtry = tune()
##   trees = 500
##
## Computational engine: ranger
```

To enable an assessment of the predictive performance of the random forest model, the error metrics “Root-mean-square error”, “R squared” and “Mean absolute error” are used. These metrics are relatively conventional for a regression.

To ensure a smoother and quicker running of the subsequent tasks, the use of multiple CPU cores is initiated using the “doParallel”-function. That way, each CPU can create individual decision trees in parallel. NOTE: since the computer this analysis is conducted on is not able to cool down all cores when using them and instead crashes mid-analysis, the function was not used. As a result, the subsequent model tuning took around 20h. To use the function, simply remove the hashtag (#) and run the chunk.

```
# registerDoParallel()
```

In the following sequence, the model is tuned using the “tune_grid”-function and 10-fold cross-validation, based on the aforementioned performance/error metrics. The tune grid is set from 1 to 13, as the model features 13 independent variables. NOTE: to knit the Rmd-file to HTML without having to rerun the model

tuning again, the previously saved data-object containing the results is simply loaded into the environment. The same goes for the later RF model.

Saving the object “rf.tune.res.no.year” so it can be simply loaded (making it quicker) in the future.

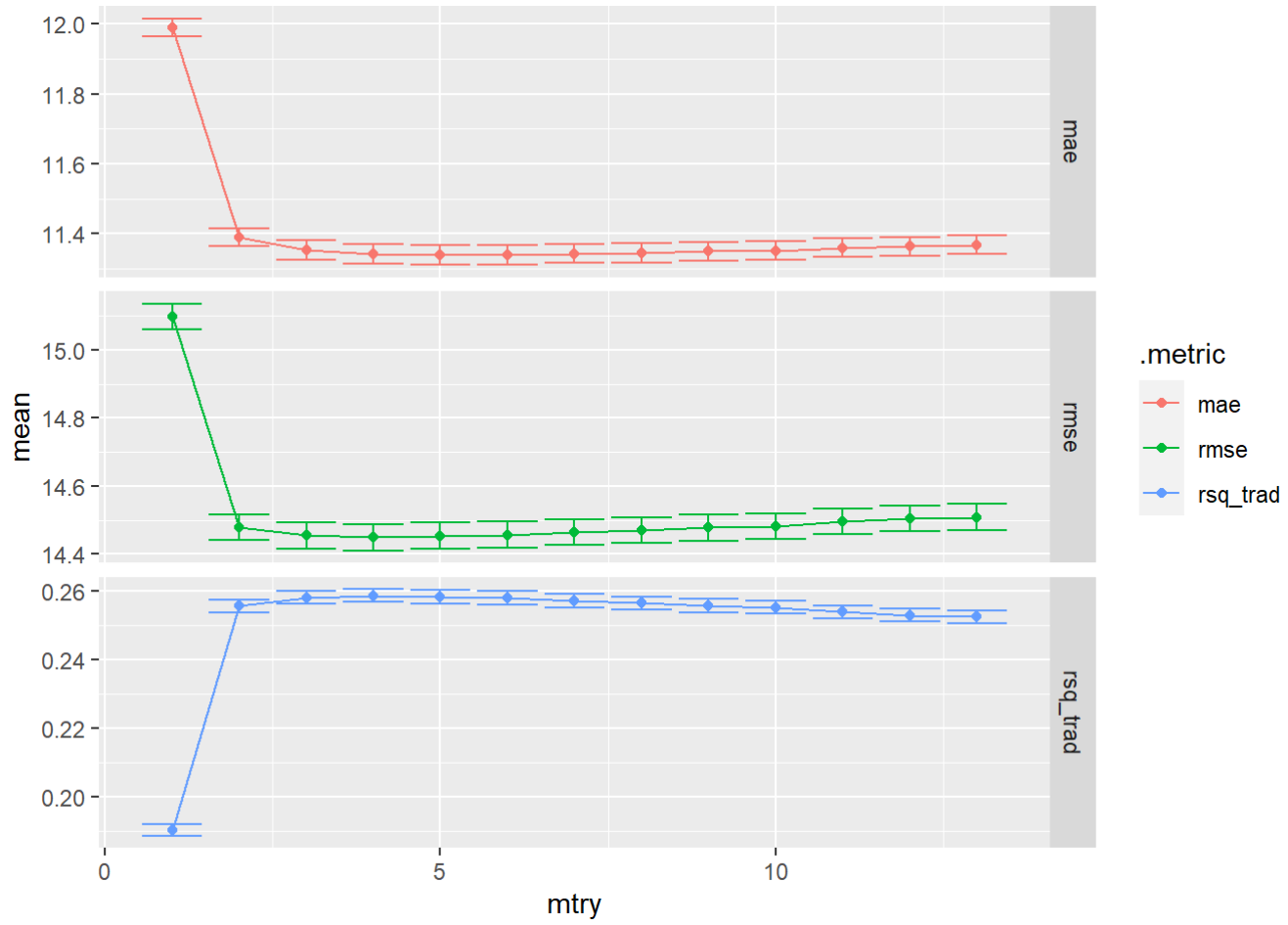
Having tuned the model, the individual metrics calculated using the 10-fold cross-validation are shown below. Thus, each of the 10 models created has different values for those metrics.

```
error.metrics.no.year <- rf.tune.res.no.year %>%
  collect_metrics()

error.metrics.no.year
```

```
## # A tibble: 39 x 7
##     mtry .metric  .estimator   mean     n std_err .config
##    <int> <chr>    <chr>       <dbl> <int>   <dbl> <fct>
##  1     1 mae      standard   12.0      10 0.0259  Preprocessor1_Model01
##  2     1 rmse     standard   15.1      10 0.0374  Preprocessor1_Model01
##  3     1 rsq_trad standard    0.191    10 0.00175 Preprocessor1_Model01
##  4     2 mae      standard   11.4      10 0.0258  Preprocessor1_Model02
##  5     2 rmse     standard   14.5      10 0.0377  Preprocessor1_Model02
##  6     2 rsq_trad standard    0.256    10 0.00180 Preprocessor1_Model02
##  7     3 mae      standard   11.4      10 0.0281  Preprocessor1_Model03
##  8     3 rmse     standard   14.5      10 0.0387  Preprocessor1_Model03
##  9     3 rsq_trad standard    0.258    10 0.00189 Preprocessor1_Model03
## 10     4 mae      standard   11.3      10 0.0276  Preprocessor1_Model04
## # ... with 29 more rows
```

In the next chunk of code, the increase or decrease of each error metric for a given number of variables used for splitting (i.e., the mtry-value) is plotted.The plot represents all mtry-values from 1 to 13, since the latter is the maximum number of variables the model can potentially consider at each split.

```
rf.tune.res.no.year %>%
  collect_metrics() %>%
  filter(.metric %in% c("mae", "rmse", "rsq_trad")) %>%
  ggplot(aes(x = mtry, y = mean, ymin = mean - std_err, ymax = mean + std_err,
             colour = .metric)) +
  geom_errorbar() +
  geom_line() +
  geom_point() +
  facet_grid(.metric ~ ., scales = "free_y")
```



The plot above provides a visual representation of the change in value of all 3 error metrics based on differing numbers of variables at disposal for each split (i.e., the mtry-value). Following this, the optimal number of variables is computationally chosen using the optimal R-squared value in the following chunk of code (it does not really matter which metric is chosen since the plot seems to show rather “similar” curves for each one of them) and subsequently spliced into the final workflow.

```
best.rsq_trad.no.year <- select_best(rf.tune.res.no.year, "rsq_trad")
rf.final.wf.no.year <- finalize_workflow(rf.tune.wf.no.year, best.rsq_trad.no.year)
rf.final.wf.no.year
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 0 Recipe Steps
##
## -- Model -----------------------------------------------------------------------
## Random Forest Model Specification (regression)
##
## Main Arguments:
##   mtry = 4
##   trees = 500
##
## Computational engine: ranger
```

Having updated the random forest model in the previous chunk using the best possible R-squared, one can see that the optimal number of variables to use at each split is 4. Following this, the model is trained again on the train-set and ultimately used and tested for predicting the test-set values.

```
set.seed(0905)
final.res.no.year <- rf.final.wf.no.year %>%
  last_fit(tracks.2k.split, metrics = regression.metrics)
```

```
## Warning: package 'rlang' was built under R version 4.0.5
```

```
## Warning: package 'vctrs' was built under R version 4.0.5
```

The next chunk shows the value of each error metric for the predictions made on the test set.

```
rf.test.performance.no.year <- final.res.no.year %>%
  collect_metrics()

rf.test.performance.no.year
```

```
## # A tibble: 3 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 rmse     standard      14.4   Preprocessor1_Model1
## 2 rsq_trad standard       0.261 Preprocessor1_Model1
## 3 mae      standard      11.3   Preprocessor1_Model1
```

Extracting the predictions made by the RF in order to calculate the residuals.

```
# collecting the predictions made by the RF on the test-set
rf.predictions.no.year <- final.res.no.year %>% collect_predictions()

# calculating the residuals by subtracting the predictions made from the actual popularity sc
ore
rf.predictions.no.year$residuals <- rf.predictions.no.year$popularity - rf.predictions.no.yea
r$.pred
```

In the next chunk, the residuals are plotted in two different styles to help visualize them and make inferences about the generalizability of the model.

```
# residuals for each observation
plot(rf.predictions.no.year$residuals, type = "prediction", abline = TRUE,
     xlab = "Observations", ylab = "Residuals")
```

```
## Warning in plot.window(...): "abline" is not a graphical parameter
```

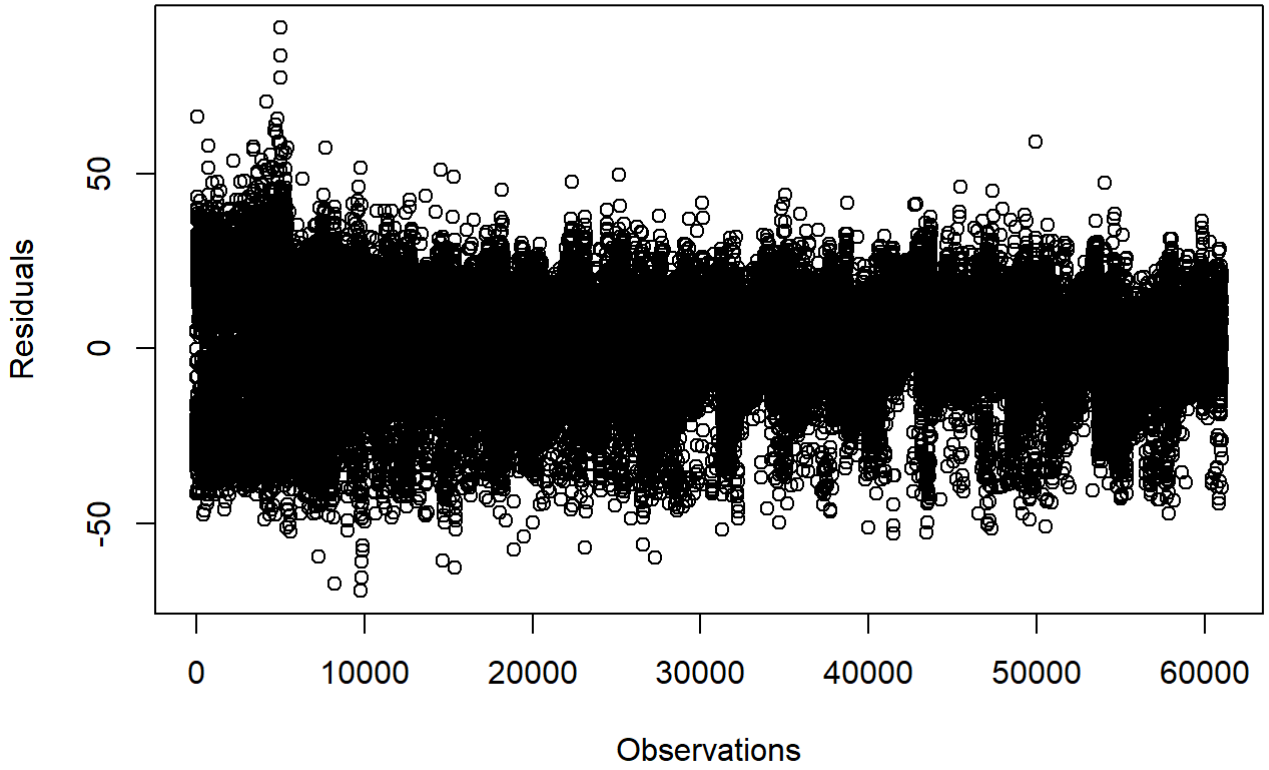
```
## Warning in plot.xy(xy, type, ...): plot type 'prediction' will be truncated to
## first character
```

```
## Warning in plot.xy(xy, type, ...): "abline" is not a graphical parameter
```

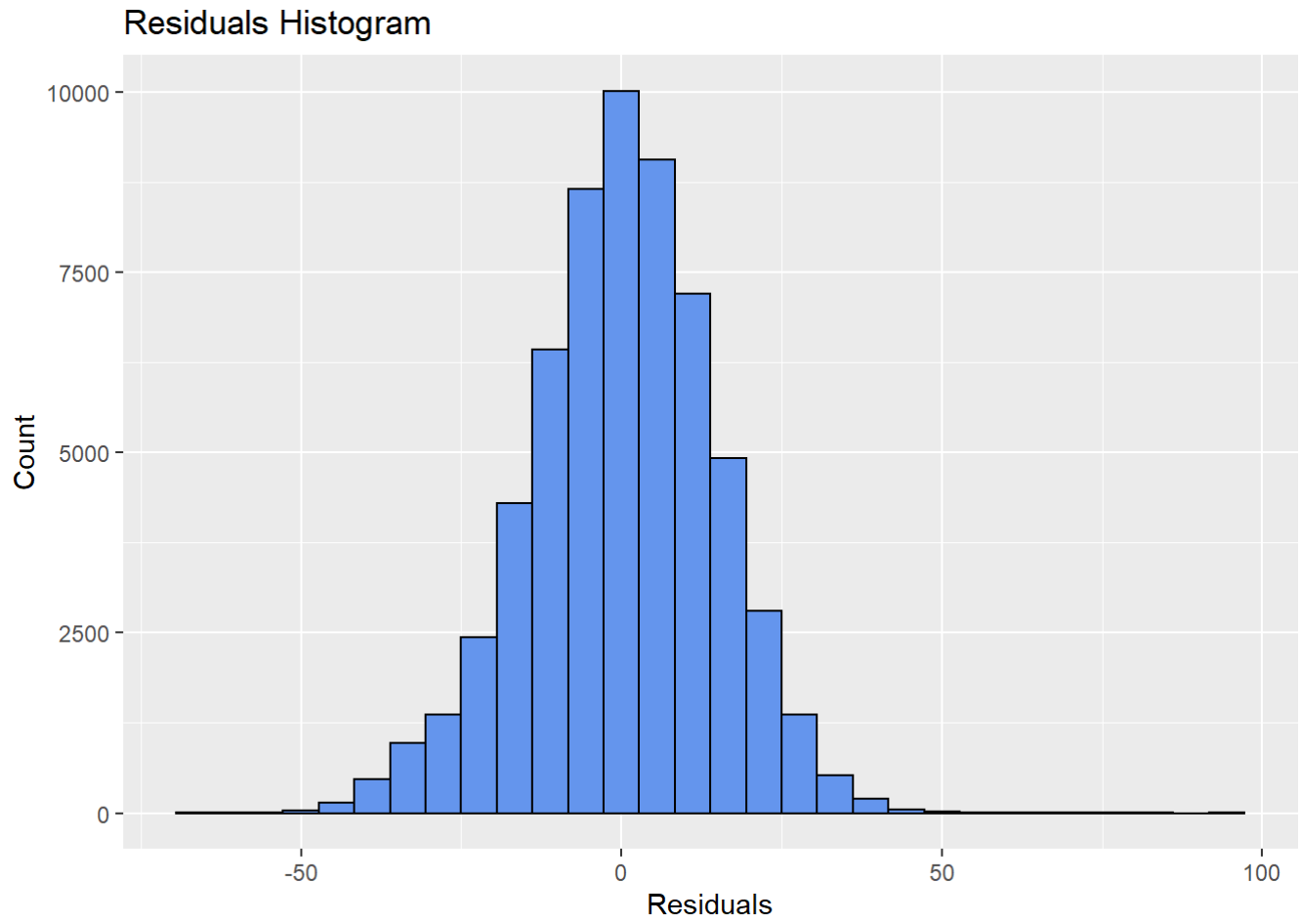
```
## Warning in axis(side = side, at = at, labels = labels, ...): "abline" is not a
## graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "abline" is not a
## graphical parameter
```

```
## Warning in box(...): "abline" is not a graphical parameter
```

```
## Warning in title(...): "abline" is not a graphical parameter
```




```
# histogram of the residuals
ggplot(data = rf.predictions.no.year, aes(x=residuals)) + geom_histogram(bins = 30, fill = "c
ornflowerblue", color = "black") + ggtitle("Residuals Histogram") + xlab("Residuals") + ylab(
"Count")
```



V A R I A B L E I M P O R T A N C E

In order to obtain measures concerning the variable importance of each predictors, the random forest model needs to be run again with the additional argument “importance”. Since the previously computed optimal value of mtry equals 4, this information is manually added to the code at this point.

```
rf.model.var.importance.no.year <- rand_forest(mtry = 4, trees = 500) %>%
  set_mode("regression") %>%
  set_engine("ranger", importance = "permutation")
```

To run this new model containing variable importance information, a new workflow is created.

```
rf.var.importance.wf.no.year <- workflow() %>%
  add_model(rf.model.var.importance.no.year) %>%
  add_recipe(rf.recipe.no.year)
```

Having established the new workflow, the new model is fitted on the train set. To do this, the same seed as previously used to assess the model’s test set performance needs to be specified.

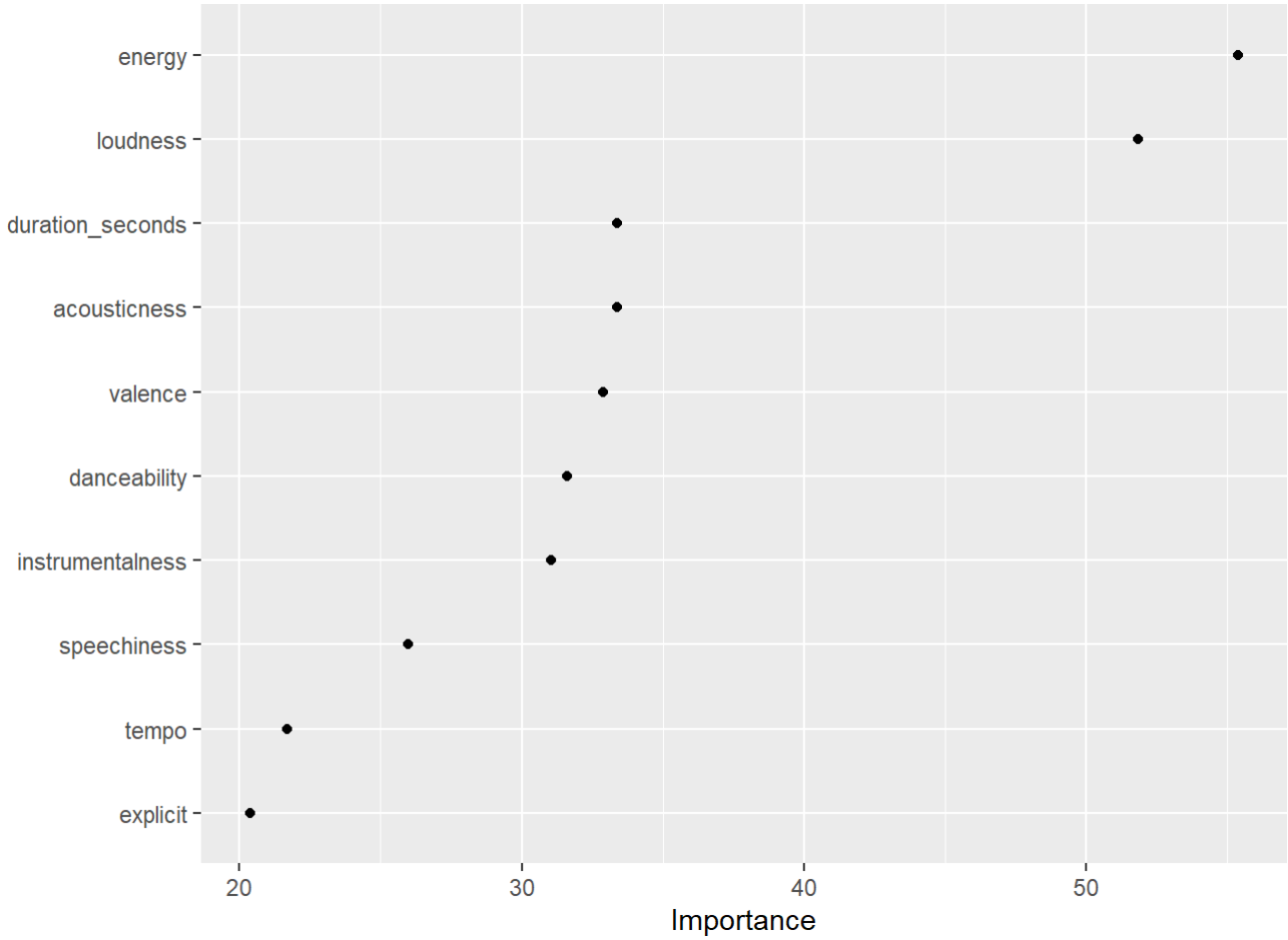
```
set.seed(0905)
rf.var.importance.fit.no.year <- rf.var.importance.wf.no.year %>% fit(data = tracks.2k.train)
```

Ultimately, the measures of variable importance can be excerpted and visualized.

```
# excerpting the variable importance measures
rf.var.importance.fit.no.year %>% pull_workflow_fit() %>% vi()
```

```
## # A tibble: 13 x 2
##    Variable         Importance
##    <chr>                 <dbl>
##  1 energy                55.4
##  2 loudness              51.9
##  3 duration_seconds      33.4
##  4 acousticness          33.4
##  5 valence               32.9
##  6 danceability          31.6
##  7 instrumentalness      31.0
##  8 speechiness           26.0
##  9 tempo                 21.7
## 10 explicit              20.4
## 11 liveness              14.6
## 12 key                    3.39
## 13 mode                   1.81
```

```
# plotting the variable importance
rf.var.importance.fit.no.year %>% pull_workflow_fit() %>% vip(geom = "point", num_features = 10)
```



MULTIPLE LINEAR REGRESSION

In the next chunk, a multiple linear regression is created on the same train-set as the RF model in order to obtain a (simpler) baseline to compare the performance of the more complex RF model against.

```
lm.popularity <- lm(popularity ~ duration_seconds + explicit + danceability + energy
                    + key + loudness + mode + speechiness + acousticness + instrumentalness
                    + liveness + valence + tempo, data = tracks.2k.train)

stargazer(lm.popularity, type = "text")
```

```
##
## ================================================
##                         Dependent variable:
##                     ---------------------------
##                               popularity
## ------------------------------------------------
## duration_seconds              -0.003***
##                                (0.0004)
##
## explicit                       8.296***
##                                (0.150)
##
## danceability                   9.367***
##                                (0.332)
##
## energy                        -12.617***
##                                (0.367)
##
## key                             0.002
##                                (0.012)
##
## loudness                       0.718***
##                                (0.018)
##
## mode                           -0.015
##                                (0.088)
##
## speechiness                   -2.890***
##                                (0.422)
##
## acousticness                    -0.062
##                                (0.200)
##
## instrumentalness              -12.694***
##                                (0.226)
##
## liveness                      -5.295***
##                                (0.244)
##
## valence                       -4.012***
##                                (0.214)
##
## tempo                          0.004***
##                                (0.001)
##
## Constant                      51.614***
##                                (0.480)
##
## ------------------------------------------------
## Observations                   142,366
## R2                              0.104
## Adjusted R2                     0.104
## Residual Std. Error      15.900 (df = 142352)
## F Statistic         1,271.889*** (df = 13; 142352)
## ================================================
## Note:                  *p<0.1; **p<0.05; ***p<0.01
```

Based on the created linear regression, the popularity of the test-set observations can be predicted using the function predict().

In the following chunk, the error metrics of the linear model are shown.

```
# rmse
RMSE(lm.predictions, tracks.2k.test$popularity)
```

```
## [1] 15.92166
```

```
# R-squared
R2(lm.predictions, tracks.2k.test$popularity)
```

```
## [1] 0.1008755
```

```
# mae
MAE(lm.predictions, tracks.2k.test$popularity)
```

```
## [1] 12.63849
```

R F - C O M P A R I T I V E P E R F O R M A N C E

As a last analysis, the performance of the previously created RF is compared against one created on a sample solely containing tracks from 2021 (as these are the most up-to-date ones). This is done to assess whether the predictive performance holds when tested on a much narrower, time-based subset.

```
tracks.2021 <- subset(tracks.2k, release_year == 2021)
```

Looking at the number of unique songs and artists in tracks.2021.

```
length(unique(tracks.2021$id))
```

```
## [1] 5142
```

```
length(unique(tracks.2021$artists))
```

```
## [1] 2798
```

In the next chunks, the same procedure for data splitting and RF modeling as applied for the previous model is undertaken.

```
set.seed(2000)
cv.folds.2021 <- tracks.2021.train %>% vfold_cv(v = 10, strata = popularity)
```

Having created the necessary train-test split and CV-folds based on popularity, a recipe including all 13 predictors of interest in the train set model is created for the random forest.

```
rf.recipe.2021 <- recipe(popularity ~ duration_seconds + explicit + danceability + energy
                    + key + loudness + mode + speechiness + acousticness + instrumentalness
                    + liveness + valence + tempo, data = tracks.2021.train)
rf.recipe.2021
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor         13
```

```
rf.tune.wf.2021 <- workflow() %>%
  add_recipe(rf.recipe.2021) %>%
  add_model(rf.model.tune.2021)

rf.tune.wf.2021
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 0 Recipe Steps
##
## -- Model -----------------------------------------------------------------------
## Random Forest Model Specification (regression)
##
## Main Arguments:
##   mtry = tune()
##   trees = 500
##
## Computational engine: ranger
```

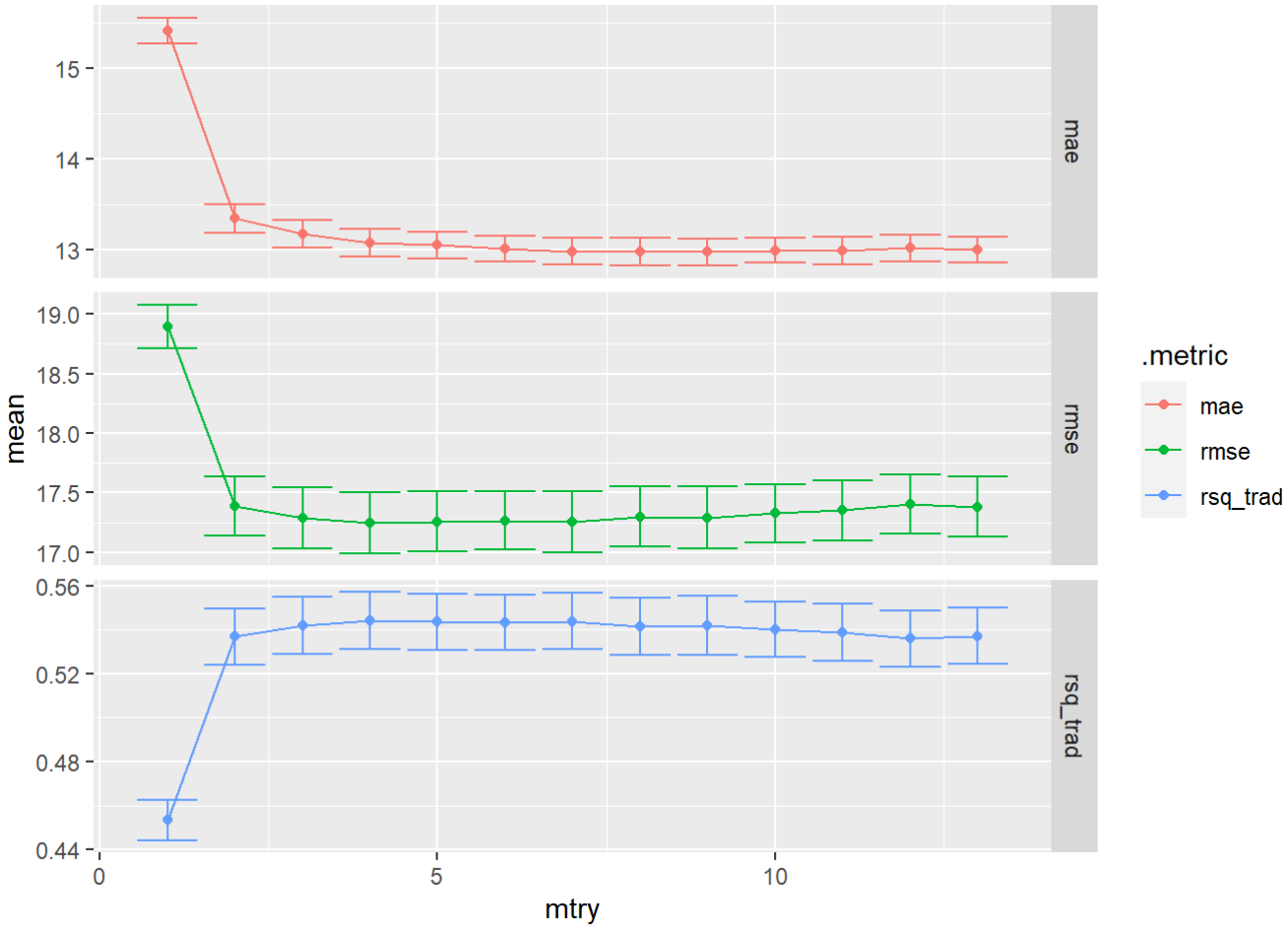
```
registerDoParallel()
```

Having tuned the model, the individual metrics calculated using the 10-fold cross-validation are shown below. Thus, each of the 10 models created has different values for those metrics.

```
error.metrics.2021 <- rf.tune.res.2021 %>%
  collect_metrics()

error.metrics.2021
```

```
## # A tibble: 39 x 7
##     mtry .metric  .estimator   mean     n std_err .config
##    <int> <chr>    <chr>       <dbl> <int>   <dbl> <fct>
##  1     1 mae      standard   15.4      10 0.144   Preprocessor1_Model01
##  2     1 rmse     standard   18.9      10 0.182   Preprocessor1_Model01
##  3     1 rsq_trad standard    0.453    10 0.00924 Preprocessor1_Model01
##  4     2 mae      standard   13.3      10 0.158   Preprocessor1_Model02
##  5     2 rmse     standard   17.4      10 0.252   Preprocessor1_Model02
##  6     2 rsq_trad standard    0.537    10 0.0128  Preprocessor1_Model02
##  7     3 mae      standard   13.2      10 0.150   Preprocessor1_Model03
##  8     3 rmse     standard   17.3      10 0.257   Preprocessor1_Model03
##  9     3 rsq_trad standard    0.542    10 0.0131  Preprocessor1_Model03
## 10     4 mae      standard   13.1      10 0.151   Preprocessor1_Model04
## # ... with 29 more rows
```

```
rf.tune.res.2021 %>%
  collect_metrics() %>%
  filter(.metric %in% c("mae", "rmse", "rsq_trad")) %>%
  ggplot(aes(x = mtry, y = mean, ymin = mean - std_err, ymax = mean + std_err,
             colour = .metric)) +
  geom_errorbar() +
  geom_line() +
  geom_point() +
  facet_grid(.metric ~ ., scales = "free_y")
```




```
best.rsq_trad.2021 <- select_best(rf.tune.res.2021, "rsq_trad")
rf.final.wf.2021 <- finalize_workflow(rf.tune.wf.2021, best.rsq_trad.2021)
rf.final.wf.2021
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 0 Recipe Steps
##
## -- Model -----------------------------------------------------------------------
## Random Forest Model Specification (regression)
##
## Main Arguments:
##   mtry = 4
##   trees = 500
##
## Computational engine: ranger
```

Once again, the random forest model having been updated in the previous chunk using the best possible R-squared, one can see that the optimal number of variables to use at each split is 4.

```
set.seed(0461)
final.res.2021 <- rf.final.wf.2021 %>%
  last_fit(tracks.2021.split, metrics = regression.metrics)
```

```
rf.test.performance.2021 <- final.res.2021 %>%
  collect_metrics()

rf.test.performance.2021
```

```
## # A tibble: 3 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 rmse     standard      17.5   Preprocessor1_Model1
## 2 rsq_trad standard       0.535 Preprocessor1_Model1
## 3 mae      standard      13.0   Preprocessor1_Model1
```

V A R I A B L E I M P O R T A N C E

```
rf.model.var.importance.2021 <- rand_forest(mtry = 4, trees = 500) %>%
  set_mode("regression") %>%
  set_engine("ranger", importance = "permutation")
```

```
rf.var.importance.wf.2021 <- workflow() %>%
  add_model(rf.model.var.importance.2021) %>%
  add_recipe(rf.recipe.2021)
```

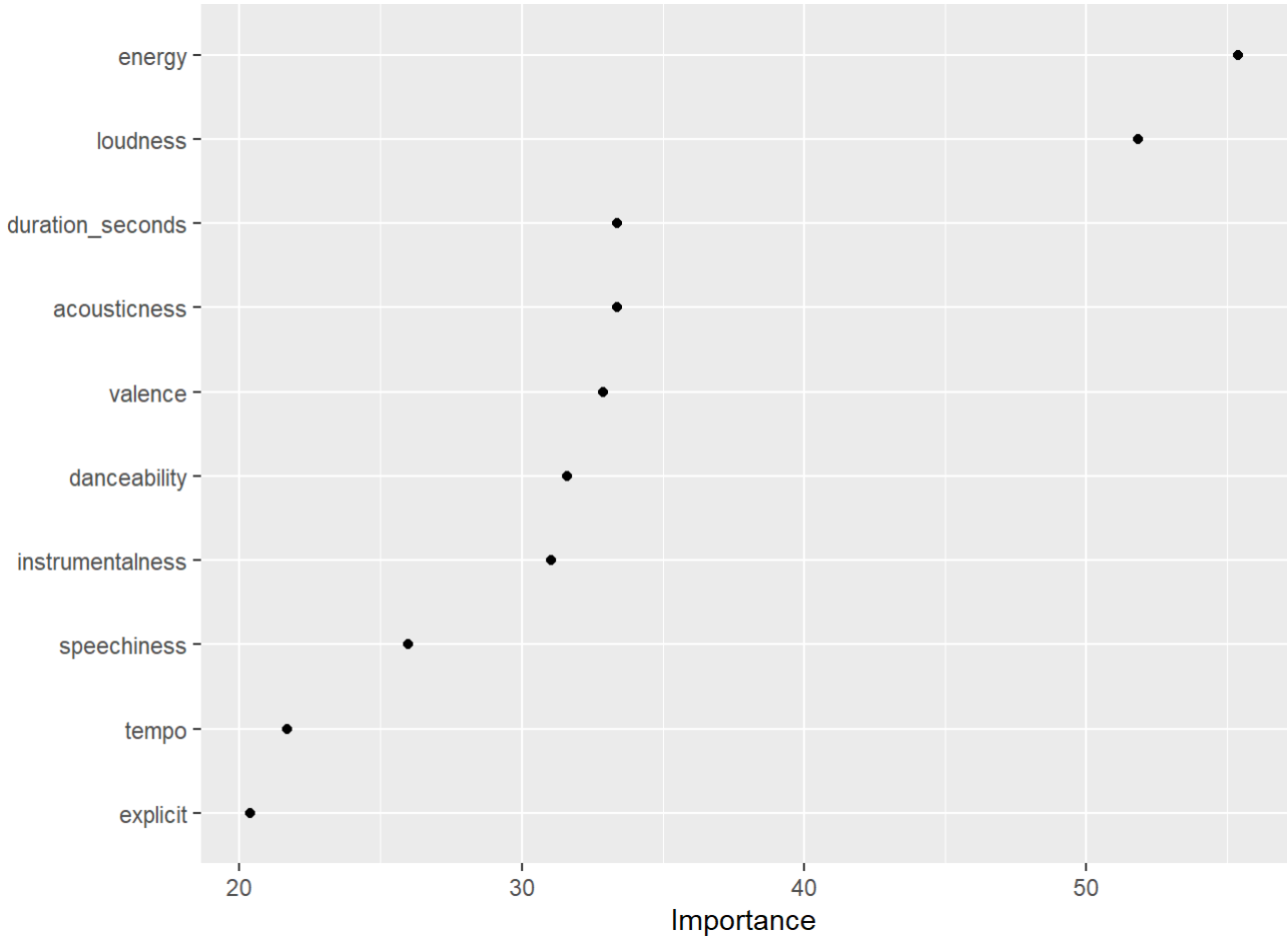
```
set.seed(0461)
rf.var.importance.fit.2021 <- rf.var.importance.wf.2021 %>% fit(data = tracks.2021.train)
```

Ultimately, the measures of variable importance can be excerpted and visualized.

```
# excerpting the variable importance measures
rf.var.importance.fit.no.year %>% pull_workflow_fit() %>% vi()
```

```
## # A tibble: 13 x 2
##    Variable         Importance
##    <chr>                 <dbl>
##  1 energy                55.4
##  2 loudness              51.9
##  3 duration_seconds      33.4
##  4 acousticness          33.4
##  5 valence               32.9
##  6 danceability          31.6
##  7 instrumentalness      31.0
##  8 speechiness           26.0
##  9 tempo                 21.7
## 10 explicit              20.4
## 11 liveness              14.6
## 12 key                    3.39
## 13 mode                   1.81
```

```
# plotting the variable importance
rf.var.importance.fit.no.year %>% pull_workflow_fit() %>% vip(geom = "point", num_features =
10)
```




```
rf.predictions.2021 <- final.res.2021 %>% collect_predictions()

rf.predictions.2021$residuals <- rf.predictions.2021$popularity - rf.predictions.2021$.pred

# residuals for each observation
plot(rf.predictions.2021$residuals, type = "prediction", abline = TRUE,
     xlab = "Observations", ylab = "Residuals")
```

```
## Warning in plot.window(...): "abline" is not a graphical parameter
```

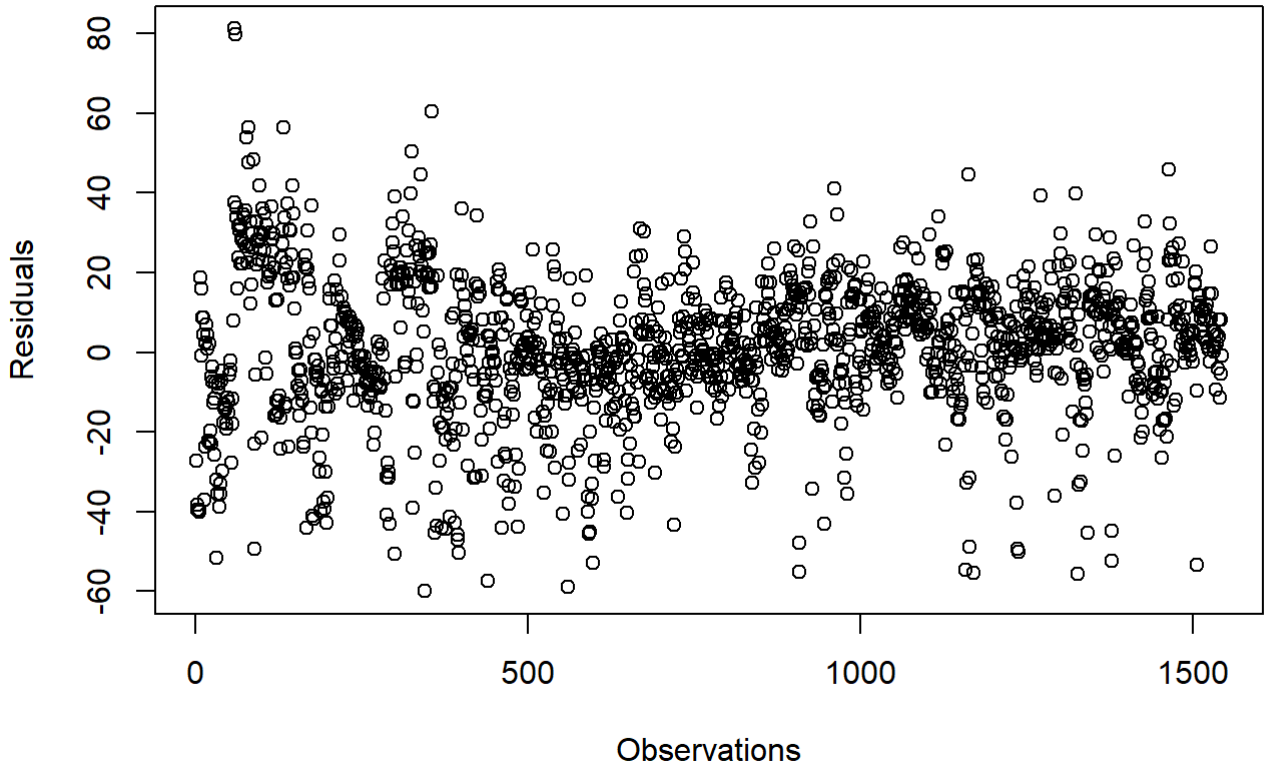
```
## Warning in plot.xy(xy, type, ...): plot type 'prediction' will be truncated to
## first character
```

```
## Warning in plot.xy(xy, type, ...): "abline" is not a graphical parameter
```

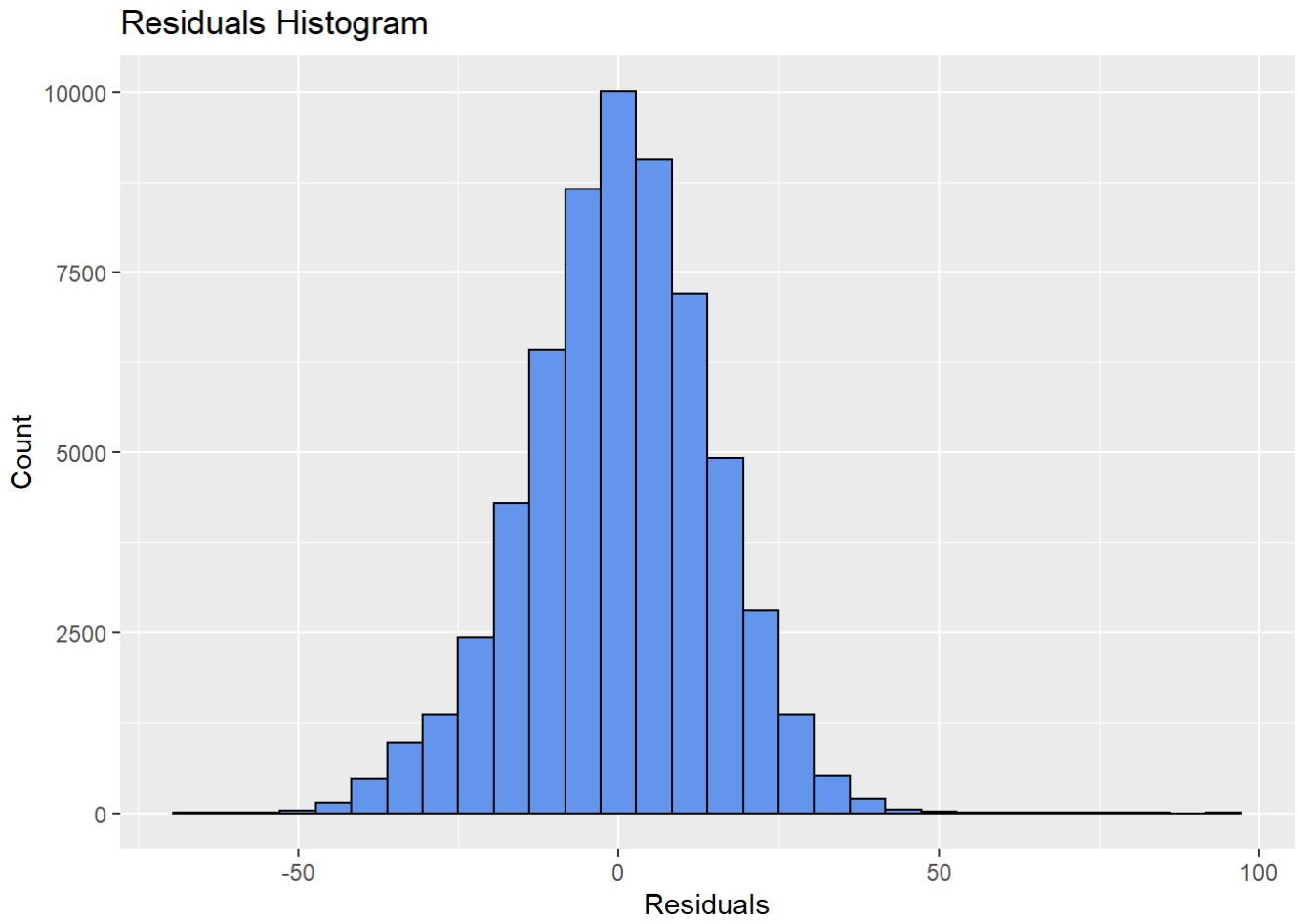
```
## Warning in axis(side = side, at = at, labels = labels, ...): "abline" is not a
## graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "abline" is not a
## graphical parameter
```

```
## Warning in box(...): "abline" is not a graphical parameter
```

```
## Warning in title(...): "abline" is not a graphical parameter
```




```
# histogram of the residuals
ggplot(data = rf.predictions.no.year, aes(x=residuals)) + geom_histogram(bins = 30, fill = "c
ornflowerblue", color = "black") + ggtitle("Residuals Histogram") + xlab("Residuals") + ylab(
"Count")
```


Residuals Histogram
10000
7500
5000
2500
0
Count
-50
0
50
100
Residuals