Audio Processing Vignette

Anna Yanchenko

4/26/2019

Abstract

This vignette demonstrates the audio processing and distance decomposition code used to process the Beethoven symphonies on various recordings of *Twinkle, Twinkle, Little Star*. The various data representations and decompositions are described and calculated. Finally, permutation tests and exploratory plots are explored for analyzing basic systematic differences across pieces for different orchestras.

1. Data Representations

1.1 Audio Recordings

The sample recordings considered here consist of 10 different recordings of *Twinkle, Twinkle, Little Star*. The 10 recordings are grouped into two sets of 5, representing two different "pieces" by five different "orchestras". These correspond to the 10 different orchestras and 37 pieces (Beethoven symphony movements) considered for the Beethoven analysis.

There are no a priori systematic difference between "orchestras" across "pieces" here. The first piece, *No 1*, has recordings that differ in tempo, while piece *No 2* differs in timbre and dynamics. For example, these pieces correspond to movement 1 and movement 2 of a symphony. There is also a reference recording for piece No 1 and piece No 2 that is the orgignal version of *Twinkle, Twinkle, Little Star* at 95 bpm and a steady, medium volume and is considered tuned to A4 = 440 Hz. Orchestras 1 through 4 are tuned to A4 = 440 Hz, while orchestra 5 is considered to be tuned to 442 Hz. The 10 recordings are described in the table below. All recordings are performed by MIDI "piano", except for Orchestra 1's recording of piece No 2.

Piece	Orchestra	Description
No1	Orch1	60 bpm for 4 measures, 90 bpm for four measures
	Orch2	120, 60, 90 bpm for three measures each
	Orch3	60 bpm throughout
	Orch4	95 bpm to start and continuously accel.
	Orch5	95 bpm to start and continuously rit.
———	———-	——
No 2	Orch1	MIDI Organ rather than piano
	Orch2	Increased volume in the bass part
	Orch3	Decr. for four measures, cresc. for four measures
	Orch4	Increased volume in the melody
	Orch5	Continuously decr. then cresc.

1.2 Code Notes

The main scripts for the analysis are `data_processing.R` and `data_analysis.R`. `data_processing.R` contains code to processes the audio and calculates the various data decompositions, while `data_analysis.R` contains code to calculate distances between orchestras for each piece and perform basic permutation tests.

The data processing uses the packages `tuneR` and `seewave`, the audio alignment is performed using the package `dtw` and the distance calculations use `philentropy`.

```r
## Load packages and functions
library(tuneR)
library(reshape2)
require(gridExtra)
library(ggplot2)
library(dtw)
library(abind)
library(seewave)
library(readr)
library(magrittr)
library(philentropy)
library(car)

source("data_processing.R")
source("data_analysis.R")
```

1.3 Data Representations

The two main data representations used are the spectrogram and the chromagram. A MIDIgram representation is also calculated as an intermediate representation in the calculation of the chromagram. All data representations and decompositions considered rely on converting the raw audio files to a spectrogram representation using a Fast Fourier Transform (FFT). The spectrogram represents time and frequency amplitudes across the entire spectrum. The FFT trades off frequency resolution for time resolution. The frequency resolution for the FFT is set to 5 Hz with a temporal resolution of 0.1 s, which corresponds to a window length of 4410*2, though other resolutions are possible.

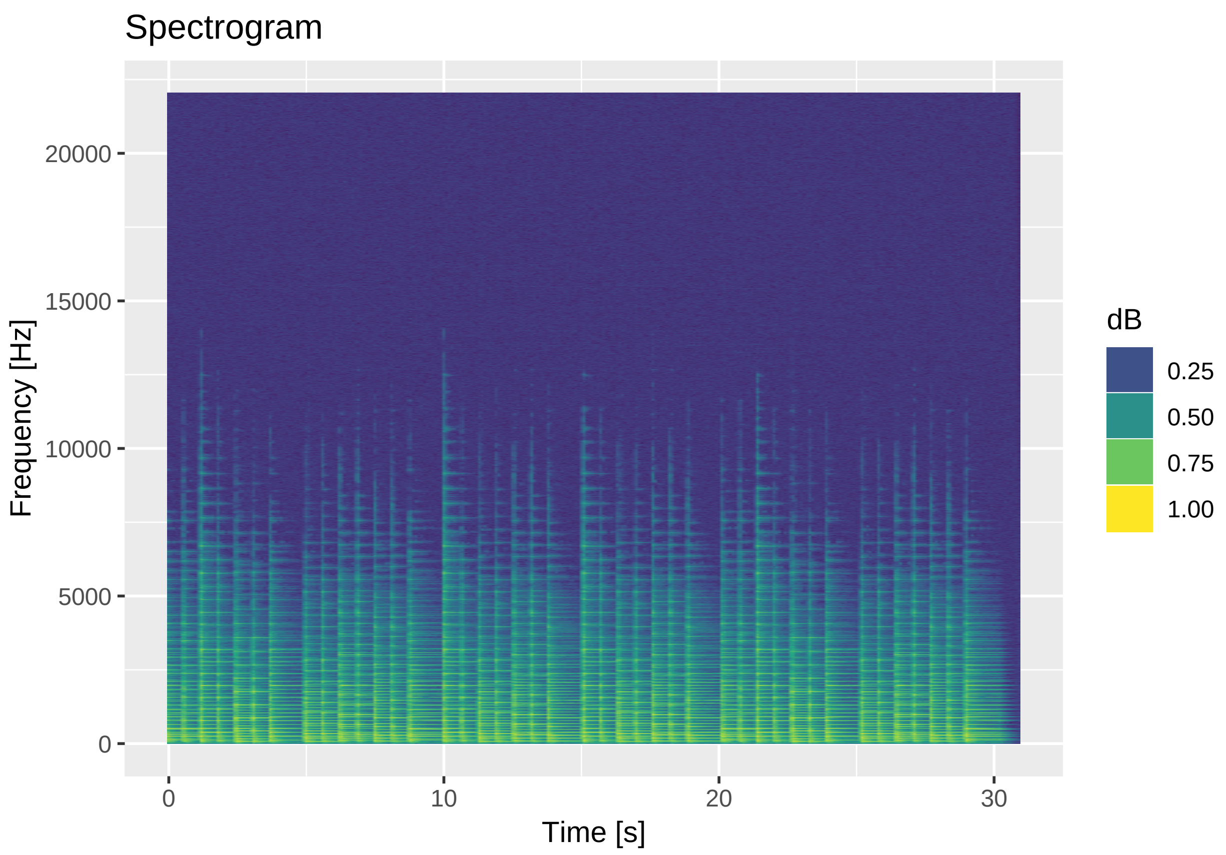
The MIDIgram representation converts the spectrogram frequencies to MIDI pitch numbers to aid in the calculation of the chromagram. In this step, the different tunings of the orchestras can be taken into account by converting the appropriate frequencies to each note pitch (standard tuning is A4 = 440 Hz).

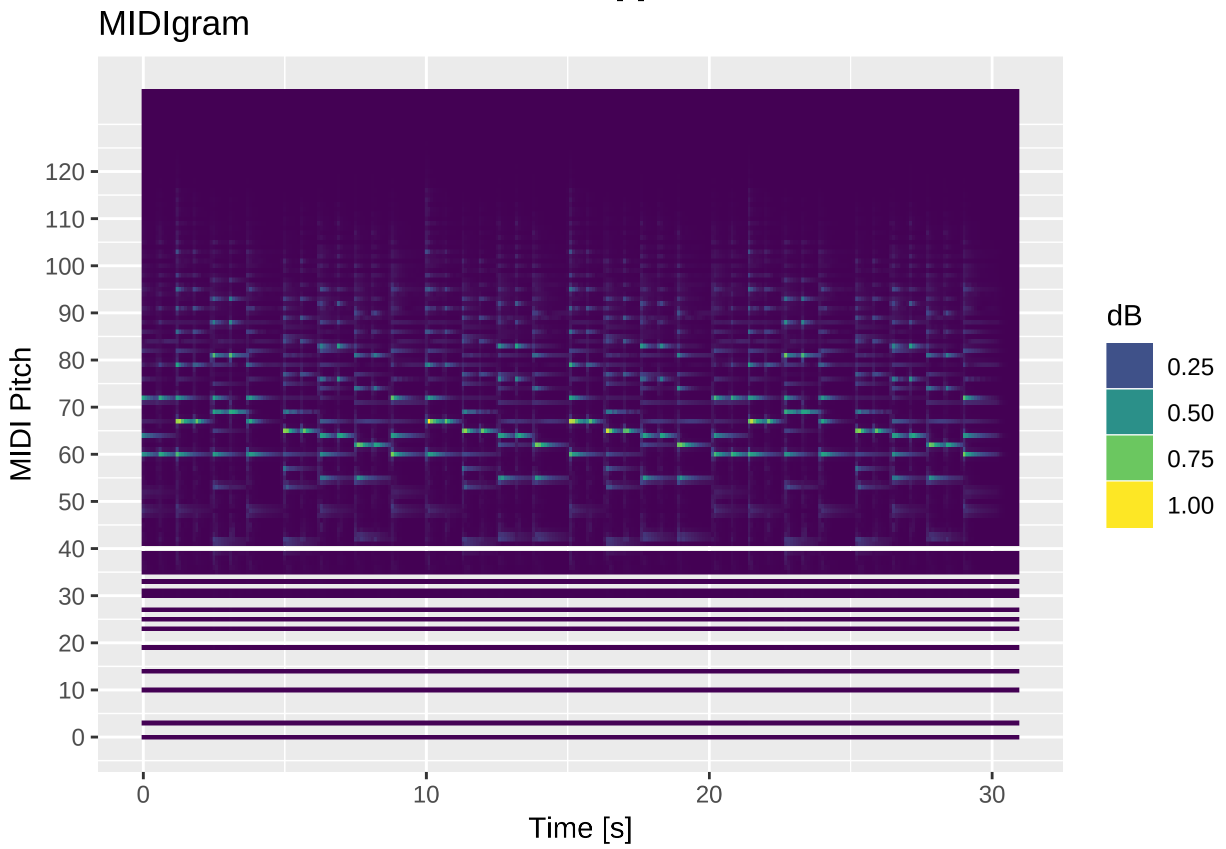
The chromagram representation plots the 12-dimensional chroma values vs. time, where the intensity is the volume (sum of the amplitudes from the spectrogram representation). These chroma vectors represent the note pitch with the octave removed and the pitches are thus collapsed across octaves. There are 12 chroma dimensions, one corresponding to each of the 12 semitones in an octave. The spectrogram, MIDIgram and chromagram for the original recording of *Twinkle, Twinkle, Little Star* are plotted below. The chroma are very distinct in time for this recording, as each note has a clear onset in this simple, piano recording. Note onsets are much less clear in true orchestral audio recordings.

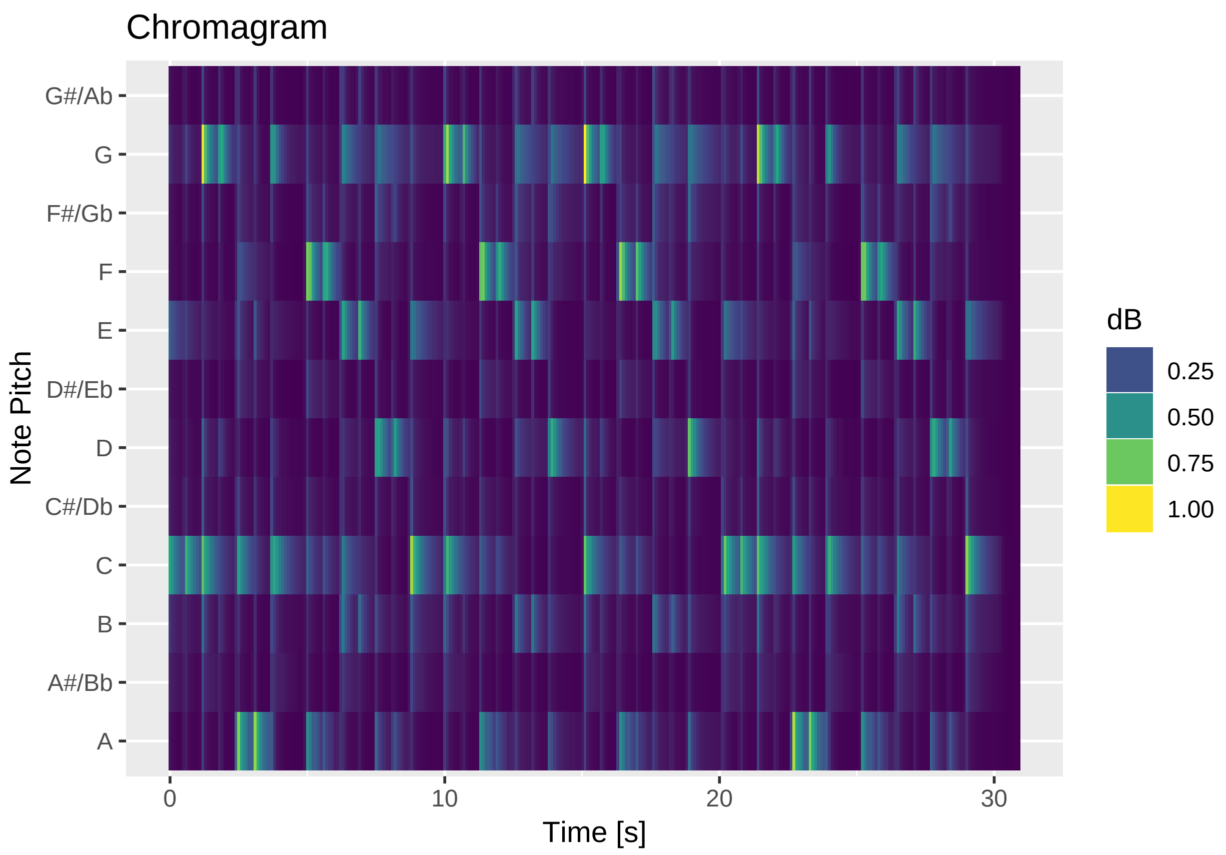
```r
freq2midi <- function(freq, A4 = 440){
  p <- round(69 + 12*log2(freq/A4))
  ## Check for 0
  if(any(p <= 0)){
    p[which(p <=0)] = 0
  }
  return(p)
}

## Convert MIDI pitch to frequency
midi2freq <- function(pitch, A4 = 440){
  f <- A4*2^((pitch - 69)/12)
  return(f)
}
```

```r
## Convert MIDI pitch (0-127) to chroma number,
## ref  is MIDI pitch of lowest note that chroma starts with, default is A0
# A0 = 21
# B0 = 23
# C1 = 24
# D1 = 26
# E1 = 28
# F1 = 29
# G1 = 31
midi2chroma <- function(midi, ref = 21){
  chrom <- (midi - ref)%%12 + 1
  labs <- c("A", "A#/Bb", "B", "C", "C#/Db", "D", "D#/Eb", "E", "F",
            "F#/Gb", "G", "G#/Ab")
  ind <- ref - 21 + 1
  ## Reorder labels
  labels <- c(labs[ind:length(labs)], labs[1:ind-1])
  return(list(chroma = chrom, labels = labels))
}
```

```r
curr <- readMP3("Music/Twinkle-Twinkle/Twinkle-Twinkle-No1-original.mp3")
sr <- curr@samp.rate
pieces <- curr@left
ind <- which(pieces > 0 )
ind1 <- ind[1] ## remove silence before piece
ind2 <- tail(ind, n = 1) ## remove silence from end of piece
curr <- pieces[ind1:ind2]
e.twinkle <- encode(curr, sr, window.length = 4410*2, A4 = 440, plot = TRUE)
```


Spectrogram
Frequency [Hz]
20000
15000
10000
5000
0
0
10
20
30
Time [s]
dB
0.25
0.50
0.75
1.00


MIDIgram
MIDI Pitch
120
110
100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
Time [s]
dB
0.25
0.50
0.75
1.00



```
# Can aditionall plot each reprsentation individually
#plot.chrom(e.twinkle$chromagram, e.twinkle$labels, e.twinkle$spec$t)
```

2. Distance Decompositions

Orchestral performances can differ across several different dimensions. The musical dimensions across which orchestras can differ that are considered here are:

- Tempo
- Volume
- Balance
- Timbre

Each decomposition is performed to find a density over time (normalized between 0 and 1) so that density distance metrics can be calculated to compare the various orchestra decompositions.

`process.curves` is the main function to encode each recording and calculate the tempo, volume and balance data decompositions. The `process.spec` function performs decompositions using the spectrogram representation to explore balance and volume by frequency group. Finally, `process.timbre` calculates the timbre based decompositions. Additionally, there are analogous functions ending in 9, e.g. `process.curves9`, that handle the case when multiple separate audio files correspond to the same piece and need to be concatenated together.

```
folder <- "Music/Twinkle-Twinkle/"
## Read in pieces and save as chromagrams
all.files <- list.files("Music/Twinkle-Twinkle/.", recursive = TRUE)
print(all.files)
```

```
##  [1] "Twinkle-Twinkle-No1-orch1-60-90.mp3"
##  [2] "Twinkle-Twinkle-No1-orch2-120-60-90.mp3"
##  [3] "Twinkle-Twinkle-No1-orch3-60.mp3"
##  [4] "Twinkle-Twinkle-No1-orch4-95accel.mp3"
##  [5] "Twinkle-Twinkle-No1-orch5-95decel.mp3"
##  [6] "Twinkle-Twinkle-No1-original.mp3"
##  [7] "Twinkle-Twinkle-No2-orch1-organ.mp3"
##  [8] "Twinkle-Twinkle-No2-orch2-vol-bass-loud-6.mp3"
##  [9] "Twinkle-Twinkle-No2-orch3-vol-decr-cres-4.mp3"
## [10] "Twinkle-Twinkle-No2-orch4-vol-melody-loud-7.mp3"
## [11] "Twinkle-Twinkle-No2-orch5-vol-var-5.mp3"
## [12] "Twinkle-Twinkle-No2-original.mp3"
## [13] "Twinkle-Twinkle-structural.mp3"
```

```
orchs <- c("orch1", "orch2", "orch3", "orch4", "orch5")

nOrch <- 5 ## number of distinct "orchestras"
nPieces <-  2 # number of distinct "pieces"

file.list <- all.files[1:12] ## file list for No 1
A4.list <- c(440, 440, 440, 440, 442, 440) ## tunings by orchestra

## Read in each recording and process different decompositions
twinkle.process <- process.curves(file.list, folder, orchs, A4.list, ref.word = "original",
                        window.length = 4410*2)
```

```
## [1] "Twinkle-Twinkle-No1-orch1-60-90.mp3"
## [1] "Twinkle-Twinkle-No1-orch2-120-60-90.mp3"
## [1] "Twinkle-Twinkle-No1-orch3-60.mp3"
## [1] "Twinkle-Twinkle-No1-orch4-95accel.mp3"
## [1] "Twinkle-Twinkle-No1-orch5-95decel.mp3"
## [1] "Twinkle-Twinkle-No1-original.mp3"
## [1] "Twinkle-Twinkle-No2-orch1-organ.mp3"
## [1] "Twinkle-Twinkle-No2-orch2-vol-bass-loud-6.mp3"
## [1] "Twinkle-Twinkle-No2-orch3-vol-decr-cres-4.mp3"
## [1] "Twinkle-Twinkle-No2-orch4-vol-melody-loud-7.mp3"
## [1] "Twinkle-Twinkle-No2-orch5-vol-var-5.mp3"
## [1] "Twinkle-Twinkle-No2-original.mp3"
## [1] 1 7
## [1] 2 8
## [1] 3 9
## [1]  4 10
## [1]  5 11
```

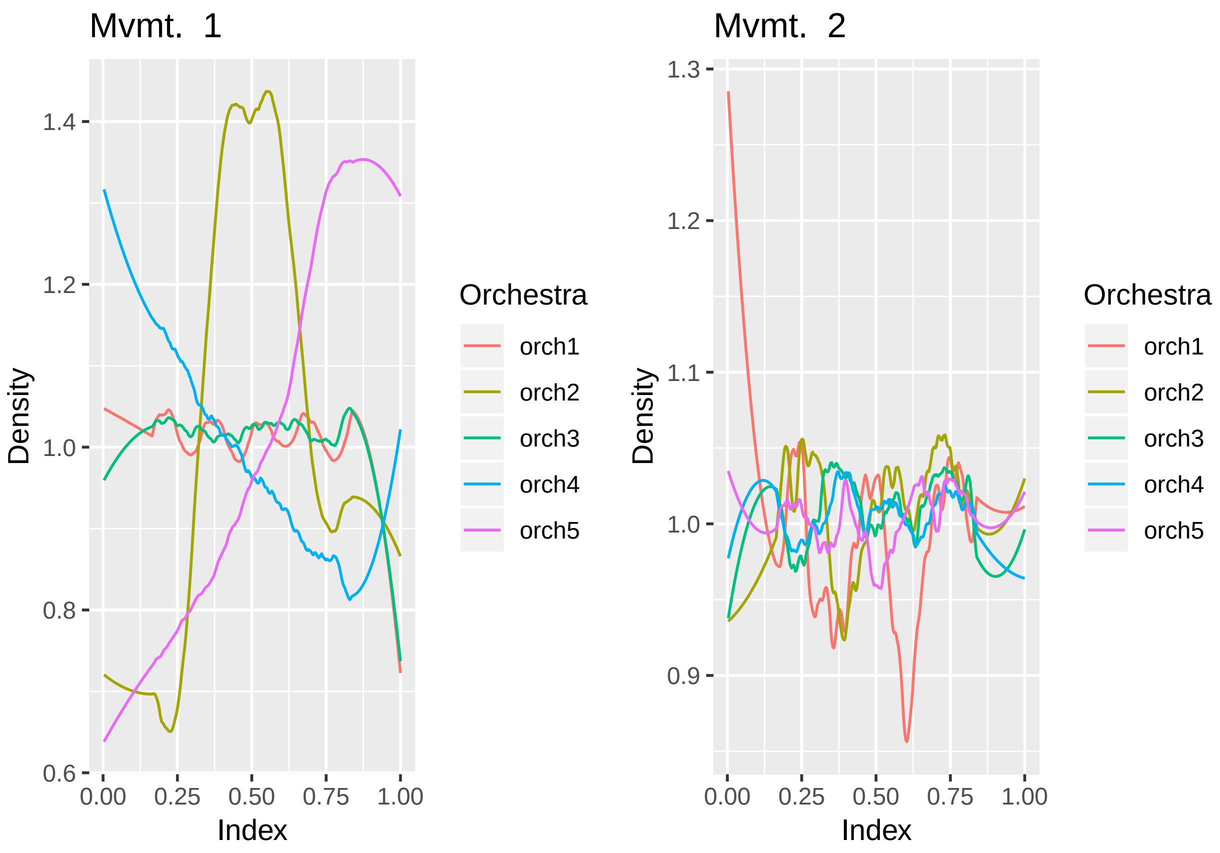
2.1 Alignment

Before any data decompositions are calculated, the recordings need to be aligned temporally so that each decomposition only measures the feature of interest and is invariant to timing differences. As each orchestra is performing the same piece and thus must perform the same notes, Dynamic Time Warping is used to temporally align the different recordings before further processing. The alignment is performed on the chromagram data representation.

2.2 Tempo

The alignment indices from the dynamic time warping alingment step are used to calculate differences in tempo curves between different orchestras. The derivative of the indices is found using the Savitzky-Golay filter to yield the tempo density curves.

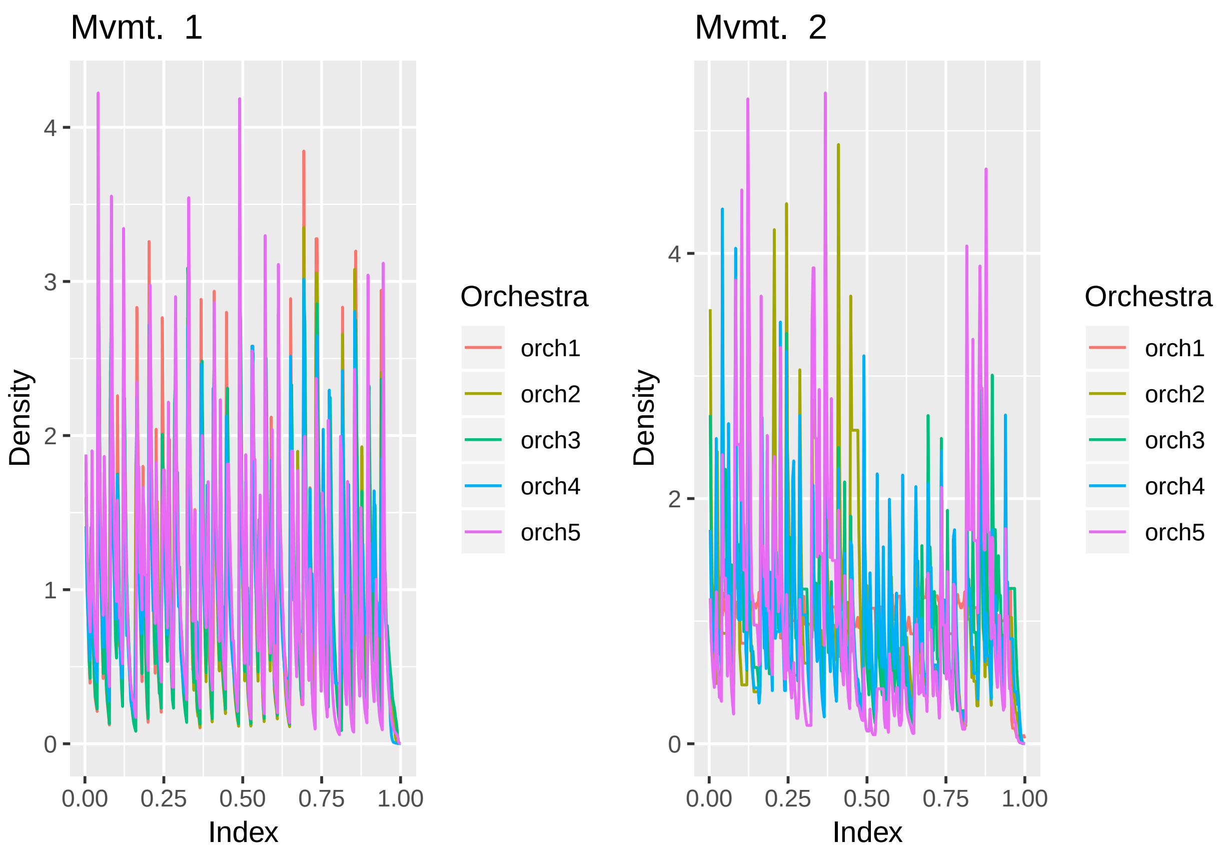
```r
p.b2c <- plot.tempo(twinkle.process$tempo)
grid.arrange(p.b2c[[1]], p.b2c[[2]], nrow = 1)
```



2.3 Volume

The volume curves are calculated by finding the overall volume at each point in time for the aligned chromagrams and then normalizing by the average volume for the entire piece. The sharp peaks in the volume curves are from the note onsets, which are quite evident in the piano, MIDI recordings considered here.

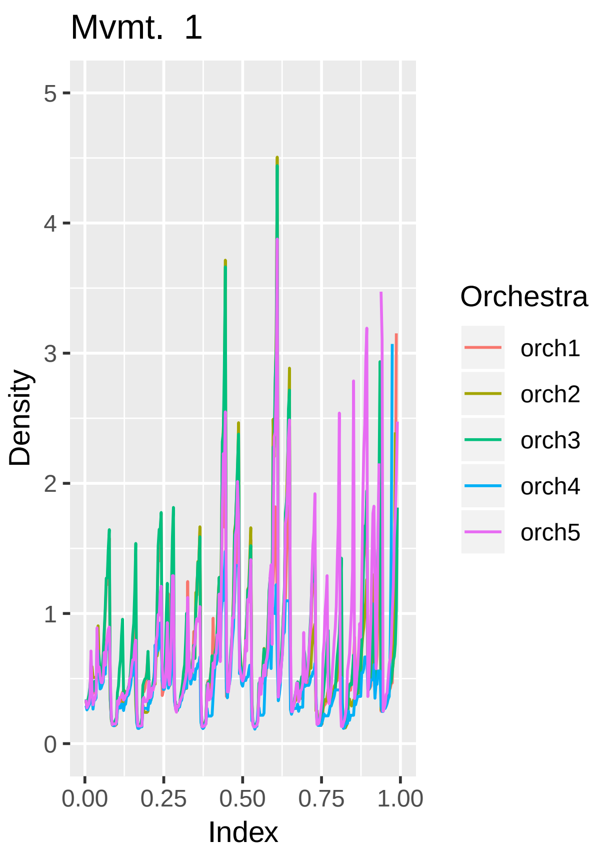
```r
p.b2c <- plot.tempo(twinkle.process$volume)
grid.arrange(p.b2c[[1]], p.b2c[[2]], nrow = 1)
```

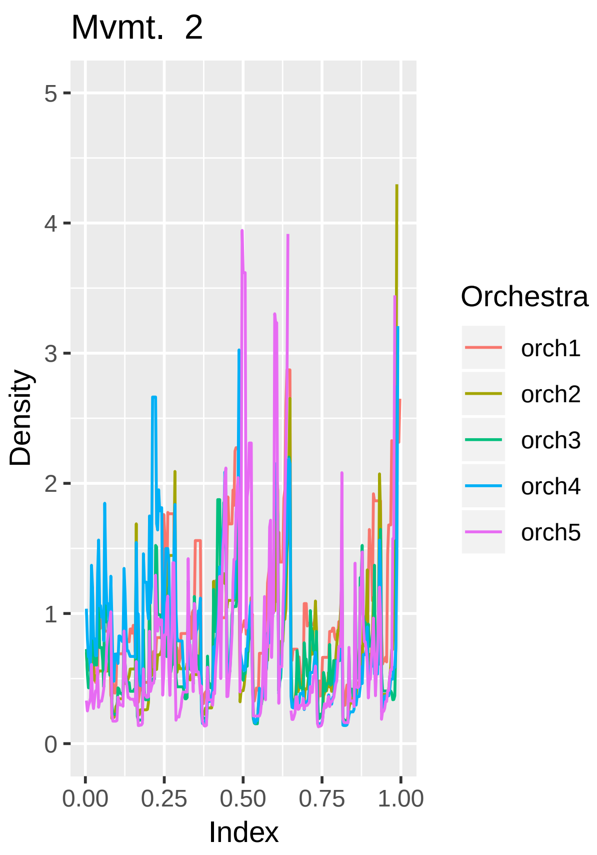


2.4 Spectral Flatness - Overall

Spectral flatness is the ratio of the geometric and arithmetic means of the amplitude of a signal and is a measure of the tonality of a recording. This measure can give an indication of the timbre of different orchestras, and is normalized to be a density.

```
plot.tempo.SF <- function(chrom.list){
  plot.list <- list()
  for(i in 1:length(chrom.list[[1]])){
    m1 <- data.frame(lapply(chrom.list, `[[`, i))
    n <- nrow(m1)
    b.m1 <- melt(m1)
    colnames(b.m1) <- c("Orchestra", "Density")
    b.m1$Index <- rep(seq(n), times = length(orchs))/n
    p <- ggplot(b.m1, aes(x = Index, y = Density, col = Orchestra)) +
      geom_line() +
      ggtitle(paste("Mvmt. ", i)) +
      ylim(0, 5)
    plot.list[[i]] <- p
  }
  return(plot.list)

}
p.b2c <- plot.tempo.SF(twinkle.process$SF)
grid.arrange(p.b2c[[1]], p.b2c[[2]], nrow = 1)
```





2.5 Balance - Volume by Frequency

The balance of different orchestras is apparent in the relative volume of the different instruments and sections. Ideally, this could be explored by extracting each instrument's volume over time from the spectrogram. However, the decomposition of an audio signal to each instrument is very challenging and an open research problem. To approximate the volume of each instrument, the volume is calculated over a frequency range, approximating the volume of different groups of instruments.

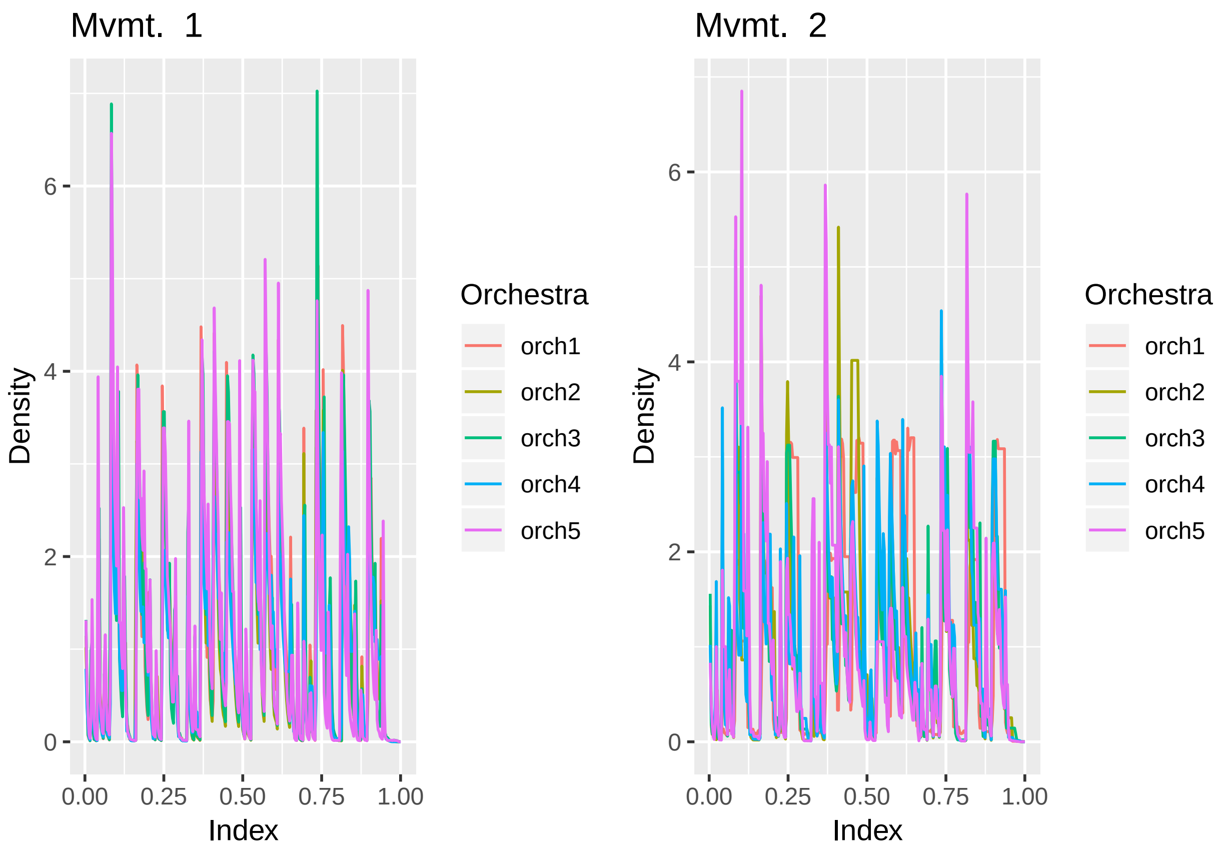
f.inds1 indicates the frequency indices in the spectrogram at which to start a grouping, while *f.inds2* indicates the ending indices for each frequency group. 10 groupings are considered here, 1 for each octave, A0 - A7, and three groupings for upper harmonics. *align* are the indices from the alignment of the chromagrams via dynamic time warping that are needed to temporally align the spectrogram. It is therefore important for the *window.length* for the `process.spec` and `process.curves` functions to be identical. Additionally, the reference recording should not be input to `process.spec` to calculate the volume by frequency decomposition. Different orchestra tunings can be accounted for by selecting the appropriate frequencies to correspond to the desired range of note pitches.

```
## Select Octaves and Upper Harmonics Chuncked -  account for tuning
## Octaves starting from A0 to A7, then three bins upper harmonics
freq.list <- seq(0, 22045, 5)
f.inds1 <- list()
f.inds2 <- list()

## Start and End frequencies
f.inds1[seq(1,11)] <- replicate(11, which(freq.list %in% c(30, 55, 110, 220, 440,
                                                          880, 1760, 3520, 5000, 10000)),
                                simplify = FALSE)
```

```r
f.inds2[seq(1,11)] <- replicate(11, c((f.inds1[[1]] - 1)[-1], length(freq.list)),
                                simplify = FALSE)
## File list needs to be all orchetra recordings sequentially, don't include reference
file.list <- all.files[c(1,7,2,8,3,9,4,10,5,11)]

## Use indices from alingnment above
align1 <-unlist(twinkle.process$align.orch, recursive = FALSE)
nMove <- 2 ## two movements
twinkle.spec <- process.spec(file.list, folder, orchs, A4.list,
                             window.length = 4410*2, f.inds1, f.inds2, align1, nMove)
```

```
## [1] "Twinkle-Twinkle-No1-orch1-60-90.mp3"
## [1] "Twinkle-Twinkle-No2-orch1-organ.mp3"
## [1] "Twinkle-Twinkle-No1-orch2-120-60-90.mp3"
## [1] "Twinkle-Twinkle-No2-orch2-vol-bass-loud-6.mp3"
## [1] "Twinkle-Twinkle-No1-orch3-60.mp3"
## [1] "Twinkle-Twinkle-No2-orch3-vol-decr-cres-4.mp3"
## [1] "Twinkle-Twinkle-No1-orch4-95accel.mp3"
## [1] "Twinkle-Twinkle-No2-orch4-vol-melody-loud-7.mp3"
## [1] "Twinkle-Twinkle-No1-orch5-95decel.mp3"
## [1] "Twinkle-Twinkle-No2-orch5-vol-var-5.mp3"
```

```r
## Plot 2nd group only
p.b2c <- plot.spec.curves(twinkle.spec, 2)
p5 <- grid.arrange(p.b2c[[1]], p.b2c[[2]], nrow = 1)
```



```r
# Plot all frequency groups
#for(i in 1:10){
```

```
#  p.b2c <- plot.spec.curves(twinkle.spec, i)
#  p5 <- grid.arrange(p.b2c[[1]], p.b2c[[2]], nrow = 1)
#}
```

2.6 Timbre

Timbre can be described as the "color" of a sound and allows listeners to distinguish between different instruments playing the same note pitch. Timbre is a subjective concept and is difficult to measure from an audio file. However, it is one of the principal dimensions along which recordings by different orchestras can differ, as many orchestras have a distinctive "sound". Some aspects of timbre are evident in the upper harmonics of a spectrum, as more "resonant" tones lead to more higher harmonics above the fundamental frequency pitch.

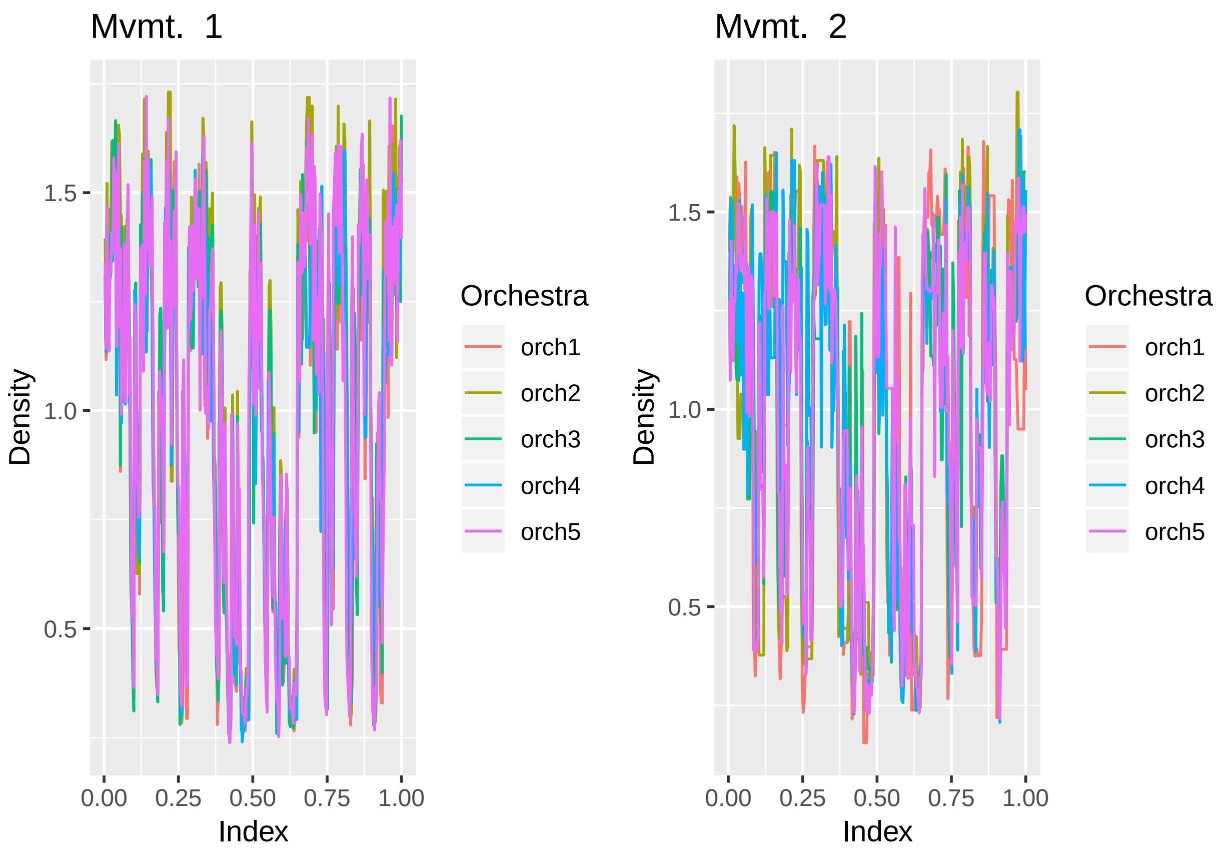
```
## Select Octaves and Upper Harmonics Chuncked -  account for tuning
## Octaves starting from A0 to A7, then three bins upper harmonics
freq.list <- seq(0, 22045, 5)
f.inds1 <- list()
f.inds2 <- list()
ref.word <- "original"
## Start and End frequencies
freq.start <- c(30, 55, 110, 220, 440,880, 1760, 3520, 5000, 10000) ## in HZ
f.inds1[seq(1,11)] <- replicate(11, which(freq.list %in% freq.start),
                                simplify = FALSE)
f.inds2[seq(1,11)] <- replicate(11, c((f.inds1[[1]] - 1)[-1], length(freq.list)),
                                simplify = FALSE)
## File list needs to be all orchetra recordings sequentially, DO include reference
file.list <- all.files[c(1,7,2,8,3,9,4,10,5,11, 6, 12)]

## Use indices from alingnment above
align1 <-unlist(twinkle.process$align.orch, recursive = FALSE)
nMove <- 2 ## two movements
nMFCC <- 13 # Number of MFCC coefficients
twinkle.timbre <- process.timbre(file.list, folder, orchs, A4.list, ref.word,
                         window.length = 4410*2, f.inds1, f.inds2, align1, nMove,
                         nMFCC, wintime = 0.05, hoptime = 0.1)
```

```
## [1] "Twinkle-Twinkle-No1-orch1-60-90.mp3"
## [1] "Twinkle-Twinkle-No2-orch1-organ.mp3"
## [1] "Twinkle-Twinkle-No1-orch2-120-60-90.mp3"
## [1] "Twinkle-Twinkle-No2-orch2-vol-bass-loud-6.mp3"
## [1] "Twinkle-Twinkle-No1-orch3-60.mp3"
## [1] "Twinkle-Twinkle-No2-orch3-vol-decr-cres-4.mp3"
## [1] "Twinkle-Twinkle-No1-orch4-95accel.mp3"
## [1] "Twinkle-Twinkle-No2-orch4-vol-melody-loud-7.mp3"
## [1] "Twinkle-Twinkle-No1-orch5-95decel.mp3"
## [1] "Twinkle-Twinkle-No2-orch5-vol-var-5.mp3"
## [1] "Twinkle-Twinkle-No1-original.mp3"
## [1] "Twinkle-Twinkle-No2-original.mp3"
## [1] 1 2
## [1] 3 4
## [1] 5 6
## [1] 7 8
## [1]  9 10
```

2.6.1 Spectral Flatness by Frequency

As spectral flatness measures the tonality of a sound, it can be used to explore timbre. However, the timbre of an orchestra differs across instruments, so, similar to the volume by frequency approach used to explore balance, the spectral flatness is calculated for each of the specified frequency groups to explore timbre by octave/instrument group.

```
## Plot second group only
p.b2c <- plot.spec.curves(twinkle.timbre$SF.byF, 2)
p5 <- grid.arrange(p.b2c[[1]], p.b2c[[2]], nrow = 1)
```

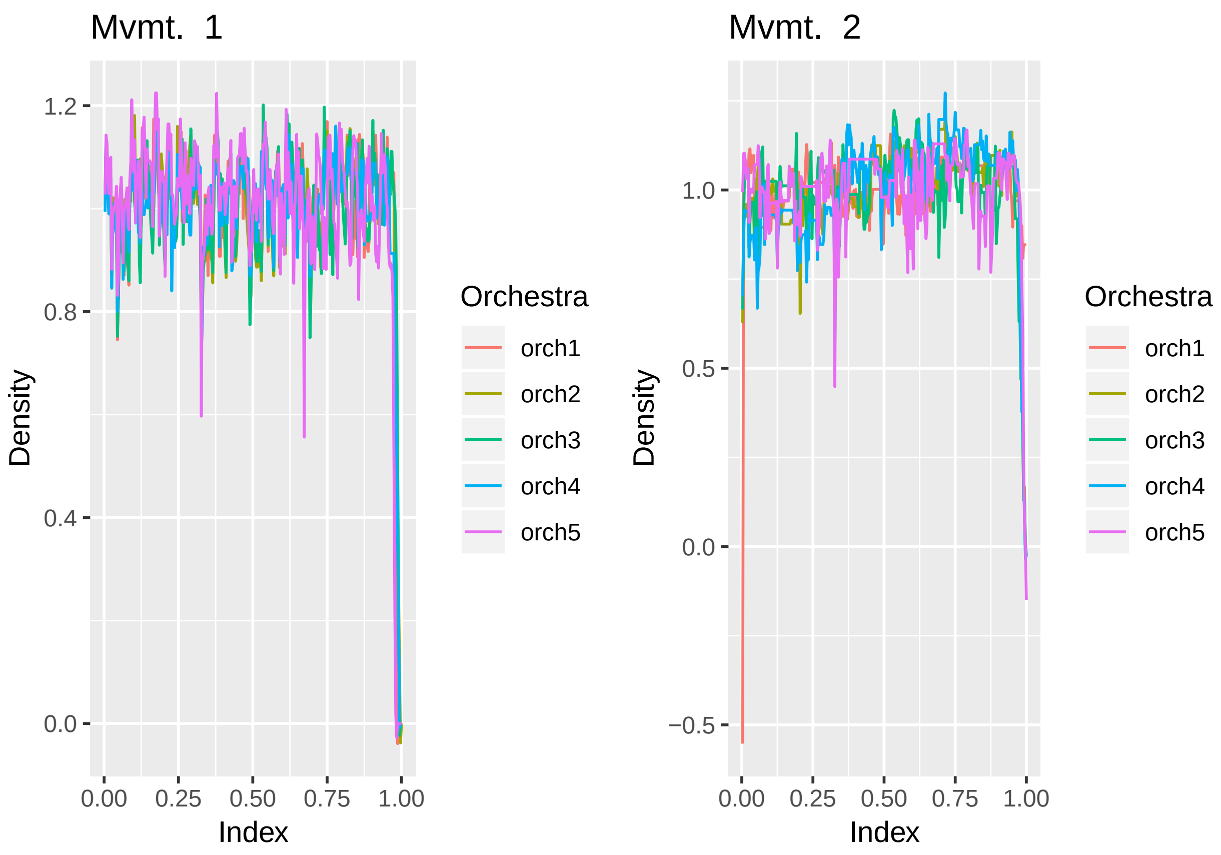


```
## plot all frequency groups
#for(i in 1:length(freq.start)){
#  p.b2c <- plot.spec.curves(twinkle.timbre$SF.byF, i)
#  p5 <- grid.arrange(p.b2c[[1]], p.b2c[[2]], nrow = 1)
#}
```

2.6.2 Mel-Frequency Cepstrum Coefficients (MFCC)

MFC (mel-frequency cepstrum) coefficients are used often in speech recognition and music information retrieval and also are an approximate measure of timbre. MFCC values are essentially spectrogram values from the FFT warped to the mel-scale and then the discrete cosine transform of the log of these values is taken to give the MFCC. The mel-scale is supposed to better represent how humans perceive frequency, that is the scale places pitches at what would be heard as equal spacing.

```
## Plot 2nd MFCC only
p.b2c <- plot.spec.curves(twinkle.timbre$MFCC, 2)
p5 <- grid.arrange(p.b2c[[1]], p.b2c[[2]], nrow = 1)
```



```
## Plot all MFCC curves
#for(i in 1:(nMFCC-1)){
#  p.b2c <- plot.spec.curves(twinkle.timbre$MFCC, i)
#  p5 <- grid.arrange(p.b2c[[1]], p.b2c[[2]], nrow = 1)
#}
```

3. Distance Calculations

Once each decomposition described above is calculated for each orchestra, the orchestras are compared by calculating a distance measure between each orchestra pair for each piece. Several distance measures are considered and described below. All distance metrics are calculated using the `philentropy` package. Any of the distance measures in that package can be calculated with the `calc.dist` and `calc.dist.spec` functions.

The Jensen-Shannon Divergence is calculated here for all data decompositions except for the MFCC representation. Except for the MFCC, all other data decompositions are treated as densities. However, MFCC values can be negative and meaningful, so different distance measures, such as the Euclidean distance, are calculated.

```
tempo.dist <- calc.dist(twinkle.process$tempo, "JSD")
volume.dist <- calc.dist(twinkle.process$volume, "JSD")
SF.dist <- calc.dist(twinkle.process$SF, "JSD")

spec1.dist <- calc.dist.spec(twinkle.spec, "JSD", 1)
spec2.dist <- calc.dist.spec(twinkle.spec, "JSD", 2)
spec3.dist <- calc.dist.spec(twinkle.spec, "JSD", 3)
spec4.dist <- calc.dist.spec(twinkle.spec, "JSD", 4)
```

```
spec5.dist <- calc.dist.spec(twinkle.spec, "JSD", 5)
spec6.dist <- calc.dist.spec(twinkle.spec, "JSD", 6)
spec7.dist <- calc.dist.spec(twinkle.spec, "JSD", 7)
spec8.dist <- calc.dist.spec(twinkle.spec, "JSD", 8)
spec9.dist <- calc.dist.spec(twinkle.spec, "JSD", 9)
spec10.dist <- calc.dist.spec(twinkle.spec, "JSD", 10)

SF1.dist <- calc.dist.spec(twinkle.timbre$SF.byF, "JSD", 1)
SF2.dist <- calc.dist.spec(twinkle.timbre$SF.byF, "JSD", 2)
SF3.dist <- calc.dist.spec(twinkle.timbre$SF.byF, "JSD", 3)
SF4.dist <- calc.dist.spec(twinkle.timbre$SF.byF, "JSD", 4)
SF5.dist <- calc.dist.spec(twinkle.timbre$SF.byF, "JSD", 5)
SF6.dist <- calc.dist.spec(twinkle.timbre$SF.byF, "JSD", 6)
SF7.dist <- calc.dist.spec(twinkle.timbre$SF.byF, "JSD", 7)
SF8.dist <- calc.dist.spec(twinkle.timbre$SF.byF, "JSD", 8)
SF9.dist <- calc.dist.spec(twinkle.timbre$SF.byF, "JSD", 9)
SF10.dist <- calc.dist.spec(twinkle.timbre$SF.byF, "JSD", 10)

MFCC1.dist <- calc.dist.spec(twinkle.timbre$MFCC, "euclidean", 1)
MFCC2.dist <- calc.dist.spec(twinkle.timbre$MFCC, "euclidean", 2)
MFCC3.dist <- calc.dist.spec(twinkle.timbre$MFCC, "euclidean", 3)
MFCC4.dist <- calc.dist.spec(twinkle.timbre$MFCC, "euclidean", 4)
MFCC5.dist <- calc.dist.spec(twinkle.timbre$MFCC, "euclidean", 5)
MFCC6.dist <- calc.dist.spec(twinkle.timbre$MFCC, "euclidean", 6)
MFCC7.dist <- calc.dist.spec(twinkle.timbre$MFCC, "euclidean", 7)
MFCC8.dist <- calc.dist.spec(twinkle.timbre$MFCC, "euclidean", 8)
MFCC9.dist <- calc.dist.spec(twinkle.timbre$MFCC, "euclidean", 9)
MFCC10.dist <- calc.dist.spec(twinkle.timbre$MFCC, "euclidean", 10)
MFCC11.dist <- calc.dist.spec(twinkle.timbre$MFCC, "euclidean", 11)
MFCC12.dist <- calc.dist.spec(twinkle.timbre$MFCC, "euclidean", 12)

twinkle.dist <- abind(abind(tempo.dist, along = 3),
                      abind(volume.dist, along = 3),
                      abind(SF.dist, along = 3),
                      abind(spec1.dist, along = 3), abind(spec2.dist, along = 3),
                      abind(spec3.dist, along = 3), abind(spec4.dist, along = 3),
                      abind(spec5.dist, along = 3), abind(spec6.dist, along = 3),
                      abind(spec7.dist, along = 3), abind(spec8.dist, along = 3),
                      abind(spec9.dist, along = 3), abind(spec10.dist, along = 3),
                      abind(SF1.dist, along = 3), abind(SF2.dist, along = 3),
                      abind(SF3.dist, along = 3), abind(SF4.dist, along = 3),
                      abind(SF5.dist, along = 3), abind(SF6.dist, along = 3),
                      abind(SF7.dist, along = 3), abind(SF8.dist, along = 3),
                      abind(SF9.dist, along = 3), abind(SF10.dist, along = 3),
                      abind(MFCC1.dist, along = 3), abind(MFCC2.dist, along = 3),
                      abind(MFCC3.dist, along = 3), abind(MFCC4.dist, along = 3),
                      abind(MFCC5.dist, along = 3), abind(MFCC6.dist, along = 3),
                      abind(MFCC7.dist, along = 3), abind(MFCC8.dist, along = 3),
                      abind(MFCC9.dist, along = 3), abind(MFCC10.dist, along = 3),
                      abind(MFCC11.dist, along = 3), abind(MFCC12.dist, along = 3),
                  rev.along=0)
dimnames(twinkle.dist)[[1]] <- orchs
dimnames(twinkle.dist)[[2]] <- orchs
```

```r
dimnames(twinkle.dist)[[3]] <- c("No1", "No2")
dimnames(twinkle.dist)[[4]] <- c("tempo", "volume", "SF", "spec1", "spec2", "spec3",
                                "spec4", "spec5", "spec6", "spec7", "spec8", "spec9",
                                "spec10", "SF1", "SF2", "SF3", "SF4", "SF5", "SF6",
                                "SF7", "SF8", "SF9", "SF10", "MFCC1", "MFCC2", "MFCC3",
                                "MFCC4", "MFCC5", "MFCC6", "MFCC7", "MFCC8", "MFCC9",
                                "MFCC10", "MFCC11", "MFCC12")
```

4. Permutation Tests

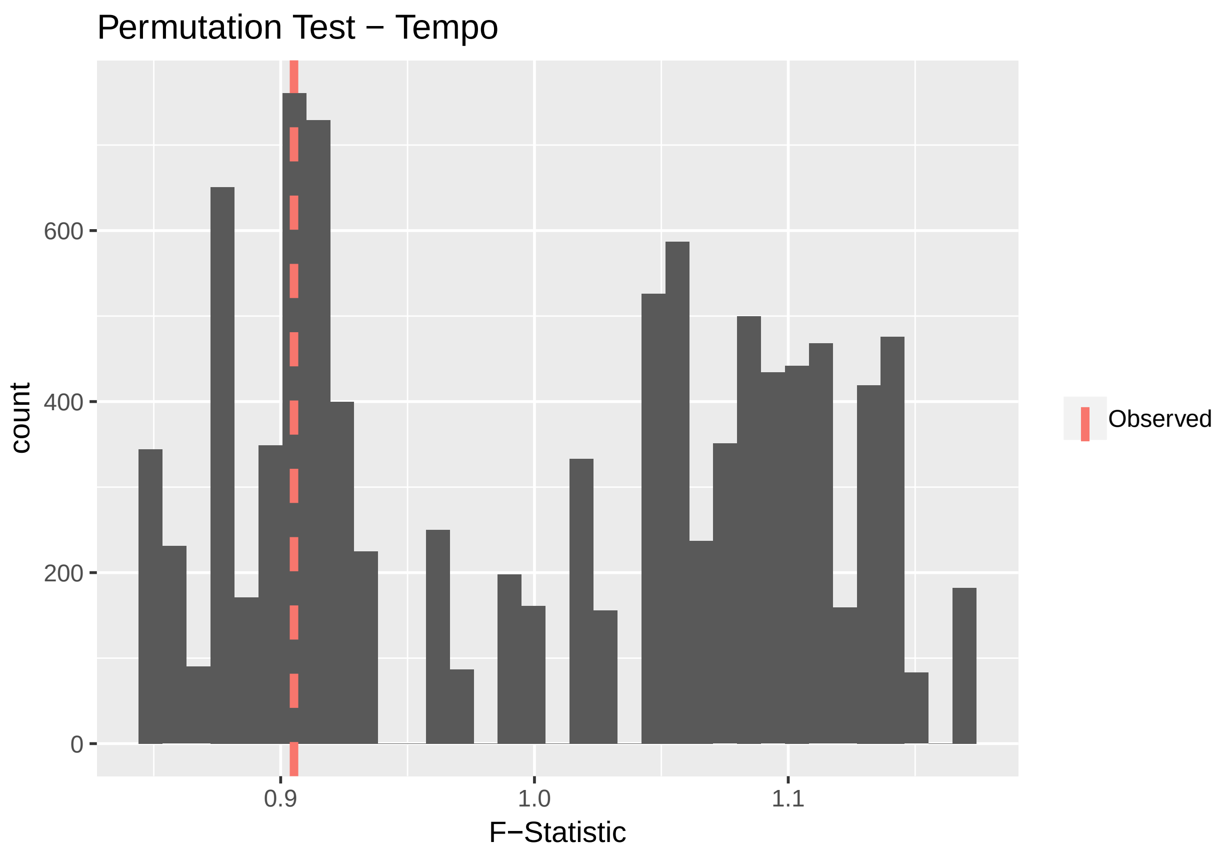
To test for the presence of orchestra effects, permutation tests on the various observed distance matrices are performed as follows:

1. Permute the order of orchestras within a piece
2. Select unique distance values by piece, resulting in a (nOrch*(nOrch-1))/2 x nPieces matrix
3. Run an ANOVA for column (piece) and row (orchestra) effects
4. Record F-statistic from ANOVA (H0 distribution, no orchestra effects)
5. Compare to F-statistic calculated with observed data

The null hypothesis for the F-statistic is that there are no systematic orchestra differences across pieces and that orchestras are exchangeable (as is the case here). The p-values for each test across each distance measure are calculated and the distributions of the F statistics are displayed below.

4.1 Tempo

```r
it <- 10000 ## number of iterations of permutation test to perform
dist.mat <- twinkle.dist[,,,1]
tempo.permute <- permutation.test(dist.mat, it)


ggplot(tempo.permute$permute.f, aes(x = tempo.permute$permute.f[,1])) +
  geom_histogram(bins = 35) +
  geom_vline(aes(xintercept=tempo.permute$obs.f, color = "Observed"),
             linetype = "dashed", show.legend = TRUE, size = 1.5) +
  theme(legend.title = element_blank()) +
  labs(x = "F-Statistic")+
  ggtitle("Permutation Test - Tempo")
```

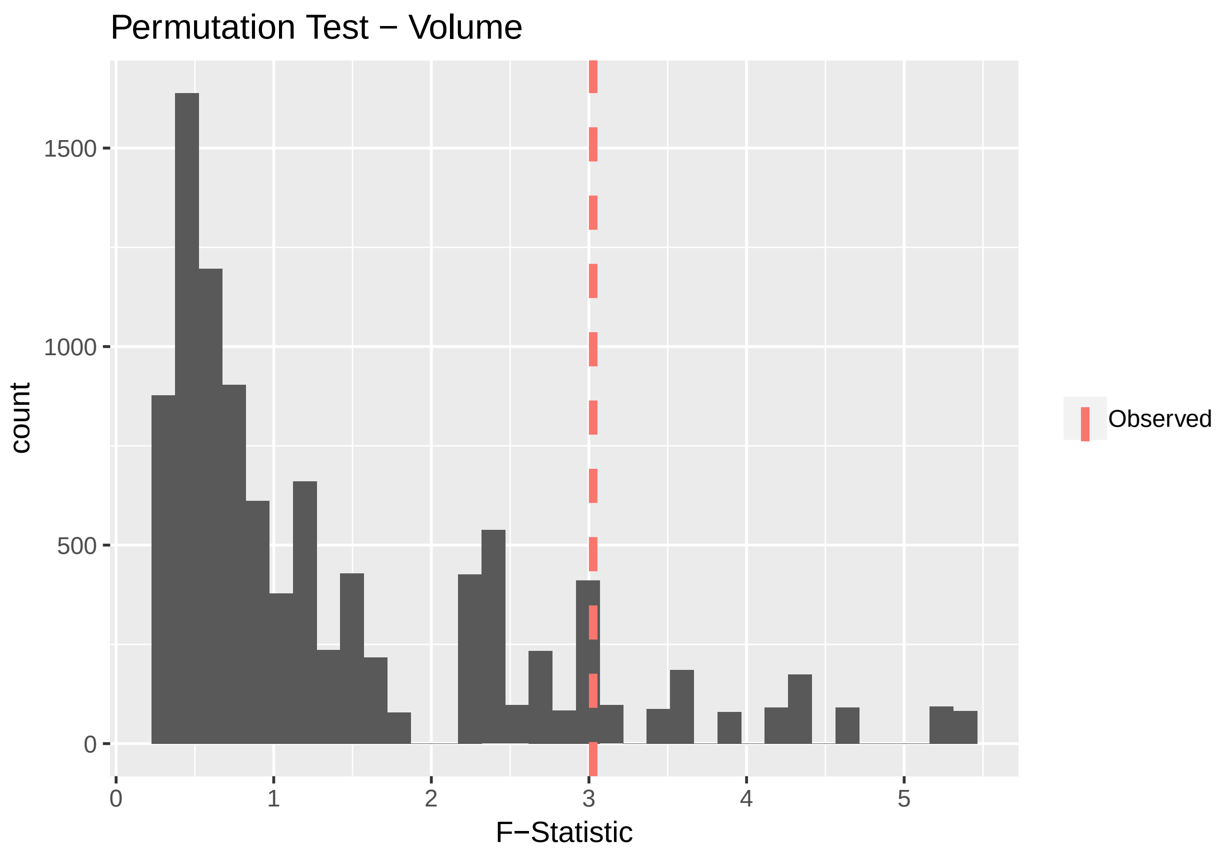


```
print(tempo.permute$p.val)
```

```
## [1] 0.7981
```

4.2 Volume

```
dist.mat <- twinkle.dist[,,,2]
volume.permute <- permutation.test(dist.mat, it)
ggplot(volume.permute$permute.f, aes(x = volume.permute$permute.f[,1])) +
  geom_histogram(bins = 35) +
  geom_vline(aes(xintercept=volume.permute$obs.f, color = "Observed"),
             linetype = "dashed", show.legend = TRUE, size = 1.5) +
  theme(legend.title = element_blank()) +
  labs(x = "F-Statistic")+
  ggtitle("Permutation Test - Volume")
```

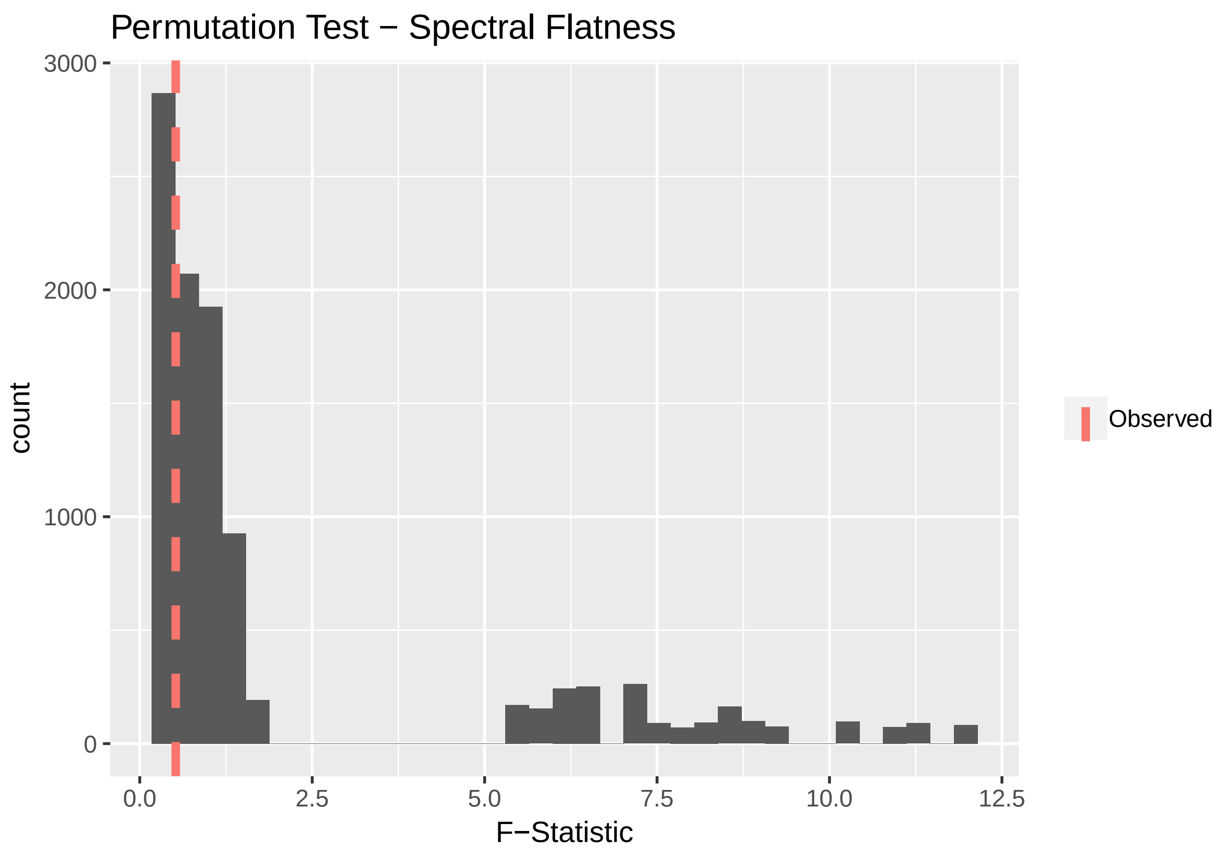


```r
print(volume.permute$p.val)
```

```
## [1] 0.1157
```

4.3 Spectral Flatness - Total

```r
dist.mat <- twinkle.dist[,,,3]
sf.permute <- permutation.test(dist.mat, it)
ggplot(sf.permute$permute.f, aes(x = sf.permute$permute.f[,1])) +
  geom_histogram(bins = 35) +
  geom_vline(aes(xintercept=sf.permute$obs.f, color = "Observed"),
             linetype = "dashed", show.legend = TRUE, size = 1.5) +
  theme(legend.title = element_blank()) +
  labs(x = "F-Statistic")+
  ggtitle("Permutation Test - Spectral Flatness")
```



```
print(sf.permute$p.val)
```

```
## [1] 0.7082
```

4.4 Balance - Volume by Frequency

```
spec.p.list <- list()
for(sl in 1:length(freq.start)){
  dist.mat <- twinkle.dist[,,,3+sl]
  spec.permute <- permutation.test(dist.mat, it)
  ggplot(spec.permute$permute.f, aes(x = spec.permute$permute.f[,1])) +
    geom_histogram(bins = 35) +
    geom_vline(aes(xintercept=spec.permute$obs.f, color = "Observed"),
               linetype = "dashed", show.legend = TRUE, size = 1.5) +
    theme(legend.title = element_blank()) +
    labs(x = "F-Statistic")+
    ggtitle("Permutation Test - Volume by Frequency")
  spec.p.list[sl] <- spec.permute$p.val
}
print(spec.p.list)
```

```
## [[1]]
## [1] 0.4088
##
## [[2]]
## [1] 0.3055
```

```
##
## [[3]]
## [1] 0.3506
##
## [[4]]
## [1] 0.1346
##
## [[5]]
## [1] 0.1531
##
## [[6]]
## [1] 0.0952
##
## [[7]]
## [1] 0.1202
##
## [[8]]
## [1] 0.0903
##
## [[9]]
## [1] 0.0861
##
## [[10]]
## [1] 0.4298
```

4.5 Timbre: SF by Frequency

```
sf.p.list <- list()
for(sl in 1:length(freq.start)){
  dist.mat <- twinkle.dist[,,,3+length(freq.start)+sl]
  spec.permute <- permutation.test(dist.mat, it)
  ggplot(spec.permute$permute.f, aes(x = spec.permute$permute.f[,1])) +
    geom_histogram(bins = 35) +
    geom_vline(aes(xintercept=spec.permute$obs.f, color = "Observed"),
               linetype = "dashed", show.legend = TRUE, size = 1.5) +
    #geom_vline(aes(xintercept=max(tempo.permute$permute.f), color = "Max-H0"),
    #           linetype = "dashed", show.legend = TRUE, size = 1.5) +
    theme(legend.title = element_blank()) +
    labs(x = "F-Statistic")+
    ggtitle("Permutation Test - Spectral Flatness by Frequency")
  sf.p.list[sl] <- spec.permute$p.val
}
print(sf.p.list)
```

```
## [[1]]
## [1] 0.7864
##
## [[2]]
## [1] 0.6159
##
## [[3]]
## [1] 0.7998
##
```

```
## [[4]]
## [1] 0.6228
##
## [[5]]
## [1] 0.5988
##
## [[6]]
## [1] 0.5422
##
## [[7]]
## [1] 0.7183
##
## [[8]]
## [1] 0.2925
##
## [[9]]
## [1] 0.1985
##
## [[10]]
## [1] 0.613
```

4.6 Timbre - MFCC (Euclidean Distance)

```
mfcc.p.list <- list()
for(sl in 1:(nMFCC - 1)){
  dist.mat <- twinkle.dist[,,,3 + 2*length(freq.start) + sl]
  spec.permute <- permutation.test(dist.mat, it)
  ggplot(spec.permute$permute.f, aes(x = spec.permute$permute.f[,1])) +
    geom_histogram(bins = 35) +
    geom_vline(aes(xintercept=spec.permute$obs.f, color = "Observed"),
               linetype = "dashed", show.legend = TRUE, size = 1.5) +
    theme(legend.title = element_blank()) +
    labs(x = "F-Statistic")+
    ggtitle("Permutation Test - MFCC")
  mfcc.p.list[sl] <- spec.permute$p.val
}
print(mfcc.p.list)
```

```
## [[1]]
## [1] 0.1472
##
## [[2]]
## [1] 0.2286
##
## [[3]]
## [1] 0.1298
##
## [[4]]
## [1] 0.2316
##
## [[5]]
## [1] 0.6651
##
```

```
## [[6]]
## [1] 0.7931
##
## [[7]]
## [1] 0.9823
##
## [[8]]
## [1] 0.3151
##
## [[9]]
## [1] 0.3678
##
## [[10]]
## [1] 0.8957
##
## [[11]]
## [1] 0.1476
##
## [[12]]
## [1] 0.5914
```

5. EDA

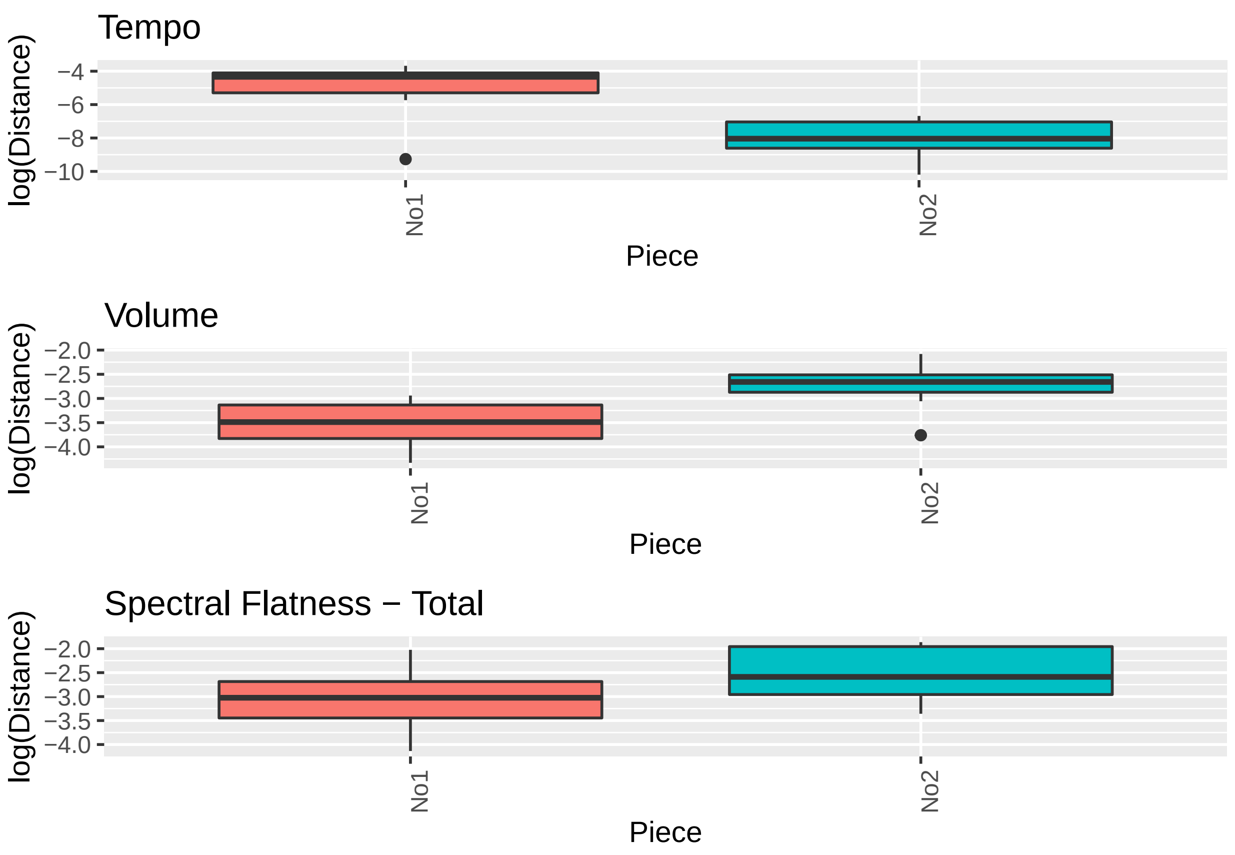
Distance values aggregated over pieces or orchestra pairs are plotted below. For the volume by frequency and spectral flatness by frequency, the plots are aggregated over all frequency groups and for the MFCC plots, the distance values are aggregated over all MFCC values.

```
nOrch <- 5 ## number of distinct "orchestras"
nPieces <-  2 # number of distinct "pieces"
JSD.t <- matrix(0, nrow = (nOrch*(nOrch-1))/2, ncol = nPieces)
JSD.v <- matrix(0, nrow = (nOrch*(nOrch-1))/2, ncol = nPieces)
JSD.SF <- matrix(0, nrow = (nOrch*(nOrch-1))/2, ncol = nPieces)
JSD.spec <- matrix(0, nrow = (nOrch*(nOrch-1))/2, ncol = nPieces)
JSD.SFbyF <- matrix(0, nrow = (nOrch*(nOrch-1))/2, ncol = nPieces)
euc.MFCC <- matrix(0, nrow = (nOrch*(nOrch-1))/2, ncol = nPieces)
for(r in 1:nPieces){
  ## Tempo
  curr <- twinkle.dist[,,r, 1]
  JSD.t[, r] <- curr[upper.tri(curr)]
  ## Volume
  curr <- twinkle.dist[,,r, 2]
  JSD.v[, r] <- curr[upper.tri(curr)]
  ## SF
  curr <- twinkle.dist[,,r, 3]
  JSD.SF[, r] <- curr[upper.tri(curr)]
}
JSD.t <- data.frame(JSD.t)
JSD.v <- data.frame(JSD.v)
JSD.SF <- data.frame(JSD.SF)
pieces <- c("No1", "No2")
orch.pairs <- c("Orch1-Orch2", "Orch1-Orch3", "Orch2-Orch3", "Orch1-Orch4",
                "Orch2-Orch4", "Orch3-Orch4", "Orch1-Orch5", "Orch2-Orch5",
                "Orch3-Orch5", "Orch4-Orch5")
```

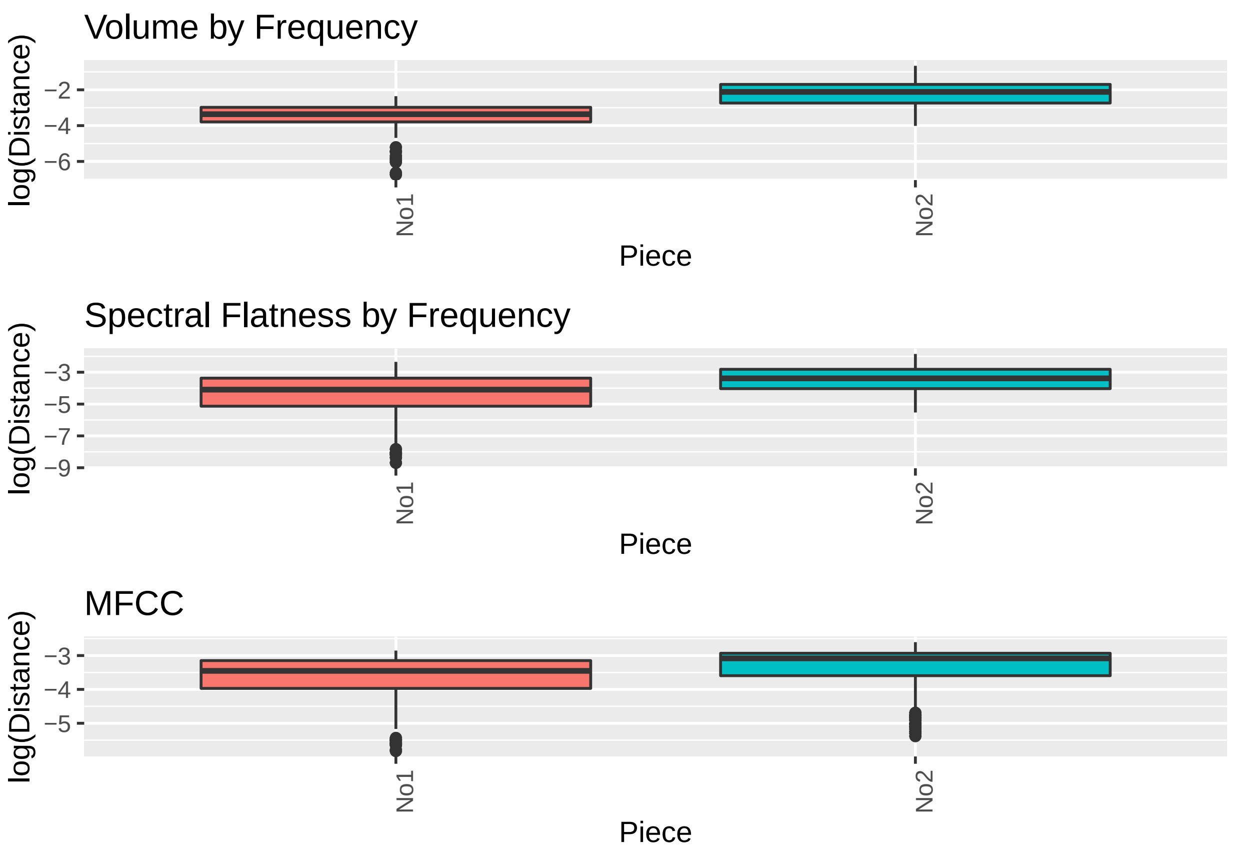
```r
colnames(JSD.t) <- pieces
JSD.t$Orchestra <- orch.pairs
colnames(JSD.v) <- pieces
JSD.v$Orchestra <- orch.pairs
colnames(JSD.SF) <- pieces
JSD.SF$Orchestra <- orch.pairs
```

```r
dat <- melt(JSD.t, id.vars = "Orchestra")
colnames(dat) <- c('Orchestra', "Piece", "Distance")


p1.piece <- ggplot(dat, aes(x = Piece, y = log(Distance), fill = Piece)) +
              geom_boxplot() +
              theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
              guides(fill=FALSE) +
              ggtitle("Tempo")

p1.orch <- ggplot(dat, aes(x = Orchestra, y = log(Distance), fill = Orchestra)) +
              geom_boxplot() +
              theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
              guides(fill=FALSE) +
              ggtitle("Tempo")

dat <- melt(JSD.v, id.vars = "Orchestra")
colnames(dat) <- c('Orchestra', "Piece", "Distance")


p2.piece <- ggplot(dat, aes(x = Piece, y = log(Distance), fill = Piece)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  guides(fill=FALSE) +
  ggtitle("Volume")

p2.orch <- ggplot(dat, aes(x = Orchestra, y = log(Distance), fill = Orchestra)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  guides(fill=FALSE) +
  ggtitle("Volume")



dat <- melt(JSD.SF, id.vars = "Orchestra")
colnames(dat) <- c('Orchestra', "Piece", "Distance")


p3.piece <- ggplot(dat, aes(x = Piece, y = log(Distance), fill = Piece)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  guides(fill=FALSE) +
  ggtitle("Spectral Flatness - Total")

p3.orch <- ggplot(dat, aes(x = Orchestra, y = log(Distance), fill = Orchestra)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  guides(fill=FALSE) +
  ggtitle("Spectral Flatness - Total")

spec.list <- list()
SFbyF.list <- list()
MFCC.list <- list()

for(i in 1:length(freq.start)){
  JSD.spec <- matrix(0, nrow = (nOrch*(nOrch-1))/2, ncol = nPieces)
  JSD.SFbyF <- matrix(0, nrow = (nOrch*(nOrch-1))/2, ncol = nPieces)

  for(r in 1:nPieces){
    ## Volume by Frequency
    curr <- twinkle.dist[,,r, 3+i]
    JSD.spec[, r] <- curr[upper.tri(curr)]

    ## SF by Frequency
    curr <- twinkle.dist[,,r, 3 + length(freq.start) + i]
    JSD.SFbyF[, r] <- curr[upper.tri(curr)]
  }
  JSD.spec <- data.frame(JSD.spec)
  colnames(JSD.spec) <- pieces
  JSD.spec$Orchestra <- orch.pairs
  dat <- melt(JSD.spec, id.vars = "Orchestra")
  colnames(dat) <- c('Orchestra', "Piece", "Distance")
  spec.list[[i]] <- dat

  JSD.SFbyF <- data.frame(JSD.SFbyF)
  colnames(JSD.SFbyF) <- pieces
  JSD.SFbyF$Orchestra <- orch.pairs
  dat <- melt(JSD.SFbyF, id.vars = "Orchestra")
  colnames(dat) <- c('Orchestra', "Piece", "Distance")
  SFbyF.list[[i]] <- dat
}


for(i in 1:(nMFCC-1)){
  JSD.MFCC <- matrix(0, nrow = (nOrch*(nOrch-1))/2, ncol = nPieces)

  for(r in 1:nPieces){
    ## MFCC
    curr <- twinkle.dist[,,r, 3 + 2*length(freq.start) + i]
    JSD.MFCC[, r] <- curr[upper.tri(curr)]

  }
  JSD.MFCC <- data.frame(JSD.MFCC)
  colnames(JSD.MFCC) <- pieces
  JSD.MFCC$Orchestra <- orch.pairs
  dat <- melt(JSD.MFCC, id.vars = "Orchestra")
  colnames(dat) <- c('Orchestra', "Piece", "Distance")
  MFCC.list[[i]] <- dat
}
```

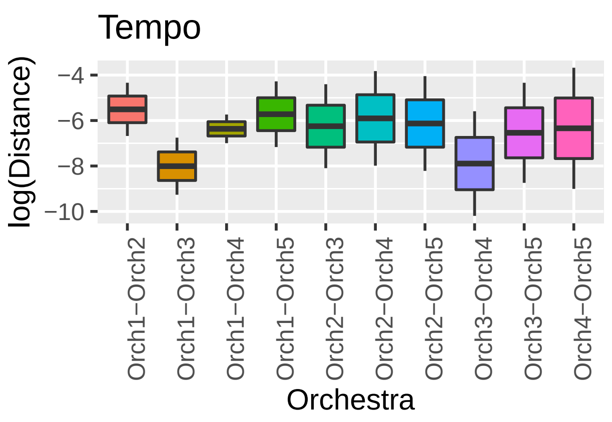
```r
MFCC.list <- rbind(MFCC.list[[1]], MFCC.list[[2]], MFCC.list[[3]], MFCC.list[[4]],
                   MFCC.list[[5]], MFCC.list[[6]], MFCC.list[[7]], MFCC.list[[8]],
                   MFCC.list[[9]], MFCC.list[[10]], MFCC.list[[11]], MFCC.list[[12]])

spec.list <- rbind(spec.list[[1]], spec.list[[2]], spec.list[[3]], spec.list[[4]],
                   spec.list[[5]], spec.list[[6]], spec.list[[7]], spec.list[[8]],
                   spec.list[[9]], spec.list[[10]])

SFbyF.list <- rbind(SFbyF.list[[1]], SFbyF.list[[2]], SFbyF.list[[3]], SFbyF.list[[4]],
                    SFbyF.list[[5]], SFbyF.list[[6]], SFbyF.list[[7]], SFbyF.list[[8]],
                    SFbyF.list[[9]], SFbyF.list[[10]])



p4.piece <- ggplot(spec.list, aes(x = Piece, y = log(Distance), fill = Piece)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  guides(fill=FALSE) +
  ggtitle("Volume by Frequency")

p4.orch <- ggplot(spec.list, aes(x = Orchestra, y = log(Distance), fill = Orchestra)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  guides(fill=FALSE) +
  ggtitle("Volume by Frequency")

p5.piece <- ggplot(SFbyF.list, aes(x = Piece, y = log(Distance), fill = Piece)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  guides(fill=FALSE) +
  ggtitle("Spectral Flatness by Frequency")

p5.orch <- ggplot(SFbyF.list, aes(x = Orchestra, y = log(Distance), fill = Orchestra)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  guides(fill=FALSE) +
  ggtitle("Spectral Flatness by Frequency")

p6.piece <- ggplot(MFCC.list, aes(x = Piece, y = log(Distance), fill = Piece)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  guides(fill=FALSE) +
  ggtitle("MFCC")

p6.orch <- ggplot(MFCC.list, aes(x = Orchestra, y = log(Distance), fill = Orchestra)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  guides(fill=FALSE) +
  ggtitle("MFCC")



grid.arrange(p1.piece, p2.piece, p3.piece, nrow = 3)
```

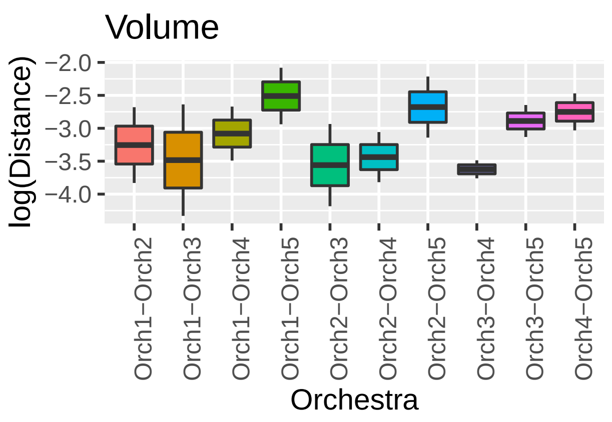


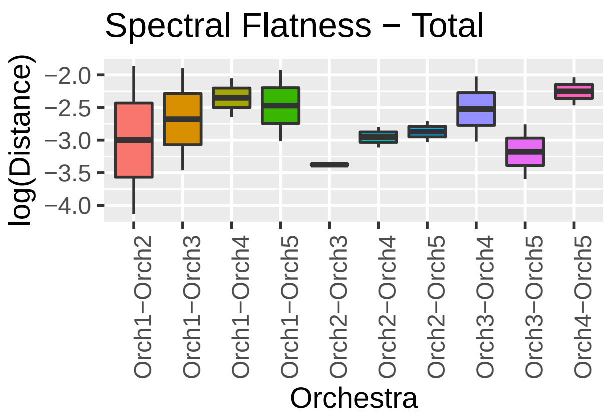
```r
grid.arrange(p4.piece, p5.piece, p6.piece, nrow = 3)
```



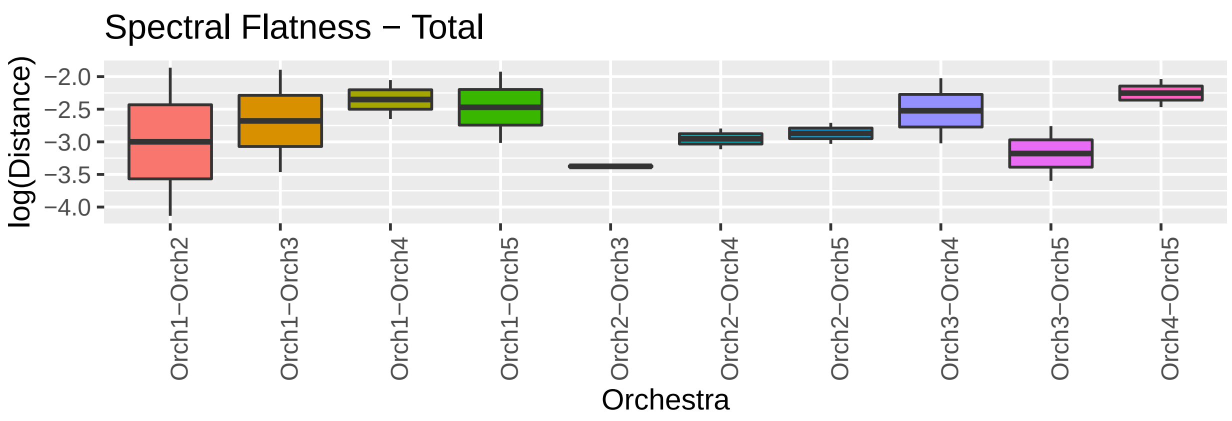
```r
grid.arrange(p1.orch, p2.orch, p3.orch, nrow = 2)
```

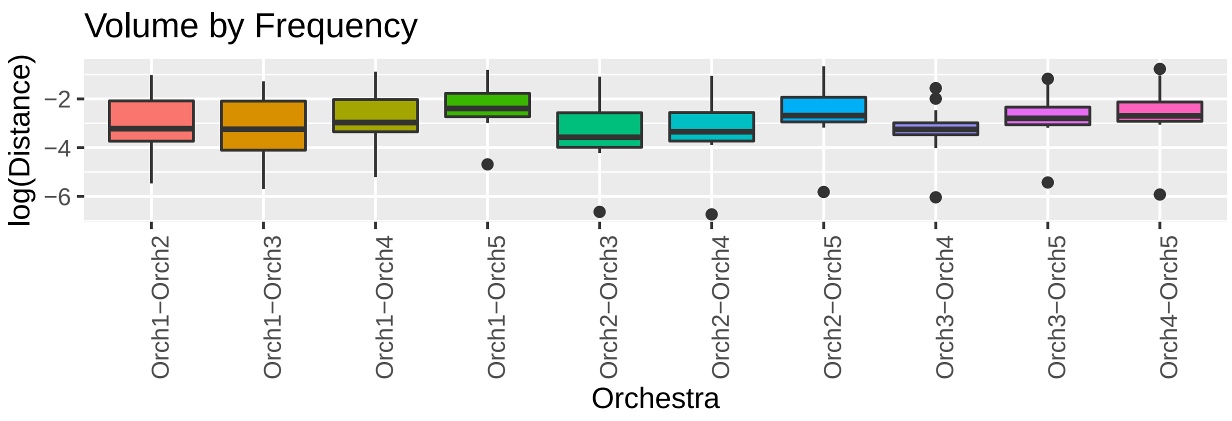




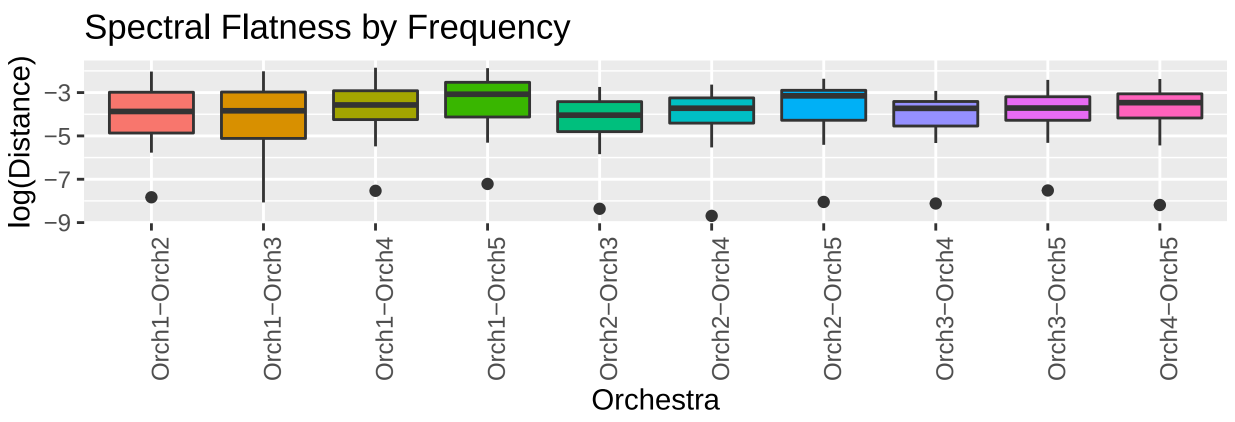


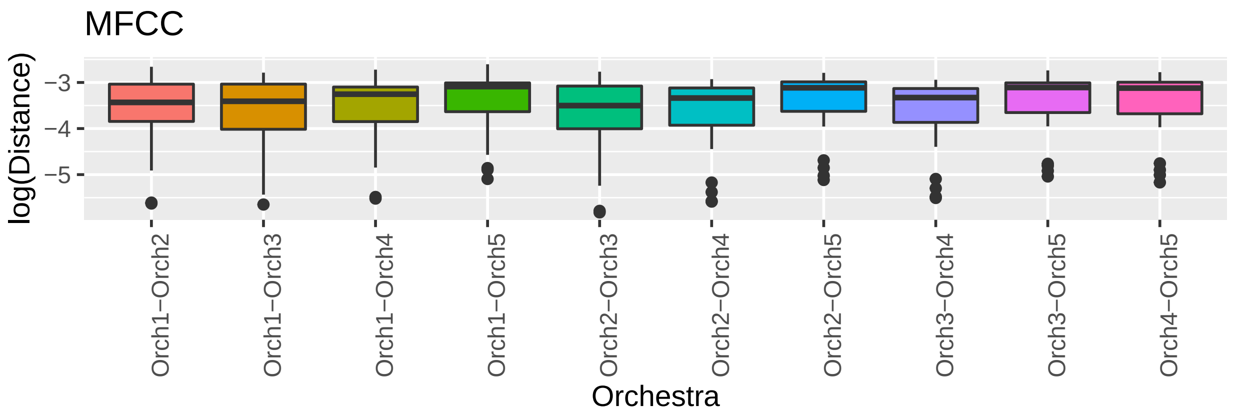
```
grid.arrange(p3.orch, p4.orch, nrow = 2)
```





```
grid.arrange(p5.orch, p6.orch, nrow = 2)
```


Spectral Flatness by Frequency
log(Distance)
−3
−5
−7
−9
Orch1−Orch2
Orch1−Orch3
Orch1−Orch4
Orch1−Orch5
Orch2−Orch3
Orch2−Orch4
Orch2−Orch5
Orch3−Orch4
Orch3−Orch5
Orch4−Orch5
Orchestra


MFCC
log(Distance)
−3
−4
−5
Orch1−Orch2
Orch1−Orch3
Orch1−Orch4
Orch1−Orch5
Orch2−Orch3
Orch2−Orch4
Orch2−Orch5
Orch3−Orch4
Orch3−Orch5
Orch4−Orch5
Orchestra