An Analysis of Chord Usage in Select Classical Composers.

Rowan Shigeno, Redd Tan, Holden Starkey

Motivation.

Our aim is to provide an analysis of classical music from a handful of composers from a theory perspective, looking specifically at aspects of harmony (chords) and tonality (roman numerals). We intend to use encoded scores to find patterns in style, also taking note of and comparing areas of dissonance and usage of unique or special chords. To begin, we listen to a few individual pieces to get the hang of what we are trying to analyze— including John Adams' *Short Ride in A Fast Machine* (a piece from the 21st century), two of our own compositions, and classical pieces by various well-known composers. Following this, we take collections of pieces from famous composers from the Baroque (Bach), Romantic (Chopin and Grieg), and Impressionist periods (Debussy) using MIDI files. These four composers provide the data that we analyze in what follows.

- Bach (1685-1750) – *Well-tempered clavier, Book I: Prelude No. 1 - 24.* Highly regulated form, use of symmetric arrangements, contrapuntal / considered a "master of counterpoint". His music is often described as being "mathematical" or "pure," with progressions being very "**functional**." (sources: Penn article, Brown article.) This article provides some interesting analysis / networks of Bach's pieces. In particular, researchers used a measure called **entropy**, or the level of variation in **note sequences** in the networks and degree of clustering, (such that networks with higher entropy have nodes that link to many more other nodes) to find that "Bach's chorales have much lower entropy than his toccatas, pointing to a difference not just in style—chorales are **simple and repetitive**, while the toccatas have complex, chromatic passages."
- Chopin (1810-1849) — *24 Preludes.* Freer form, thicker texture, emotional depth. "Chopin was known for his highly expressive melodies, which often featured intricate ornamentation and leaps across wide intervals" (see this website). Chopin often uses root position chords such as the E minor chord to create a sense of stability.
- Grieg (1843-1907) — 12 "quality live recordings" selected by Katsuhiro Oguri (playwright), plus *Anitra's Dance*; as well as roughly the same number of "piano rolls converted" in a separate collection. Grieg is also famous for *Peer Gynt* and *In the Hall of the Mountain King*. Influenced by classical and Norwegian folk music. Praised for his wide range of styles and orchestral effects. His use of **parallelism and chromatic melodies** that followed rules of **functional harmony** were qualities associated with his music (Juilliard).
- Debussy (1862-1918) — *24 Preludes*. Shift from tonal to atonal music, prevalent dissonance and NCTs, emphasize on sound color atmosphere. **Nonfunctional**, unlike Bach: music is allowed to flow. (Source.)

- John Adams (1947-present) — *Short Ride in a Fast Machine*. Emphasis on percussion, with frequent dissonance occurring and some unusual chords. Significant focus on tonality as opposed to harmony.

Additional theory background (source here)**.** The theory of modern tonal **harmony** originated around the 16th century, and developed slowly over the next couple hundred years. While in earlier times counterpoint two-part writing had been seen as fundamental, chords were later regarded as a key component (rather than a byproduct) regardless of the difference between homophonic (multiple parts move together with a single prominent melody) and polyphonic (multiple independent melodic lines playing simultaneously) style. Bach's usage of counterpoint, in particular, had a strong basis in harmony.

The term "**tonality,**" in contrast, is used primarily to describe the "key" of a piece, but can also refer to which chords contain or do not contain notes considered foreign to the scale— provided that those chords, in context, do not signify a change in key.

Prior to 1600, a contrapuntal **dissonance** in voices was regarded as what necessarily came before a resolution of the harmony. The notion that a chord presented as an indivisible unit would in the 17th century result in new approaches surrounding the construction of dissonance: in particular, the border between dissonant chords, and notes "foreign" to the chord, would be significantly blurred beginning the 17th century— from which point the term "dissonance" was completely redefined. What *caused* the dissonance would come to be thought of as not so much a note, as a combination of notes. In other words, while "**traditional dissonance**" had been thought of and implemented as what necessarily comes before a resolution, "**modern dissonance**" occurs when notes in a chord 'argue' or 'clash' with each other, but do not necessarily need to be resolved back to the tonal key from there. It is critical that we take these two types of dissonance into account, and distinguish between them.

Implementation.

1. Our original plan had been to use all-XML files, but this had its own limitations— in particular, requiring payment for unrestrained access, as well as cumbersome downloading process. We were able to obtain MIDI files without trouble, however, as most of the pieces we choose are currently in the public domain. So, while we were able to download music XML files for Chopin's *24 Preludes* from musescore website, we shifted to kunstderfuge and downloaded MIDI files for all of the Bach, Debussy and Grieg pieces. We had additionally planned to work with one piece each from Redd, Rowan, and John Adams, but ended up not doing so, as our primary goal shifted to comparison of these four prominent composers.
2. Once we had all the files we required, we placed them all in a folder in Google Drive, using Google Collab to write the code. The Collab file could only be run by the individual whose Drive was "mounted" to the file, so the other two of us had to create our own copies of the files to test code.

3. Through Music21 and Professor Freedman's assistance, we were able to extract chords from each of the pieces using the 'Chordify' module, placing each of the composer's set of chords into its own DataFrame. We then examined the chords while looking at scores and listening to pieces, to confirm that the chords seemed to line up.
4. Since chords were not 100% accurate, we also attempted to 'quantize' the music to get more consistent chord data— as it seemed Music21 was counting chords that occurred on the off-beats, which we did not want. However, this was quite unsuccessful despite several attempts, primarily due to poor documentation of Music21 and other unwanted consequences (such as deletion of notes) in the resulting files.
5. Subsequent to this, we examined the list of unique chords for each composer. As these lists were very long, we went through each composer's list and came up with a comprehensive list of recurring chords (containing about ~115 chords)— then used this list to filter the chords for each composer to eliminate unwanted ones. Upon examining the resulting value counts of chords for each composer, however, we realized that our list was still far too long to get valuable insights into the chord usage and be able to compare composers. Thus, we created an ever smaller list comprising only of the 14 most common chord types, effectively reducing different variations of any single chord to that chord.
6. A similar process was applied to roman numerals. Initially, we had had a list of ~30 roman numerals (primarily just eliminating inversions and those with excessive accidentals), but decided to filter further to the 10 most commonly used roman numerals— so that it would be easier to compare the usage of roman numerals across composers.
7. Using the value counts of the 14 chord and 10 roman numerals for each of the four composers, we created bar charts to represent the distributions of chord and numeral usage for each composer. (For example, 12-20% of chords used by any given composer were major triads.)
8. We proceeded to then create a map of "chord eccentricity" vs "chord versatility" for each composer (as points in the xy-plane) via the following steps:
    a. For each chord type, take the average of the proportion of that chord type used for each composer. Then for each composer, calculate the difference between their proportion and the average one; do that for all chord types and take the mean of that. This is a measure of how "eccentric" or "unique" the composer is with respect to their chord types, and in comparison to the other three composers.
    b. Calculate the chord type distributions for *each song* by a given composer, and use these values to calculate the standard deviation in chord type distributions across the composers' songs. This could be a measure of how "versatile" a composer is in their usage of chords (the greater the standard deviation, the greater the versatility)
    c. Take chord eccentricity vs chord versatility to be the x and y coordinates for each composer, giving each composer a unique point on a scatter plot.

Interpretation.

Observations from listening to music:

- BACH: pure and upbeat, with consistent themes across each fugue. Little to no modern dissonance (i.e. disagreement between notes), unlike other composers.
- CHOPIN: often very fast-paced, with lots of scaling up and down the keyboard (almost in every piece). Most of the chords that he plays are triads, although chords were scarce compared to intervals.
- GRIEG: very wide range of styles and themes across pieces, but consistent themes within each piece. Each song seems to tell its own story.
- DEBUSSY: abundance of grace notes and unique chords, many of which seemed complex and even dissonant, to an extent. Quite the opposite of Bach, in that some of his pieces sounded scattered and at times even chaotic.

Observations from chord type value counts for each composer:

- BACH: most commonly used chord types are major and minor triads, followed by various types of trichords. More "predictable" compared to other composers.
- CHOPIN: most commonly used chords are major and minor triads, followed by sevenths (dominant, diminished). However, we also notice (before filtering chords) that Chopin used significantly more intervals than chords.
- GRIEG: most commonly used chords are major and minor triads, followed by sevenths (dominant, diminished, and minor).
- DEBUSSY: uses less triads compared to other composers, and more "special" chords.

Observations from bar chart of proportions of chord type usage:

- Chopin uses the most octaves, by far, with nearly 25% of "chords" being/containing octaves;
- Debussy uses quite a few tetrachords and trichords;
- Bach is frequent with major & minor triads and trichords;
- Grieg is moderate (surprisingly!), but uses more ninth and major seventh chords;
- Chopin and Grieg are comparable in their use of diminished/dominant sevenths;
- Bach and Debussy both use diminished/dominant sevenths less so;
- The pentatonic chord is more or less unique to Debussy.

Observations from roman numeral value counts and bar chart:

- Bach most common numerals (in order): i, iv, V, ii.
- Grieg's most common numerals: i, V/IV, v, V, ii, iv.
- Debussy's most common numerals: ii, i, V/IV, vi, v.
- Chopin's most common numerals: i, v, V, V/IV, ii, iv.

- Debussy uses significantly more ii, vi, and I chords than the others— in fact, Debussy is supposedly the only composer here whose most commonly used chord is not i, but rather ii. His least common chords are V and iv. This may indicate either one of two things: that either Debussy has a very unique style, or chordify is incorrect in some way. Either way, Debussy is doing something that the other composers are not.
- Chopin is very infrequent with vi and I chords, and uses more octaves than any other composer. Bach, meanwhile, uses the most iv chords.

Observations from chord eccentricity vs chord versatility scatter plot:

- Out of Debussy, Chopin, and Bach, Chopin comes out as the most eccentric and versatile composer, Bach the least eccentric and versatile, and Debussy in the middle. Grieg falls behind all three in chord eccentricity, but his chord versatility rating (across both collections) is somewhere in the middle.

Conclusion.

In summary, we took 5 collections of pieces: Bach's **24 Preludes**, Chopin's **24 Preludes**, Debussy's **24 Preludes**, Grieg's **12 “quality live recordings”** from Katsuhiro Ogirui (labelled "grieg" in the graphs) plus Anitra's Dance, and **18 Grieg "piano rolls converted"** MIDI files (labelled "grieg2" in our graphs); we then produced bar charts for (1) chord type distributions for each of the 5 collections, and (2) roman numeral distributions for each of the 5 collections, as well as a scatterplot that illustrates the "chord eccentricity" for each collection (basically how far the chords deviate from the mean of the 5 collections, i.e. how "different" they are from the average) vs the "chord versatility" (essentially how "varied" the chord type distributions are within each collection of pieces, highest being the most varied).

There were some challenges in implementation, however. In particular, we attempted to “quantize” the music as a means of cleaning up the data for more accurate results, but failed due to poor documentation of Music21. Thus, our results are not extremely accurate and certainly contain some level of bias; but keeping in mind that music is inherently messy to work with, we did what we could with the data we collected. Additionally, not all collections were guaranteed to be comparable, having been taken from different contexts: in particular, while we took 24 preludes from Bach, Chopin, and Debussy, we analyzed selections of live piano recordings and converted piano rolls from Grieg. Nevertheless, the results produced did align somewhat with what we had expected, given the styles of the four authors.

Debussy seemed to stand out with regard to chord types, his chord usage being the most varied and evenly spread across types— which aligns with his reputation for experimentation and harmonic innovation. Chopin’s distributions of chords convey a more concentrated use of specific chord types (in particular, triads and octaves); this

reflects his characteristic style of exploring certain tonal areas deeply, as opposed to broadly, and is supported by the fact that his chord eccentricity rating is the highest by far (i.e. pointing to a more concentrated approach rather than a more evenly distributed one) and that he is known for his highly expressive and emotionally deep pieces. However, his chord versatility rating was also the highest, perhaps contrary to the observation of his "concentrated" approach, but in alignment with his high expressiveness.

Grieg strikes a balance between diversity and focus in terms of chord usage, placing as an intermediate composer here (in both the bar charts, as well as versatility rating). This is somewhat surprising, considering the fact that his music is somewhat flavorful (for example, considering his begin influenced by Norwegian folk music)— but could perhaps be explained by the fact that chord types and tonality do not convey everything about a composer's taste (e.g. not taking into account things like orchestral effects or having a broad range of thematic styles, which could balance out to make him appear "average"), in addition to the fact that Grieg's pieces were contextually very different from Debussy's, Chopin's and Bach's (all of whose are preludes).

Bach's usage of chords is somewhat structured, with some diversity but a degree of predictability: this aligns with the nature of his contrapuntal focus and harmonic purity. Prior to more selectively filtering the chords, we noticed that his chord preferences were not quite as complex as the others. Indeed, his primary chord types are simply triads and trichords.

Out of Debussy, Chopin, and Bach, however, Chopin came out as both the most eccentric and versatile composer. Listening to his pieces, we did notice that they did have a considerable degree of variation, but it was very hard to tell by ear whether his variation across preludes was at all more extreme than that of the other composers.

Overall, Debussy's harmonic richness contrasts with Chopin's more selective and expressive focus, with Bach coming out as a "pure" and somewhat predictable composer, and Grieg falling in the middle. Bach's and Debussy's results (pure vs diverse) were perhaps unsurprising considering where they came into play in the course of musical history; Chopin's chord type profile was somewhat predictable, although his eccentricity and versatility ratings were unexpectedly high. Grieg's appearing somewhat "average" (based on the numbers) was quite surprising, but could potentially be accounted for by the fact that his collection arguably was not suited to be comparable to the others. Chopin's data was overall the hardest to make sense of, with conflicting arguments as to whether his focus was more restrictive, more varied, more unique, or some combination of these.

An Analysis of Chord Usage in Select Classical Composers

Rowan Shigeno, Redd Tan, Holden Starkey

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
from music21 import *
import pandas as pd
import glob
import pickle
import matplotlib.pyplot as plt
import statistics

# function used to remove grace notes:
def remove_grace_notes(midi_file_path):
    # Create a copy of the stream to avoid modifying the original
    modified_stream = stream.Stream(score.flat.notes)
    # List to store grace notes
    grace_notes = []
    # Find grace notes
    for element in modified_stream.recurse():
        if isinstance(element, note.Note) and element.duration.isGrace:
            grace_notes.append(element)
    # Remove grace notes
    for grace_note in grace_notes:
        grace_note.activeSite.remove(grace_note)
    return modified_stream

# function that helps locate intervals of interest
def get_special_intervals(chord):
    if 'Octave' in chord:
        return 'octave'
    elif 'tritone' in chord.lower():
        return 'tritone'
    else:
        return chord

# list of chords that are relevant (used to filer).
# Also includes octave and tritone, as these tell us info about consonace/dissoncance.
our_chords = [
    'octave',
    'tritone',
    "Debussy's heptatonic",
    "Messiaen's truncated mode 6",
    "Scriabin's Mystic-chord",
    "Stravinsky's Petrushka-chord",
    'all-interval tetrachord',
    'alternating hexamirror',
    'alternating pentachord',
    'alternating tetramirror',
```

```
        'alternating tetramirror',
        'Arabian tetramirror',
        'Asian pentacluster',
        'asymmetric pentamirror',
        'Augmented Third',
        'augmented major tetrachord',
        'augmented seventh chord',
        'augmented-diminished ninth chord',
        'Balinese Pelog pentatonic',
        'Balinese pentachord',
        'blues scale',
        'center-cluster pentachord',
        'center-cluster pentamirror',
        'chromatic heptamirror',
        'chromatic tetramirror',
        'chromatic trimirror',
        'diminished minor-ninth chord',
        'diminished pentacluster',
        'diminished triad',
        'diminished-augmented ninth chord',
        'diminished-major ninth chord',
        'diminished seventh chord',
        'dorian hexachord',
        'dorian pentachord',
        'double pentacluster',
        'double-fourth tetramirror',
        'double-phrygian hexachord',
        'dominant seventh chord',
        'dominant-eleventh',
        'dominant-ninth',
        'enigmatic pentachord',
        'flat-ninth pentachord',
        'French augmented sixth chord',
        'French augmented sixth chord in first inversion',
        'French augmented sixth chord in root position',
        'French augmented sixth chord in third inversion',
        'German augmented sixth chord',
        'German augmented sixth chord in root position',
        'German augmented sixth chord in second inversion',
        'German augmented sixth chord in third inversion',
        'Guidonian hexachord',
        'half-diminished seventh chord',
        'harmonic minor scale',
        'harmonic minor tetrachord',
        'Hirajoshi pentatonic',
        'Italian augmented sixth chord',
        'Italian augmented sixth chord in root position',
        'Italian augmented sixth chord in root position in second inversion',
        'Italian augmented sixth chord in second inversion',
        'Javanese pentachord',
        'Javanese pentatonic',
        'Kumoi pentachord',
        'Lebanese pentachord',
        'locrian hexachord',
        'locrian pentachord',
        'lydian pentachord',
        'lydian tetrachord',
```

```
        'major pentachord',
        'major pentatonic',
        'major triad',
        'major-augmented ninth chord',
        'major-diminished tetrachord',
        'major-minor diminished pentachord',
        'major-minor tetramirror',
        'major-minor trichord',
        'major-ninth chord',
        'major-second major tetrachord',
        'major-second minor tetrachord',
        'major-second tetracluster',
        'major seventh chord',
        'Major Seventh',
        'melodic minor ascending scale',
        'melodic-minor hexachord',
        'minor hexachord',
        'minor triad',
        'minor tetramirror',
        'minor-augmented tetrachord',
        'minor-diminished tetrachord',
        'minor-major ninth chord',
        'minor-major tetrachord',
        'minor-major trichord',
        'minor-ninth chord',
        'minor-second diminished tetrachord',
        'minor-second quartal tetrachord',
        'minor seventh chord',
        'Neapolitan pentachord',
        'nonatonic blues',
        'perfect fourth tetramirror',
        'perfect-fourth diminished tetrachord',
        'perfect-fourth major tetrachord',
        'perfect-fourth minor tetrachord',
        'Persian pentamirror',
        'phrygian hexamirror',
        'phrygian pentachord',
        'phrygian tetrachord',
        'phrygian trichord',
        'quartal tetramirror',
        'quartal trichord',
        'Roma (Gypsy) hepatonic',
        'Roma (Gypsy) pentachord',
        'Schoenberg Anagram hexachord',
        'Spanish phrygian',
        'whole tone scale',
        'whole-tone pentachord',
        'whole-tone tetramirror',
        'whole-tone trichord',
    ]

    selective_chords = [
        'octave',
        'tritone',
        'major triad',
        'minor triad',
        'major seventh',
```

```
        'minor seventh',
        'dominant seventh',
        'diminished seventh',
        'half-diminished seventh',
        'trichord',
        'tetrachord',
        'pentachord',
        'hexachord',
        'tetramirror',
        'pentatonic',
        'ninth chord',
        'sixth chord',
        ]

    def get_selective_chords(chord):
        if 'Octave' in chord:
            return 'octave'
        for item in selective_chords:
          if item in chord:
            return item
        else:
            return chord

    def ignore_inversions(roman):
      for num in ['0','1','2','3','4','5','6','7','8','9']:
        if num in roman:
          roman = roman.replace(num, '')
      return roman

    our_numerals = [
        'I', 'i', 'II', 'ii', 'III', 'iii', 'IV', 'iv', 'V', 'v', 'V#', 'V/III',
        'V/IV', 'V/VI', 'V/V', 'V/VII', 'V/vii', 'V/iv', 'V/v', 'VI', 'vi', 'bVI', 'bvi',
        'VII', 'vii', 'bVII', 'bbII', 'bbVI#', 'ib', 'iiø', 'iii#', 'viø', 'viø#', 'v#'
        ]

    selective_numerals = ['I', 'i', 'ii', 'iv', 'V', 'v', 'vi', 'V/IV', "vii", 'III', 'iii', "IV"]

    def get_selective_numerals(roman):
        for item in selective_numerals:
          if item == roman:
            return item
        else:
            return roman


    # create score object from xml file
    score = converter.parse('/content/drive/MyDrive/MUSC255finalcopy/Debussy1/preludes_1_2_(c)galimberti.mid')

    """
    # 'quantize' notes by snapping offsets/durations to nearest multiples of quarter length value
    quantized = score.quantize((4,), inPlace=True, recurse = True)
    """

    # compress the staves into one 'chordified' object
    chord_score = score.chordify()
    chord_score = remove_grace_notes(chord_score)
```

```
# create df of all the chords
list_dicts = []
ky=key.Key("C")

chords = chord_score.flatten().getElementsByClass(['Chord'])

for chord in chords:
    temp_dict = {}
    temp_dict['offset'] = chord.offset
    temp_dict['meas'] = chord.measureNumber
    temp_dict['beat'] = chord.beat
    temp_dict['root'] = chord.root()
    temp_dict['type'] = chord.commonName
    temp_dict['inversion'] = chord.inversion()
    temp_dict['pitches'] = chord.pitches
    temp_dict['roman'] = roman.romanNumeralFromChord(chord, ky).figure
    list_dicts.append(temp_dict)
chord_df = pd.DataFrame(list_dicts)
chord_df.head(10)
```

```
/usr/local/lib/python3.10/dist-packages/music21/midi/translate.py:874: TranslateWarning: Unable to determine instrument from <music21.midi.MidiEvent SEQUENCE_TRA
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/music21/midi/translate.py:874: TranslateWarning: Unable to determine instrument from <music21.midi.MidiEvent SEQUENCE_TRA
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/music21/stream/base.py:3689: Music21DeprecationWarning: .flat is deprecated.  Call .flatten() instead
  return self.iter().getElementsByClass(classFilterList)
```

	offset	meas	beat	root	type	inversion	pitches	roman
0	3.0	2	2.0	E5	Major Third	0	(E5, G#5)	III#3
1	11/3	2	8/3	D5	whole-tone tetramirror	3	(F#5, D5, C5, E5)	II#432
2	3.75	2	2.75	B-4	Major Third	0	(D5, B-4)	bVII#3
3	4.0	3	1.0	C5	Diminished Fourth	2	(G#4, C5)	i+4
4	4.75	3	1.75	C6	Diminished Fourth	2	(G#5, C6)	i+4
5	5.0	3	2.0	B-5	Diminished Fourth	2	(B-5, F#5)	bvii+##84
6	7.0	4	2.0	E5	Major Third	0	(E5, G#5)	III#3
7	23/3	4	8/3	D5	whole-tone tetramirror	3	(D5, F#5, E5, C5)	II#432
8	7.75	4	2.75	B-4	Major Third	0	(B-4, D5)	bVII#3
9	8.0	5	1.0	C5	Diminished Fourth	2	(G#4, C5)	i+4

Next steps: Generate code with chord_df | View recommended plots | New interactive sheet

```
bach=glob.glob("/content/drive/MyDrive/MUSC255finalcopy/Bach1/*")
chopin=glob.glob("/content/drive/MyDrive/MUSC255finalcopy/Chopin1/*")
deb=glob.glob("/content/drive/MyDrive/MUSC255finalcopy/Debussy1/*")
grieg=glob.glob("/content/drive/MyDrive/MUSC255finalcopy/Grieg1/*")
grieg2=glob.glob("/content/drive/MyDrive/MUSC255finalcopy/Grieg2/*")
adams=glob.glob("/content/drive/MyDrive/MUSC255finalcopy/John Adams/*")
tan=glob.glob("/content/drive/MyDrive/MUSC255finalcopy/Tan/*")
```

```
shigeno=glob.glob("/content/drive/MyDrive/MUSC255finalcopy/shigeno/*")
filelist=[bach,chopin,deb,grieg,grieg2,adams,tan,shigeno]


#Get all the files
#Get all the lists of files
allcch={}
composers=["bach","chopin","deb","grieg","grieg2","adams","tan","shigeno"]
overall=[]
indexer=0
for comp in filelist:
  allsch=[]
  overall.append([])
  for song in comp:
    chords=[]
    overall[-1].append(song)
    song=converter.parse(song)
    song.quantize((4,), inPlace=True, recurse = True)
    ky=song.analyze("key")
    songch=song.chordify().flatten().getElementsByClass(["Chord"])
    for ch in songch:
      temp_dict = {}
      temp_dict['offset'] = ch.offset
      temp_dict['meas'] = ch.measureNumber
      temp_dict['beat'] = ch.beat
      temp_dict['root'] = ch.root()
      temp_dict['type'] = ch.commonName
      temp_dict['inversion'] = ch.inversion()
      temp_dict['pitches'] = ch.pitches
      temp_dict['roman'] = roman.romanNumeralFromChord(ch, ky, preferSecondaryDominants=True).figure
      chords.append(temp_dict)
    allsch.append(pd.DataFrame(chords))
  allcch.update({composers[indexer]:allsch})
  indexer+=1


with open("ch","wb") as fp:
  pickle.dump(allcch,fp)
with open("bort","wb") as fp:
  pickle.dump(overall,fp)
print(overall)
```

```
[['/content/drive/MyDrive/MUSC255finalcopy/Bach1/well-tempered-clavier-i_bwv-846_(c)sankey.mid', '/content/drive/MyDrive/MUSC255finalcopy/Bach1/well-tempered-cla
```

```
with open("/content/drive/MyDrive/MUSC255finalcopy/chords","rb") as fp:
  b=pickle.load(fp)
with open("/content/drive/MyDrive/MUSC255finalcopy/song index","rb") as fp:
  a=pickle.load(fp)
```

Double-click (or enter) to edit

```
# get bach dataframe
bach = pd.DataFrame()
for i in b['bach']:
  bach=pd.concat([bach, pd.DataFrame(i)], ignore_index=True)
```

```
# look at value counts of chord types in all of bach pieces
print(a[0])
bach.head(20)

# clean up dataframe, filtering by chords
bach['type'] = bach['type'].apply(get_special_intervals)
bach = bach[bach['type'].isin(our_chords)]
bach['type'].value_counts().head(20)

# filter chords even more
bach['type'] = bach['type'].apply(get_selective_chords)
bach = bach[bach['type'].isin(selective_chords)]
bach['type'].value_counts()
```

```
['/content/drive/MyDrive/MUSC255finalcopy/Bach1/well-tempered-clavier-i_bwv-846_(c)sankey.mid', '/content/drive/MyDrive/MUSC255finalcopy/Bach1/well-tempered-clav
```

type	count
trichord	2005
major triad	1993
minor triad	1664
tetrachord	1590
octave	559
tetramirror	467
dominant seventh	445
diminished seventh	336
sixth chord	293
tritone	289
minor seventh	232
pentachord	210
ninth chord	195
major seventh	164
pentatonic	37
hexachord	19

dtype: int64

```
# get roman numerals, with numbers removed (inversions irrelevant)
bach['roman'] = bach['roman'].apply(ignore_inversions)
bach = bach[bach['roman'].isin(our_numerals)]
bach['roman'].value_counts()

# only focus on most common numerals
```

```
bach['roman'] = bach['roman'].apply(get_selective_numerals)
bach = bach[bach['roman'].isin(selective_numerals)]
bach['roman'].value_counts()
```

```
<ipython-input-7-f269af66bb66>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  bach['roman'] = bach['roman'].apply(get_selective_numerals)
```

roman	count
i	1064
iv	748
V	676
ii	527
V/IV	468
v	448
vi	390
IV	326
iii	321
I	181
III	138
vii	93

dtype: int64

```
"""
grieg1=pd.DataFrame(b['grieg'][0])
print(a[3][0])
grieg1.head()
grieg1['roman'].value_counts()
"""

# get grieg dataframe
grieg = pd.DataFrame()
for i in b['grieg']:
  grieg=pd.concat([grieg, pd.DataFrame(i)], ignore_index=True)

# look at value counts of chord types in all of grieg pieces
print(a[3])
grieg.head(20)

# clean up dataframe
grieg['type'] = grieg['type'].apply(get_special_intervals)
grieg = grieg[grieg['type'].isin(our_chords)]
grieg['type'].value_counts().head(20)
```

```
# filter chords even more
grieg['type'] = grieg['type'].apply(get_selective_chords)
grieg = grieg[grieg['type'].isin(selective_chords)]
grieg['type'].value_counts()
```

```
['/content/drive/MyDrive/MUSC255finalcopy/Grieg1/grieg_peer_gynt_1_anitras_dans_(c)icking-archive.mid', '/content/drive/MyDrive/MUSC255finalcopy/Grieg1/grieg_pie
```

type	count
major triad	661
octave	470
minor triad	467
tetrachord	401
dominant seventh	218
trichord	203
diminished seventh	188
minor seventh	142
ninth chord	139
tritone	88
major seventh	84
pentachord	69
sixth chord	57
tetramirror	56
pentatonic	17
hexachord	5

dtype: int64

```
# get roman numerals, with numbers removed (inversions irrelevant)
grieg['roman'] = grieg['roman'].apply(ignore_inversions)
grieg = grieg[grieg['roman'].isin(our_numerals)]
grieg['roman'].value_counts()

# only focus on most common numerals
grieg['roman'] = grieg['roman'].apply(get_selective_numerals)
grieg = grieg[grieg['roman'].isin(selective_numerals)]
grieg['roman'].value_counts()
```

```
<ipython-input-9-63a7929a8f91>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  grieg['roman'] = grieg['roman'].apply(get_selective_numerals)
```

roman	count
i	346
V/IV	201
v	199
V	171
ii	166
iv	144
vi	134
iii	130
I	59
IV	50
III	33
vii	24

dtype: int64

```
"""deb1=pd.DataFrame(b['deb'][0])
print(a[2][0])
deb1.head()
deb1['type'].value_counts().head(10)"""

# get debussy dataframe
deb = pd.DataFrame()
for i in b['deb']:
  deb=pd.concat([deb, pd.DataFrame(i)], ignore_index=True)

# look at value counts of chord types in all of deb pieces
print(a[2])
deb.head(20)

# clean up dataframe
deb['type'] = deb['type'].apply(get_special_intervals)
deb = deb[deb['type'].isin(our_chords)]
deb['type'].value_counts()

# filter chords even more
deb['type'] = deb['type'].apply(get_selective_chords)
deb = deb[deb['type'].isin(selective_chords)]
deb['type'].value_counts()
```

```
['/content/drive/MyDrive/MUSC255finalcopy/Debussy1/preludes_1_1_(c)galimberti.mid', '/content/drive/MyDrive/MUSC255finalcopy/Debussy1/preludes_1_2_(c)galimberti.
```

type	count
tetrachord	914
trichord	671
major triad	486
pentachord	460
octave	363
tetramirror	265
minor triad	217
minor seventh	180
pentatonic	154
sixth chord	144
dominant seventh	128
tritone	128
diminished seventh	124
ninth chord	79
hexachord	63
major seventh	41

dtype: int64

```
# get roman numerals, with numbers removed (inversions irrelevant)
deb['roman'] = deb['roman'].apply(ignore_inversions)
deb = deb[deb['roman'].isin(our_numerals)]
deb['roman'].value_counts().head(20)

# only focus on most common numerals
deb['roman'] = deb['roman'].apply(get_selective_numerals)
deb = deb[deb['roman'].isin(selective_numerals)]
deb['roman'].value_counts()
```

```
<ipython-input-11-9407f4ec3d5d>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  deb['roman'] = deb['roman'].apply(get_selective_numerals)
```

roman	count
ii	193
i	175
V/IV	139
vi	138
v	129
iii	115
IV	104
I	77
V	70
iv	55
vii	27
III	27

dtype: int64

```
"""chopin1=pd.DataFrame(b['chopin'][0])
chopin1.head()
chopin1['type'].value_counts().head(10)"""

# get chopin dataframe
chopin = pd.DataFrame()
for i in b['chopin']:
  chopin=pd.concat([chopin, pd.DataFrame(i)], ignore_index=True)

# look at value counts of chord types in all of chopin pieces
print(a[1])
chopin.head()

# clean up dataframe
chopin['type'] = chopin['type'].str.replace('enharmonic equivalent to ', '')
chopin['type'] = chopin['type'].str.replace(' with octave doublings', '')
chopin = chopin[chopin['type'] != 'note']
chopin['type'].value_counts()

# filter chords even more
chopin['type'] = chopin['type'].apply(get_selective_chords)
chopin = chopin[chopin['type'].isin(selective_chords)]
chopin['type'].value_counts()
```

```
['/content/drive/MyDrive/MUSC255finalcopy/Chopin1/1.mxl', '/content/drive/MyDrive/MUSC255finalcopy/Chopin1/2.mxl', '/content/drive/MyDrive/MUSC255finalcopy/Chopi
<ipython-input-12-6b7247a39601>:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  chopin['type'] = chopin['type'].apply(get_selective_chords)
```

type	count
octave	997
major triad	836
minor triad	544
tetrachord	338
dominant seventh	289
trichord	237
diminished seventh	229
tetramirror	150
minor seventh	113
pentachord	85
major seventh	66
tritone	51
ninth chord	48
pentatonic	26
sixth chord	22
hexachord	1

dtype: int64

```
# get roman numerals, with numbers removed (inversions irrelevant)
chopin['roman'] = chopin['roman'].apply(ignore_inversions)
chopin = chopin[chopin['roman'].isin(our_numerals)]
chopin['roman'].value_counts().head(20)

# only focus on most common numerals
chopin['roman'] = chopin['roman'].apply(get_selective_numerals)
chopin = chopin[chopin['roman'].isin(selective_numerals)]
chopin['roman'].value_counts()
```

```
<ipython-input-13-ac4e9a1eedf6>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  chopin['roman'] = chopin['roman'].apply(get_selective_numerals)
```

roman	count
i	495
v	294
V	284
V/IV	251
ii	250
iv	208
iii	153
IV	135
vi	84
vii	57
I	54
III	26

dtype: int64

```
# get grieg dataframe
grieg2 = pd.DataFrame()
for i in b['grieg2']:
  grieg2=pd.concat([grieg2, pd.DataFrame(i)], ignore_index=True)

# look at value counts of chord types in all of grieg pieces
print(a[4])
grieg2.head(20)

# clean up dataframe
grieg2['type'] = grieg2['type'].apply(get_special_intervals)
grieg2 = grieg2[grieg2['type'].isin(our_chords)]
grieg2['type'].value_counts().head(20)

# filter chords even more
grieg2['type'] = grieg2['type'].apply(get_selective_chords)
grieg2 = grieg2[grieg2['type'].isin(selective_chords)]
grieg2['type'].value_counts()
```

```
['/content/drive/MyDrive/MUSC255finalcopy/Grieg2/59.MID', '/content/drive/MyDrive/MUSC255finalcopy/Grieg2/143.mid', '/content/drive/MyDrive/MUSC255finalcopy/Grie
```

type	count
major triad	919
minor triad	626
tetrachord	576
trichord	450
octave	398
minor seventh	204
major seventh	152
tetramirror	134
diminished seventh	120
pentachord	99
dominant seventh	88
ninth chord	56
pentatonic	43
sixth chord	36
tritone	8
hexachord	8

dtype: int64

```
# get roman numerals, with numbers removed (inversions irrelevant)
grieg2['roman'] = grieg2['roman'].apply(ignore_inversions)
grieg2 = grieg2[grieg2['roman'].isin(our_numerals)]
grieg2['roman'].value_counts().head(20)

# only focus on most common numerals
grieg2['roman'] = grieg2['roman'].apply(get_selective_numerals)
grieg2 = grieg2[grieg2['roman'].isin(selective_numerals)]
grieg2['roman'].value_counts()
```

```
<ipython-input-15-9598ae1ab3ea>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  grieg2['roman'] = grieg2['roman'].apply(get_selective_numerals)
```

roman	count
i	505
ii	254
V/IV	251
v	230
vi	205
iv	144
V	122
I	95
IV	69
iii	49
III	27
vii	14

dtype: int64

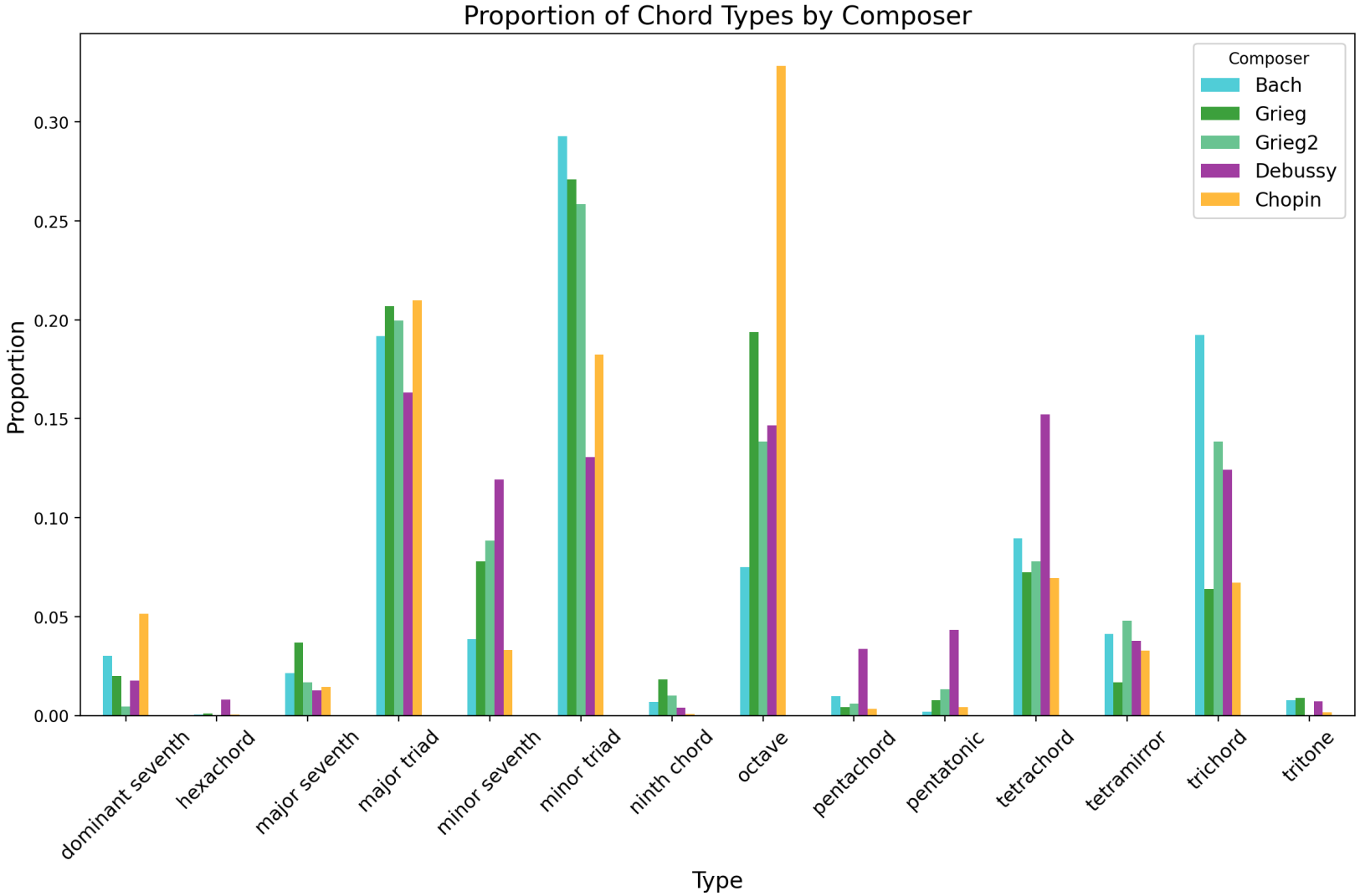
```
c2=b.keys()
print(c2)

for composer in c2:
  for i in range(len(b[composer])):
    df=pd.DataFrame(b[composer][i])
    df["piece"]=f"composer"

"""
    list_dicts = []
    for chord in b[i][j]:
      temp_dict = {}
      temp_dict['offset'] = chord.offset
      temp_dict['meas'] = chord.measureNumber
      temp_dict['beat'] = chord.beat
      temp_dict['root'] = chord.root()
      temp_dict['type'] = chord.commonName
      temp_dict['inversion'] = chord.inversion()
      temp_dict['pitches'] = chord.pitches
      temp_dict['roman'] = roman.romanNumeralFromChord(chord, ky).figure
      list_dicts.append(temp_dict)
    chord_df = pd.DataFrame(list_dicts)
    b[i][j]=chord_df
with open("ch2","wb") as fp:
```

```
  pickle.dump(b,fp)
"""
```

```
dict_keys(['bach', 'chopin', 'deb', 'grieg', 'grieg2', 'adams', 'tan', 'shigeno'])
'\n    list_dicts = []\n    for chord in b[i][j]:\n      temp_dict = {}\n      temp_dict[\'offset\'] = chord.offset\n      temp_dict[\'meas\'] = chord.measureNum
ber\n      temp_dict[\'beat\'] = chord.beat\n      temp_dict[\'root\'] = chord.root()\n      temp_dict[\'type\'] = chord.commonName\n      temp_dict[\'inversion
\'] = chord.inversion()\n      temp_dict[\'pitches\'] = chord.pitches\n      temp_dict[\'roman\'] = roman.romanNumeralFromChord(chord, ky).figure\n      list_dic
ts.append(temp_dict)\n    chord_df = pd.DataFrame(list_dicts)\n    b[i][j]=chord_df\nwith open("ch2","wb") as fp:\n  pickle.dump(b,fp)\n'
```

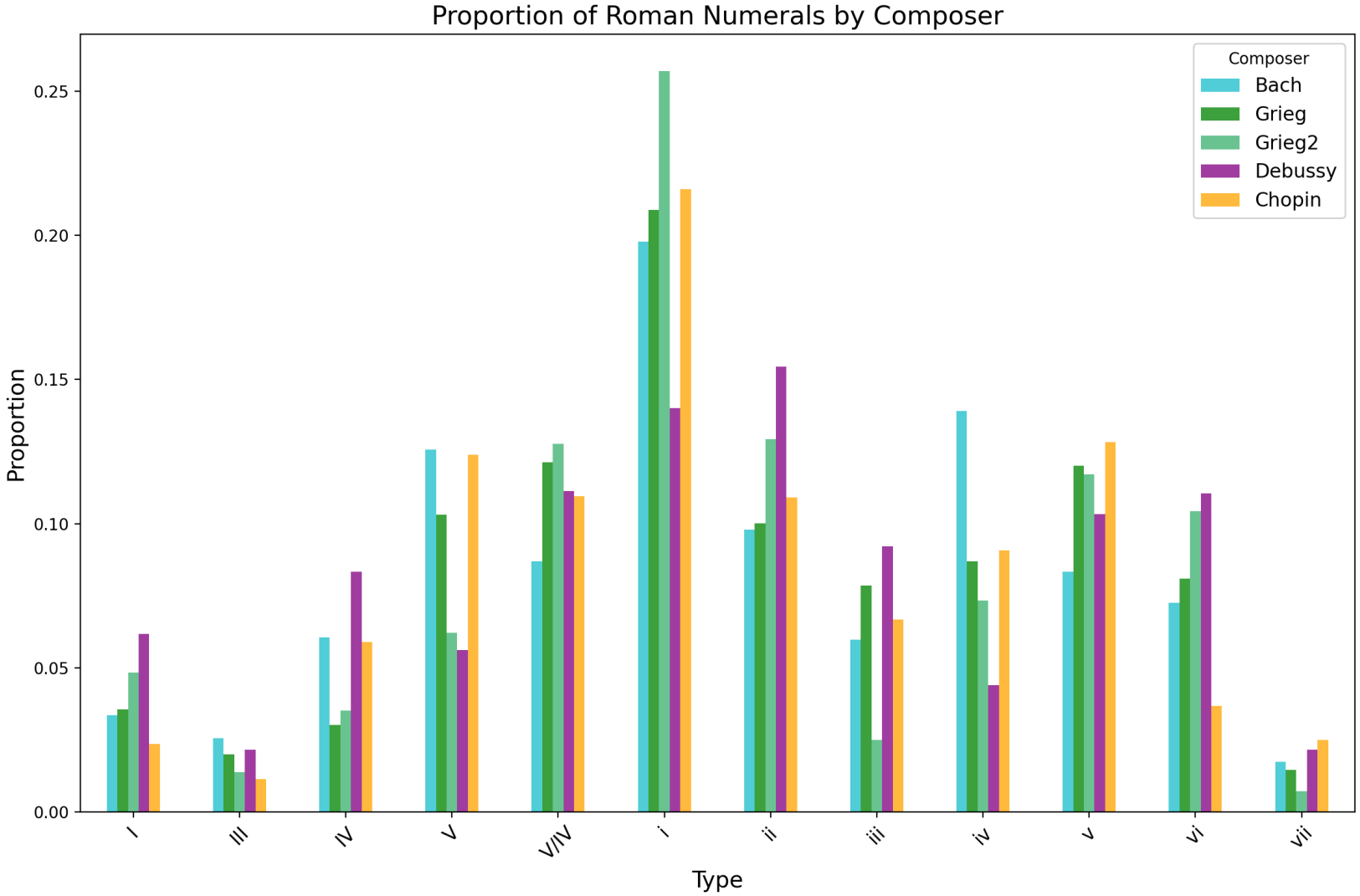
```
print(b['bach'][0]["meas"])
```

```
# chord type proportions for each composer
bach_ch = bach['type'].value_counts(normalize=True)  # Normalize to get proportions
grieg_ch = grieg['type'].value_counts(normalize=True)
grieg2_ch = grieg2['type'].value_counts(normalize=True)
deb_ch = deb['type'].value_counts(normalize=True)
chopin_ch = chopin['type'].value_counts(normalize=True)

# combine the proportions into a single DataFrame
df_combined = pd.DataFrame({
    'Bach': bach_ch,
    'Grieg': grieg_ch,
    'Grieg2': grieg2_ch,
    'Debussy': deb_ch,
    'Chopin': chopin_ch
})

# create bar plot of chord usage proportions for composers
df_combined.plot(
    kind='bar',
    figsize=(12, 8),
    color=['tab:cyan', 'green', 'mediumseagreen', 'purple', 'orange'],
    alpha=0.75
)

# add labels and title
plt.title('Proportion of Chord Types by Composer', fontsize=16)
plt.xlabel('Type', fontsize=14)
plt.ylabel('Proportion', fontsize=14)
plt.xticks(rotation=45, fontsize=12)
plt.legend(title='Composer', fontsize=12)
plt.tight_layout()

# show plot
plt.show()
```




```
# roman numeral proportions for each composer
bach_roman = bach['roman'].value_counts(normalize=True)  # Normalize to get proportions
grieg_roman = grieg['roman'].value_counts(normalize=True)
grieg2_roman = grieg2['roman'].value_counts(normalize=True)
deb_roman = deb['roman'].value_counts(normalize=True)
chopin_roman = chopin['roman'].value_counts(normalize=True)

# combine the proportions into a single DataFrame
df_combined = pd.DataFrame({
    'Bach': bach_roman,
    'Grieg': grieg_roman,
    'Grieg2': grieg2_roman,
    'Debussy': deb_roman,
```

```
        'Chopin': chopin_roman
    })

    # create bar plot of chord usage proportions for composers
    df_combined.plot(
        kind='bar',
        figsize=(12, 8),
        color=['tab:cyan', 'green', 'mediumseagreen', 'purple', 'orange'],
        alpha=0.75
    )

    # add labels and title
    plt.title('Proportion of Roman Numerals by Composer', fontsize=16)
    plt.xlabel('Type', fontsize=14)
    plt.ylabel('Proportion', fontsize=14)
    plt.xticks(rotation=45, fontsize=12)
    plt.legend(title='Composer', fontsize=12)
    plt.tight_layout()

    # show plot
    plt.show()
```




```
# calculate chord versatility scores
verslist = []
complist = ['bach', 'chopin', 'deb', 'grieg', 'grieg2']
for comp in complist:
  # create list of dataframes of proportions of chord types for each piece
  ratiolist = []
  for i in range(0, len(b[comp])):
    piece = pd.DataFrame(b[comp][i])
    piece['type'] = piece['type'].apply(get_special_intervals)
    piece = piece[piece['type'].isin(our_chords)]
    piece['type'] = piece['type'].apply(get_selective_chords)
    piece = piece[piece['type'].isin(selective_chords)]
    chordratios = piece['type'].value_counts(normalize=True)
```

```
        ratiolist.append(chordratios)
      # array of proportion values (row corresponds to chord type, column correspond to song)
      ratioarray = []
      for ch in selective_chords:
        songindex = 0
        this = []
        for song in ratiolist:
          if ch in song:
            this.append(ratiolist[songindex][ch])
          else:
            this.append(0)
          songindex += 1
        ratioarray.append(this)
      # list of standard deviations for proportions of each chord type across songs
      stdevs = []
      for this in ratioarray:
        stdevs.append(statistics.stdev(this))
      # chord versatility score for this composer
      mean = statistics.mean(stdevs)
      verslist.append(mean)

    # calculate chord eccentricity scores
    print('')
    avg_chord_proportions = []
    for ch in selective_chords:
      props = []
      for comp in [bach_ch, chopin_ch, deb_ch, grieg_ch, grieg2_ch]:
        if ch in comp:
          props.append(comp[ch])
        else:
          props.append(0)
      avg_chord_proportions.append(statistics.mean(props))
```

```
Composers:  ['bach', 'chopin', 'deb', 'grieg', 'grieg2']
Chord versatility scores:  [0.036589568705980326, 0.06472348079106346, 0.0570312689495354, 0.041419565885407385, 0.05938754436925293]
Chord eccentricity scores:  [0.018365378444354156, 0.022851151647627292, 0.020730472549868893, 0.013016902634587667, 0.010549948347687622]
```