American University - DATA 312

Instructions for Homework 3-D = Project 3

Unit 3 – Music analysis
Part D – Timing and rhythm

Pre-requisite work:
Please complete Homework 3-C before starting this assignment. You will need the same four MIDI files and one WAV file from the previous worksheets, but will not need any new R libraries. That is, the R libraries you need are:

- tidyverse
- tabr
- audio
- GENEAread

Objectives:

The analysis done in the previous three worksheets was based upon elementary statistical ideas. This worksheet introduces a fancier technique called *principal components analysis* (PCA). While we won't go into the details of how PCA works mathematically, its objective is relatively simple. Starting with a "tidy" table of numerical observations (each row being an observation and each column being a variable), a convenient visualization is a scatter plot using two (or maybe three) columns to supply the x, y, (and possibly z) coordinates of each point.

What do you do if you have more than three columns? Unless you have really practiced, visualizing data in more than three dimensions is really, really hard! (Never mind the fact that plotting data with more than two dimensions on a flat display is a bit of an issue...) You might hope for some tactic to "reduce the number of dimensions (columns)". Certainly one possible solution is to make several scatter plots, using different selections of columns for each. But which of these selections is most informative? PCA attempts to reduce the dimensions by "combining" your variables (columns) to give you the best scatter plot representation of your data. If you're lucky, most of the variance of your data can be explained by one or two *composite variables* (weighted sums of columns). Then you just need to make a scatter plot with just those variables to get a good summary of your data.

On the other hand, PCA may tell you that you need more variables than you can display... in that case you'll need a fancier dimension reduction algorithm. This is what machine learning is good for, and we'll get to that later in the course!

This particular worksheet gets these large tidy tables of numbers from music by grouping adjacent musical events: overlapping samples in WAV or consecutive blocks of MIDI events. In the case of the WAV data, the table is **very** wide and has hundreds of columns. As we'll find, PCA manages to reduce these down to just two, at least in the recording we're using. Once we've done that, it's not too hard to pick out individual notes from recording.

In the case of the MIDI events, what we'll see is a bit more abstract. Although the Bach piece and the improvisational piano pieces were both recorded from a musician's playing an instrument, the Bach piece was "cleaned up" quite a bit. PCA gives us a little insight into how this cleanup was done.

This same technique was used by Prof. Robinson and two coauthors a few years ago to analyze the structure of improvisational Cuban drumming. Check it out:

Fernando Benadon, Andrew McGraw, and Michael Robinson, "Quantitative Analysis of Temporal Structure in Cuban Guaguanco Drumming," *Music and Science*, Volume 1, July 4, 2018.

Open access link: https://doi.org/10.1177/2059204318782642

Work Process:

1. As usual, start by clearing the workspace and loading libraries:

```
rm(list=ls())
gc()
library(tidyverse)
library(tabr)
library(audio)
library(GENEAread)
```

Note that PCA is actually in the stats library, which is part of base R. You don't need to load it manually; it's already loaded!

2. Let's start out by playing with PCA on some data that we fully understand before we try to use it on data we are less certain about. I hope you recall from high school trigonometry that the coordinates of a point on the unit circle is given by x = cos(t), y = sin(t), where t is the angle between the x axis and the line going through the origin and the point on the circle. (Note that I'm assuming t is in radians.) If you don't remember this, that's OK, because the computer will show it to you!

Let's get a bunch of points on that circle! An easy way to do that is to make a table with three columns: t, x, and y using the two formulas in the previous paragraph:

```
tab1 <- tibble(t=seq(0,2*pi,by=0.1),   # Make a `t` column from 0 to 6.28... in steps of 0.1
  x=cos(t),    # The formula for the `x` column, in terms of the `t` column
  y=sin(t))    # The formula for the `y` column, in terms of the `t` column
```

3. It's easy to check that this is indeed a circle by having R make a scatter plot for you:

```
tab1 %>% ggplot(aes(x,y)) + geom_point()
```

This should look like a circle!

4. If you wanted to connect the dots, you can try to use geom_line() instead of geom_point(), but that doesn't look like a circle. The reason is that geom_line() sorts the points in order of their `x` coordinate. Since the circle consists of an upper and a lower part, the resulting plot jumps between the upper and lower parts of the circle. Really what you want is geom_path() which uses the points in the order they're given in the table. So if you plot

```
tab1 %>% ggplot(aes(x=x,y=y)) + geom_point() + geom_path()
```

you'll get a nice looking circle. The last little bit of the circle isn't closed up fully, but that's OK.

5. I had just given you tab1 without labeling the columns in any reasonable way, you wouldn't necessarily know to pick aes(x=x,y=y) for your scatter plot. You might try

```
tab1 %>% ggplot() + geom_point(aes(x=t,y=x)) + geom_path(aes(x=t,y=y))
```

which probably reminds you of trigonometry!

6. But the fact that in Step 4 we don't get a circle means that some of these scatter plots are more or less useful for explaining what the data are telling us. PCA is a quick way to figure this out, but the circle is a bit too simplistic. Let's fancy it up a bit with some more formulas:

```
tab2 <- tibble(t=seq(0,2*pi+0.01,by=0.01),
  x1=cos(2*t)/2,
  y1=sin(t),
  x2=cos(3*t)/3,
  y2=sin(2*t)/2,
  x3=cos(t),
  y3=sin(3*t)/3) %>%
  select(-t)  # That's a "minus" `t`; this gets rid of the `t` column... I don't want it in our data for
              # this exercise
```

This describes a twisty-curvy curve that "lives in" 6 dimensions (that is, it has 6 columns).

7. What does the tab2 data "look like"? Well, it's kind of hard to figure out! Of course, you can still make scatter plots:

```
tab2 %>% ggplot(aes(x=x2,y=y2)) + geom_path()
```

The resulting knotted curve is called a "Lissajous figure" (pronounced "Lee-suh-joo"), and is sometimes used to help tune musical instruments (or old radios). Notice in particular that the axes are scaled a bit differently.

8. Although PCA is a rather elaborate mathematical algorithm, base R knows how to do it for you. It's easy:

```
pca_tab2 <- prcomp(tab2)
```

Note: if you search for how to do PCA in R, you will also find that princomp() does PCA as well. The truth is that princomp() is not quite the same as prcomp(). (Technically, prcomp() uses *singular value decomposition*, while princomp() uses *eigenvalue decomposition*. This means that prcomp() handles noise in the data a little better. The difference is usually not very noticeable.)

9. After the prcomp() runs, pca_tab2$x contains a "combined" copy of our data. The columns of pca_tab2$x are weighted sums of the columns of tab2, sorted in decreasing order of variance. You can think of this as sorting the columns in decreasing order of importance. The columns of pca_tab2$x are labeled `PC1`, `PC2`, `PC3`, ... Therefore, the PCA'ed scatter plot is easy to get:

```
pca_tab2$x %>%
  as_tibble() %>% # Note: pca_tab2$x isn't tidyverse-compatible, so we need to fix here...
  ggplot(aes(x=PC1,y=PC2)) + geom_path()
```

Looks pretty circle-like to me! What that is saying is that the data we have made mostly follows the path of a circle if you rotate it correctly (in 6 dimensions!).

10. Now you are right if you're a little skeptical of Step 8. We can get an idea of how reliable the picture we made in Step 8 is by plotting what's called a *Scree plot*. That is graph of the standard deviations of each column in pca_tab2$x. Helpfully, prcomp() already put the results of that computation in pca_tab2$sdev for us, so the Scree plot is easy to get:

```
pca_tab2$sdev %>%
  as_tibble() %>% # Again, make the data tidyverse-compatible
  mutate(PC=row_number()) %>% # Label the entries of pca_tab2$sdev
  ggplot(aes(x=PC,y=value)) + geom_col() # plot!
```

What you should see is that the first two bars on the left are big compared to the rest, which get progressively smaller. If that's how the Scree plot looks for your data, then this means that the first two columns of pca_tab2$x are a reliable summary of the data. If there are more large bars in the Scree plot, then this means you need more dimensions in your scatter plot to get a good summary of the data. There is a bit of human judgment that is necessary to determine "how big" or "how many" subsequent bars in the Scree plot that you can tolerate without getting an unreliable scatter plot, but that comes with some practice. (In this worksheet, everything works well enough.)

11. We'll start music analysis in this worksheet with the WAV file this time, rather than the MIDI. Let's start off just the same way we did in HW3C:

```
wavfile <- load.wave('lightly_row_violin.wav')
wav<-tibble(sample=wavfile)%>%mutate(time=row_number()/wavfile$rate)
wav_stft <- stft(wav$sample,
                 freq=wavfile$rate,
                 win=0.5,
                 inc=0.05)
```

12. If you View the output of the STFT, namely

```
View(wav_stft$values)
```

you'll see that there are many columns! Way too many to visualize via scatter plots! But prcomp() is not bothered by this, and will run reasonably quickly:

```
pca_wav <- prcomp(wav_stft$values)
```

13. Let's make a Scree plot, just like we did in Step 9:

```
pca_wav$sdev %>%
  as_tibble() %>%
  mutate(PC=row_number()) %>%
  filter(PC<10) %>%      # Seriously.  Who can visualize 10 dimensions, anyway? Cut it off!
  ggplot(aes(x=PC,y=value)) + geom_col()
```

OK, looks like there is just one big bar and many smaller ones. That means that the scatter plot of the first two columns should be a reliable summary of the data.

14. The PCA scatter plot is easy to make, and is just like Step 8:

```
pca_wav$x %>%
  as_tibble() %>%
  ggplot(aes(x=PC1,y=PC2)) + geom_point()
```

Each dot in the scatter plot represents a particular instant in the recording, along with the sounds recorded shortly after that instant. Specifically, each dot corresponds to a snippet of sound 0.5 seconds long, starting at some specific time within the longer recording. Since the STFT we computed in Step 10 used a window increment `inc` = 0.05 seconds, these snippets start every 0.05 seconds during the recording, and therefore often overlap quite a bit.

Because of all that, it's helpful to mark the times on the plot using colors

```
pca_wav$x %>%
  as_tibble() %>%
  mutate(time=wav_stft$times) %>%
  ggplot(aes(x=PC1,y=PC2,color=time)) + geom_point()
```

The smoothly varying colors, radiating out from the center point indicates that neighboring snippets of sound are pretty similar. That's because they overlap!

15. If you look at the plot(s) created in Step 13, you'll notice that it looks a bit like a star with several "arms" radiating from a central point. If you're lucky, you should be able to see four or five distinct "arms". (The colors help to distinguish between two of the arms, since they overlap a bit.) Each of these "arms" correspond to distinct pitches being played by the violin. The central part, where all the "arms" come together, corresponds to periods of silence in the recording. You can check this idea by following the color gradient on the points! The points move in/out of that central region as time progresses.

This kind of plot isn't particularly perfect, because of the overlapping "arms". Part of that is because we've squashed the data from hundreds of columns down to two! If you were to make a 3d plot, using the first three columns, this would help a bit. But that's not the direction we'll go in this worksheet.

16. Let's see if we can transcribe a bit of the music using this PCA plot, since that will give us some confidence that the "arms" correspond to notes played by the violin. (As it happens, we won't get as good results as we did in HW3-C, but bear with me...) The first step is to get rid of the rests. This is pretty easy: we simply want to filter out all snippets of sound that are too close to the central point. If you remember the Pythagorean theorem, then the following kind of filter would make sense:

```
filter(sqrt(PC1^2+PC2^2)>1000) # Don't run this!
```

where the "^" represents exponents, and sqrt() is the square root operator. Putting this in line with the plot:

```
pca_wav$x %>%
  as_tibble() %>%
  mutate(time=wav_stft$times) %>%
```

```
  filter(sqrt(PC1^2+PC2^2)>1000) %>%
  ggplot(aes(x=PC1,y=PC2,color=time)) + geom_point()
```

should break off those five "arms" into five little clusters of points.

17. Now, we'd like to label the snippets of sound (that are not rests) based on those clusters we see in Step 15. This is a machine learning task, but a low-tech version called *k-means clustering* works well enough for our purposes. K-means clustering takes a table of data and labels the rows based on which cluster it thinks each row goes with. You have to tell the k-means clustering algorithm how many clusters your data has (that's the *k*) but it does the rest.

Helpfully, kmeans() is base R, so it's already loaded and ready for you. (Usually k-means clustering requires a bit of tweaking to get it "just right", so feel free to repeat this Step until you get a good result. That's why it's "low-tech machine learning"...) Basically the data pipeline is the same as in Step 15:

```
pca_kmeans <-pca_wav$x %>%
  as_tibble() %>%
  mutate(time=wav_stft$times) %>%
  filter(sqrt(PC1^2+PC2^2)>1000) %>%
  kmeans(5)  # the number 5 is the number of clusters we want
```

18. The output of kmeans() is not tidyverse-compatible (even though the input was...), and that lives in pca_kmeans$cluster. This is a list of cluster numbers, one for each row in the input table. We'd like these to be in a new column of our data, labeled `cluster`. Since we filtered the input table (to get rid of the rests, see Step 15), we have to filter the data in the same way so that everything lines up nicely. Then adding the clusters using mutate() is no problem:

```
pca_wav_clustered <- pca_wav$x %>%
  as_tibble() %>%
  mutate(time=wav_stft$times)  %>%
  filter(sqrt(PC1^2+PC2^2)>1000) %>%
  mutate(cluster=pca_kmeans$cluster)
```

Plotting is then just what you should expect:

```
pca_wav_clustered %>%
  ggplot(aes(x=PC1,y=PC2,color=cluster)) + geom_point()
```

You should see each of the five clusters (at the tips of the arms) colored differently. If you do not see this, this is not your fault, it's just that k-means didn't work out so well. Since k-means has an element of randomness, just rerun Steps 16 and 17 repeatedly until each cluster shows up with a different color. (This shouldn't take more than two or three tries!)

19. Finally, plot a timeseries of the clusters (time versus cluster number). This is effectively a transcription of the music, but with a (big) caveat. That caveat is that there is no guarantee that the cluster IDs come in order of note frequency or pitch. Bearing that in mind, try

```
pca_wav_clustered %>%  ggplot() + geom_point(aes(x=time,y=cluster))
```

Well, this plot should show you that the clusters do seem note-like, in that they are contiguous in time, and more-or-less match the tempo of the music in the recording. You can compare this plot with the last few Steps in HW3C; they're quite similar!

Shifting topic now to MIDI, let's start with the same time blocking analysis we did in HW3C; that's Steps 20-24 below. Please refer to HW3C if you forgot the details of what these are doing!

20. Load the MIDI files

```
bach<-read_midi('bach_846_format0.mid')
edwin_improv<-read_midi('improv.mid')
edwin_improv2<-read_midi('improv2.mid')
duodecatonic<-read_midi('justin_rubin_lyric.mid')
```

21. Glue together everything

```
songs<-bind_rows(mutate(bach,name='bach'),
            mutate(edwin_improv,name='edwin_improv'),
            mutate(edwin_improv2,name='edwin_improv2'),
            mutate(duodecatonic,name='duodecatonic'))
```

22. Pick out just the "Note On" events

```
songs_notes <- songs %>% filter(!is.na(pitch)) %>%
  mutate(as_music_df(pitch),
       end_time=time+length)
```

23. Get the length of each song

```
song_lengths <- songs_notes %>%
  group_by(name) %>%
  summarize(total_time=max(end_time))
```

24. Collect the notes into blocks:

```
num_time_blocks<-25
songs_notes <- songs_notes %>%
  inner_join(song_lengths) %>%
  mutate(time_block=floor(num_time_blocks*time/total_time))
```

25. Well, that was a bit of wrangling! But now, we're ready to get the data into the right form for PCA. We can do several possible experiments with the data, looking through the lens of PCA. This worksheet will show you one of them, since it's closest to what was done in the Cuban drumming paper mentioned in the Objectives section.

What we do is group the notes by time_block, and then using the earliest note in each group as a reference, we measure the elapsed time to each of the other notes in the group. This gives a model of the rhythm of the music. Music theorists call these elapsed times "inter onset intervals (IOIs)". If the rhythm is stable over time, then the elapsed times recorded in the groups will be pretty similar across groups. If the rhythm isn't stable, or there are several different variants of the rhythm, then we'll see that.

Ultimately, what we want is a table in which each row corresponds to a time_block, and the columns (left to right) are the elapsed times from each note to the first one. This means that the first column is all zeros, but the rest aren't. It's also best if we sort the note times so that they increase as you move left-to-right in each row.

Getting to that table takes quite a few steps!

Let's start by grouping by the time_block, and then sorting the notes within each block based on their times:

```
note_starts1 <- songs_notes %>%
  group_by(name,time_block) %>%
  arrange(time,.by_group=TRUE)
```

26. We next want to add a column: `num_within_block` to keep track of which order the notes come in each block, and we want to modify the time column to be the elapsed time on each given to the first note in the block. That's done by two mutates:

```
note_starts2 <- note_starts1 %>%
  mutate(num_within_block=row_number()) %>%
  mutate(time=time-min(time))
```

27. Let's focus our attention on just the columns we'll need going forward:

```
note_starts3 <- note_starts2 %>% select(name,time,time_block,num_within_block)
```

28. Now for the tricky part. Right now, each note is a separate row. We want each row to be a time block instead, so this is a pivot operation. Since we're adding columns, it's a pivot_wider(). What are the columns? They should be corresponding to the order of the notes in the block, which is what `num_within_block` tells us.

```
note_starts4 <- note_starts3 %>%
  pivot_wider(names_from=num_within_block,values_from=time) %>%
  ungroup()  # Since we are now done with the time_block groups... they're now rows!
```

Take a moment to

```
View(note_starts4)
```

You should see that the first few columns are pretty well filled-out, but later there are some NAs. Unfortunately, prcomp() doesn't like NAs at all, and it also chokes on the column of all 0s. Although there are better ways to proceed, a quick thing to try is simply to pick out the good columns and discard the rest:

```
note_starts <- note_starts4 %>% select(name,`2`:`5`)
```

29. Take another moment to compare the following two tables:

```
View(note_starts %>% filter(name=='bach'))
```

and

```
View(note_starts %>% filter(name=='edwin_improv'))
```

You'll notice that the IOIs for the Bach piece are very rigidly stable, while the improvisational piece has some considerable variability.

30. Now, we're ready for some PCA! Continuing on the subject of prcomp() grumpiness, it doesn't like categorical variables, so once we select a piece of music to analyze, we'll need to remove the name column for it.

```
pca_note_starts_bach <- note_starts %>%
  filter(name == 'bach') %>%
  select(-name ) %>%
  prcomp()
```

31. Rather than doing a straight-up scatterplot with geom_point(), it's handy to use geom_count() instead, because this will make a larger dot if there are duplicated points.

```
pca_note_starts_bach$x %>%
  as_tibble() %>%
  ggplot(aes(x=PC1,y=PC2)) + geom_count()
```

Most of Bach's notes are very regularly spaced, though there are some exceptions.

32. Rerun Steps 30-31 with the other pieces, and take note of the differences! What you should find is that the two improvisational pieces have some clustering in their IOI patterns, but not a lot. I suspect the rigidity of the Bach piece is actually artificial; it was probably recorded and then the note timings were "rounded to the nearest 1/16 note" or something like that. That would explain why there are some outliers as well.

33. You can rerun the analysis we did with IOIs for note lengths, frequencies, or velocities. How? You'll need to change Steps 26, 27, and 28 to swap out the references to `time` with the other attribute you want to study. Here's the big-long-pipeline for `length`, so you can see what changed (marked in bold):

```
note_lengths <- songs_notes %>%
  group_by(name,time_block) %>%
  arrange(time,.by_group=TRUE) %>%
  mutate(num_within_block=row_number()) %>%
# Note: the mutate() step to modify the times was simply unnecessary and so was removed
  select(name,length,time_block,num_within_block) %>%
  pivot_wider(names_from=num_within_block,values_from=length) %>%
  ungroup() %>%
  select(name,`2`:`5`)
```

Then from there on out, the PCA process is the same!

Assignment:
For submission, include your .Rmd file as well as your .pdf file. Please also include any data files (MIDI or WAV files).

The Project has two options: MIDI or WAV. Select one to submit. You may do both if you like, but only one is required. You may use the same files as in previous assignments in this module, as this Project is a summary of your work on those assignments.

The first part of this Project is simply to **explain** your findings from HW3C. You get to reuse all that code! The instructions are repeated here for your convenience. To that good starting point, you'll simply add some PCA analysis like was done in this worksheet.

MIDI Option:

1. Like in HW3B, please legally obtain two MIDI files for your analysis, and be sure to include them in your upload to Canvas. The files must contain the following properties: (a) have more than 300 notes (b) contain at least two definite "movements", "verses", or other structural changes within them. Multiple channels or tracks are not necessary.
2. Show that your files satisfy property (a) by counting the notes in each file, and determine the length of each file in seconds.
3. Using the num_time_blocks idea discussed in Steps 7 in HW3C (and following), produce a plot showing the number of notes played over time. Show that your files satisfy property (b) by using a plot or a table.
4. Rerun your analysis using several other num_time_blocks values. Explain what changes you see in your results.
5. Show how the usage of notes changes over time in your files, like Step 15 in HW3C . You may have to make several pages of plots; please make sure to explain (in comments) what the key features are.
6. Does the tempo vary in your piece? Justify your answer using an analysis like Step 16 in HW3C.
7. Perform a PCA analysis of one or more of the following: IOIs, note lengths, note frequencies, or note velocities. Mimic Steps 25-31 of this worksheet. Explain why you picked the property you did: was it because the rhythm was pronounced? Or is there a particular note "motif" that is repeated? (Hint: a chorus/refrain will do that if you pick your num_time_blocks carefully.)
8. Summarize your findings! What interesting or unexpected things did you find as a result of your analysis?

WAV Option:

1. Legally obtain two WAV files of music for your analysis. and be sure to include them in your upload to Canvas. (Hint: Wikipedia has many musical WAV files that are legal to use.) The files must contain at least 10 seconds and not more than 2 minutes of music. The upper limit is for your sanity and mine! **It will be much easier if there is only one instrument heard in the file.** (This is not a requirement, though.)

2. Load the file into R, and show a time series plot of the file. Can you see individual notes?
3. Produce a frequency series for the file, zooming in on the most interesting part of the file. Sometimes you will see "harmonics", which are spikes occurring at many multiples of the actual note frequency. Do you see any harmonics?
4. Using the frequency series for the file, what do you think the most common (or possibly important) notes are in the piece? If there are harmonics, you may have difficulty choosing between multiple copies of a note in different octaves. Make a note of this if this is happening.
5. Produce an STFT spectrogram plot for the file. Narrow the frequency range to cover the notes actually being played, or if you can't, please explain why! (Hint: you might have trouble if there are many harmonics)
6. Attempt to read off the first few notes of the song, much like was done in Step 30 in HW3C.
7. Fixing the problems caused by harmonics is one of the reasons to do the PCA analysis in this worksheet. To that end, run a PCA analysis on the spectrogram like was done in Steps 11-15 of this worksheet. Can you see "arms" and/or "rests"?
8. Finally, compare the transcription you obtained from Step 30 in HW3C (Step 6 above) with a "transcription" from the PCA plot like Steps 16-19 in this worksheet. Do you see different features? If so, what are they? Which transcription worked better? Where do they have trouble?